Advances in Engineering Software xxx (2015) xxx–xxx

Contents lists available at ScienceDirect

Advances in Engineering Software

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

Balancing throughput and response time in online scientific Clouds via Ant Colony Optimization (SP2013/2013/00006)

Elina Pacini [a], Cristian Mateos [b,c,*], Carlos García Garino [a,d]

[a] ITIC – UNCuyo University, Mendoza, Argentina
[b] ISISTAN Research Institute, UNICEN University, Campus Universitario, Tandil B7001BBO, Argentina
[c] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Av. Rivadavia 1917, CABA (C1033AAJ), Argentina
[d] Facultad de Ingeniería – UNCuyo University, Mendoza, Argentina

A R T I C L E I N F O

Article history:
Available online xxxx

Keywords:
Cloud Computing
Scientific problems
Job scheduling
Swarm intelligence
Ant Colony Optimization
Genetic Algorithms

A B S T R A C T

The Cloud Computing paradigm focuses on the provisioning of reliable and scalable infrastructures (Clouds) delivering execution and storage services. The paradigm, with its promise of virtually infinite resources, seems to suit well in solving resource greedy scientific computing problems. The goal of this work is to study private Clouds to execute scientific experiments coming from multiple users, i.e., our work focuses on the Infrastructure as a Service (IaaS) model where custom Virtual Machines (VM) are launched in appropriate hosts available in a Cloud. Then, correctly scheduling Cloud hosts is very important and it is necessary to develop efficient scheduling strategies to appropriately allocate VMs to physical resources. The job scheduling problem is however NP-complete, and therefore many heuristics have been developed. In this work, we describe and evaluate a Cloud scheduler based on Ant Colony Optimization (ACO). The main performance metrics to study are the number of serviced users by the Cloud and the total number of created VMs in online (non-batch) scheduling scenarios. Besides, the number of intra-Cloud network messages sent are evaluated. Simulated experiments performed using CloudSim and job data from real scientific problems show that our scheduler succeeds in balancing the studied metrics compared to schedulers based on Random assignment and Genetic Algorithms.

1. Introduction

Scientific computing is a field of study that applies computer science to solve typical scientific problems in disciplines such as Bioinformatics [44], Earth Sciences [23], High-Energy Physics [7], Molecular Science [53] and even Social Sciences [5]. Scientific computing is usually associated with large-scale computer modeling and simulation, and often requires large amounts of computer resources to satisfy the ever-increasing resource intensive nature of its experiments. An example of these experiments is parameter sweep experiments (PSEs), which we have extensively described in previous works [19,30,36].

Cloud Computing [11] is a paradigm which suits well in solving the above cited computing problems, because of its promise of provisioning infinite resources. Within a Cloud, resources can be effectively and dynamically managed using virtualization technologies. Cloud Computing comes in three flavors: infrastructure, platform, and software as services. In commercial Clouds, these services are made available to customers on a subscription basis using pay-as-you-use models. Although the use of Clouds finds its roots in IT environments, the idea is gradually entering scientific and academic ones [37].

Currently, there are several commercial Clouds that offer computing/storage resources, platform-level services or applications. Moreover, it is possible to build private Clouds (i.e., intra-datacenter) using open-source Cloud Computing solutions. This work is focused on the Infrastructure as a Service (IaaS) model, where physical resources are exposed as services. Under this model, users request virtual machines (VM) to the Cloud, which are then associated to physical resources. However, in order to achieve the best performance, VMs have to fully utilize the physical resources by adapting to the Cloud environment dynamically. To perform this, scheduling the processing units of a Cloud (hosts) is an important issue and it is necessary to develop efficient scheduling strategies to appropriately allocate the VMs in physical resources. Here, *scheduling* refers to the way VMs are allocated to run on the

* Corresponding author at: ISISTAN Research Institute, UNICEN University, Campus Universitario, Tandil B7001BBO, Argentina. Tel.: +54 (249) 4439682x35; fax: +54 (249) 4439681.

E-mail addresses: epacini@itu.uncu.edu.ar (E. Pacini), cmateos@conicet.gov.ar (C. Mateos), cgarcia@itu.uncu.edu.ar (C. García Garino).

http://dx.doi.org/10.1016/j.advengsoft.2015.01.005
0965-9978/© 2015 Elsevier Ltd. All rights reserved.

available computing resources, since there are typically many more VMs running than physical resources. The VM allocation is responsibility of a software component called *scheduler*. However, scheduling is an NP-complete [52] problem and therefore it is not trivial from an algorithmic perspective. In this context, scheduling may also refer to two goals, namely delivering efficient high performance computing or supporting high throughput computing. High performance computing (HPC) focuses on decreasing job execution time whereas high throughput computing (HTC) aims at increasing the processing capacity of the system. As will be shown, the studied ACO scheduler attempts to balance both aspects.

Swarm Intelligence (SI) metaheuristics have been suggested as interesting techniques to solve combinatorial optimization problems – e.g., job scheduling – by simulating the collective behavior of social insects swarms [10]. Within these, the ACO metaheuristic proposed by Marco Dorigo [16] was inspired by the ability of real ant colonies to efficiently organize the foraging behavior of the colony using external chemical pheromone trails for communication. Since then, ACO algorithms have been widely used for solving many combinatorial optimization problems [17], many of them closely related to the problem at hand. A review of the literature about the uses of ACO algorithms for scheduling problems can be found in the work of Tavares Neto and Godinho Filo [46]. Moreover, since scheduling in Clouds is also a combinatorial optimization problem, some schedulers in this line that exploit ACO have been surveyed in our previous work [35]. In this paper, we describe a scheduler based on ACO to allocate VMs to physical Cloud resources.

Unlike previous work of our own [19,30], the aim of this paper is to experiment with the ACO scheduler in an online Cloud (non-batch) scenario in which multiple users connect to the Cloud at different times to execute their PSEs. In this paper, by extending the preliminary results first reported in a previous work presented at the Pareng 2013 Conference [36], we have deepened the experimental analysis by incorporating two new pure HTC and HPC scenarios. Moreover, we measure network resources consumed by the scheduler and its competitors when handling VM requests issued by users.

Experiments have been conduced in order to evaluate the trade-off between the number of serviced users (which relates to throughput) among all users that are connected to the Cloud, and the total number of VMs that are allocated by the scheduler (which relates to response time). The more the users served, the more the executed PSEs, and hence throughput increases. Moreover, when more VMs can be allocated, more physical resources can be taken advantage of, and hence PSE execution time decreases. The main performance metric to study in this paper is a weighted metric in which the results obtained from different scheduling algorithms have been normalized and weighted in order to determine, from the evaluated algorithms, which one better balances the aforementioned metrics. For this, two weights have been assigned to the individual metrics, i.e., a weigh for the number of serviced users (*weightSU*) and a weight for the number of created VMs (*weightVMs*). Each pair of weight combinations (weightSU, weightVMs) represent a different scenario. In this paper we evaluate two pure HTC and HPC scenarios by assigning the weight combinations (1, 0) and (0, 1), and a mixed HTC/HPC scenario by assigning weights (0.5, 0.5) with the aim of balancing these two basic metrics.

In addition, similarly to the preliminary results reported in [36], we study how the number of serviced users and created VMs is affected when using an exponential back-off strategy to retry allocating failing VMs. Experiments were performed with job data obtained from a real-world PSE [21] based on 3D finite element study whereas our previous results [19,30,36] were computed from 2D finite element simulations. In computational terms, this problem led to much more computing intensive jobs. It is worth mentioning that we have deliberately included some of the explanations from [36], specially the description of our ACO scheduler, so as to make this paper self-contained.

The comparisons have been performed against alternative Cloud schedulers, namely a Random allocation algorithm and a Cloud scheduler based on Genetic Algorithms [1]. Results show that our ACO scheduler performs competitively with respect to the number of serviced users and allows for a fair assignment of VMs. In other words, our scheduler provides a good balance to the number serviced users, i.e., the number of Cloud users that the scheduler is able to successfully serve, and the created VMs. The common ground for comparison is an ideal scheduler that always achieves the best possible allocation of VMs to physical resources according to these metrics. Experiments were performed by using CloudSim [12], a Cloud simulator that is widely employed for assessing Cloud schedulers.

The rest of the paper is structured as follows. Section 2 gives some background necessary to understand the concepts underpinning our scheduler. Then, Section 3 presents the scheduler. Section 4 reports the experimental evaluation. Then, Section 5 surveys relevant related works. Lastly, Section 6 concludes the paper and delineates future research opportunities.

2. Background

Cloud Computing [11] is a computing paradigm that has been recently incepted in the academic community [4]. Within a Cloud, *services* that represent computing resources, platforms or applications are provided across (sometimes geographically) dispersed organizations. Moreover, a Cloud provides resources in a highly dynamic and scalable way and offers to end-users a variety of services covering the entire computing stack. Particularly, within IaaS Clouds, slices of computational power in networked hosts are offered with the intent of reducing the owning and operating costs of having such resources in situ. Besides, the spectrum of configuration options available to scientists, such as PSEs scientific users, through Cloud services is wide enough to cover any specific need from their research.

2.1. Cloud Computing basics

The growing popularity of Cloud Computing has led to several definitions, as previously indicated by Vaquero et al. [48]. Some of the definitions given by scientists in the area include:

- Buyya et al. [11] define Cloud Computing in terms of its utility to end users: ‘‘A Cloud is a market-oriented distributed computing system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers’’.
- On the other hand, Mell and Grance [32] define Cloud Computing as ‘‘a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (i.e. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This Cloud model is composed of five essential characteristics, three services models (Software/Platform/Infrastructure as a Service), and four deployment models, whereas the five characteristics are: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured services. The deployment models include private, community, public and hybrid Clouds’’.

As suggested, central to Cloud Computing is the concept of *virtualization*, i.e., the capability of a software system of emulating various operating systems. In a Cloud, virtualization is an essential mechanism for providing resources flexibly to each user and isolating security and stability issues from other users. Clouds allow the dynamic scaling of users applications by the provisioning of computing resources via *machine images*, or VMs. In addition, users can customize the execution environments or installed software in the VMs according to the needs of their experiments.

Virtualization technologies allows a Cloud infrastructure to remap VMs to physical resources according to the change in resources load [43]. In order to achieve good performance, VMs have to fully utilize its services and resources by adapting to the Cloud Computing environment dynamically. Proper allocation of resources must be guaranteed in order to improve resource utility [14].

A Cloud offer its services according to three fundamental models [49] as shown in Fig. 1: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS is the most basic but at the same time ubiquitous model in which an IT infrastructure is deployed in a datacenter as VMs. With the growing popularity of IaaS Clouds, many tools and technologies are emerging, which can transform an organization's existing infrastructure into a private (intra-datacenter) or hybrid Cloud. An IaaS Cloud enables on-demand provisioning of computational resources in the form of VMs deployed in a datacenter, minimizing or even eliminating associated capital costs for Cloud consumers, and letting those consumers add or remove capacity from their IT infrastructure to meet peak or fluctuating service demands. Moreover, PaaS implementations supply users with an application framework and APIs that can be used to program or compose applications for the Cloud. Finally, SaaS is a software delivery model that provides end users with an integrated service comprising hardware, development platforms, and applications. Users are not allowed to customize the service but to get access to a specific application hosted in the Cloud.

On an IaaS environment, unlike traditional job scheduling (e.g., on clusters) where the executing units are mapped directly to physical resources at one level (execution middleware), resources are scheduled at two levels (Fig. 2): Cloud-wide or Infrastructure-level, and VM-level. At the Cloud-wide level, one or more Cloud infrastructures are created and through a VM scheduler the VMs are allocated into real hardware. Then, at the VM-level, by using job scheduling techniques, jobs are assigned for execution

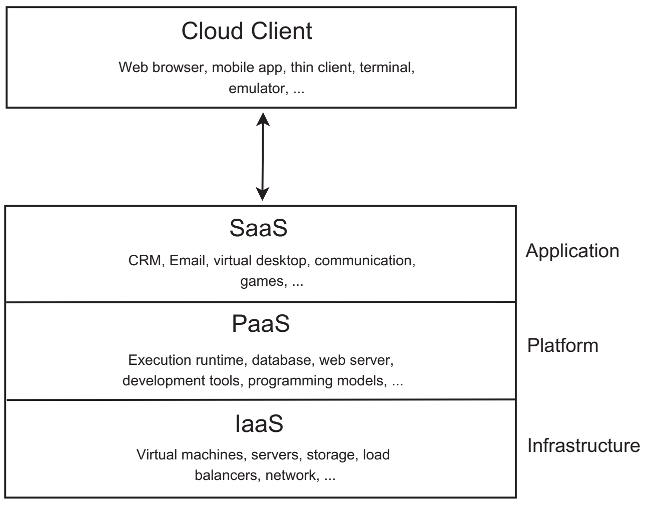
Cloud Client
Web browser, mobile app, thin client, terminal, emulator, ...
SaaS
CRM, Email, virtual desktop, communication, games, ...
PaaS
Execution runtime, database, web server, development tools, programming models, ...
IaaS
Virtual machines, servers, storage, load balancers, network, ...
Application
Platform
Infrastructure

Fig. 1. Cloud computing offerings by services.

into virtual resources. Broadly, job scheduling is a mechanism that maps jobs to appropriate resources to execute, and the delivered efficiency will directly affect the performance of the whole distributed environment. Furthermore, Fig. 2 illustrates a Cloud where one or more scientific users are connected via a network and require the creation of a number of VMs for executing their experiments (a set of jobs).

For scientific applications in general, virtualization has shown to provide many useful benefits, including user-customization of system software and services, check-pointing and migration, better reproducibility of scientific analyses, and enhanced support for legacy applications [25]. The value of Cloud Computing as a tool to execute complex scientific applications in general [50,51] has been already recognized within the scientific community. Although the use of Cloud infrastructures helps scientific users to run complex applications, job and VM management is a key concern that must be addressed. Particularly, in this work we focus on the Infrastructure-level in order to more efficiently solve the allocation of VMs to physical resources in an online, multi-user Cloud. However, job scheduling is NP-complete [52], and therefore approximation heuristics are necessary.

2.2. Swarm Intelligence (SI) techniques for Cloud scheduling

SI techniques [10] are increasingly used to solve optimization problems, and thus they result good alternatives to achieve the goals proposed in this work. SI is a discipline that deals with natural and artificial systems composed of many individuals that coordinate themselves using decentralized control and self-organization. In particular, SI focuses on the collective behaviors that result from the local interactions of the individuals with each other and with their environment. Examples of systems studied by SI are ants colonies, fish schools, flocks of birds, and herds of land animals, where the whole group of agents perform a desired task (i.e. feeding), which might not be made individually. The advantage of these techniques derives from their ability to explore solutions in large search spaces in a very efficient way along with little initial information. Moreover, using SI techniques is an interesting approach to cope in practice with the NP-completeness of job scheduling [35,46]. In particular, the great performance of Ant Colony Optimization (ACO) algorithms for job scheduling problems was first shown in [33].

ACO [16] arises from the way real ants behave in nature. Real ants initially wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep traveling at random, but to follow the trail instead, returning and reinforcing it if they eventually find food. Thus, when one ant finds a short path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leaves all the ants following a single path. However, if over time ants do not visit a certain path, pheromone trails start to evaporate, thus reducing their attractive strength. The more the time an ant needs to travel down the path and back again, the less the pheromone trails are reinforced. From an algorithmic point of view, the pheromone evaporation process is useful for avoiding the convergence to a local optimum solution.

Fig. 3 shows two possible paths from the nest to the food source, but one of them is longer than the other one. Fig. 3(a) shows how ants will start moving randomly at the beginning to explore the ground and then choose one of two paths. The ants that follow the shorter path will naturally reach the food source before the others ants, and in doing so the former group of ants will leave behind them a pheromone trail. After reaching the food, the ants will turn back and try to find the nest. Moreover, the ants that perform the round trip faster, strengthen more quickly the quantity of pheromone in the shorter path, as shown in Fig. 3(b). The ants that

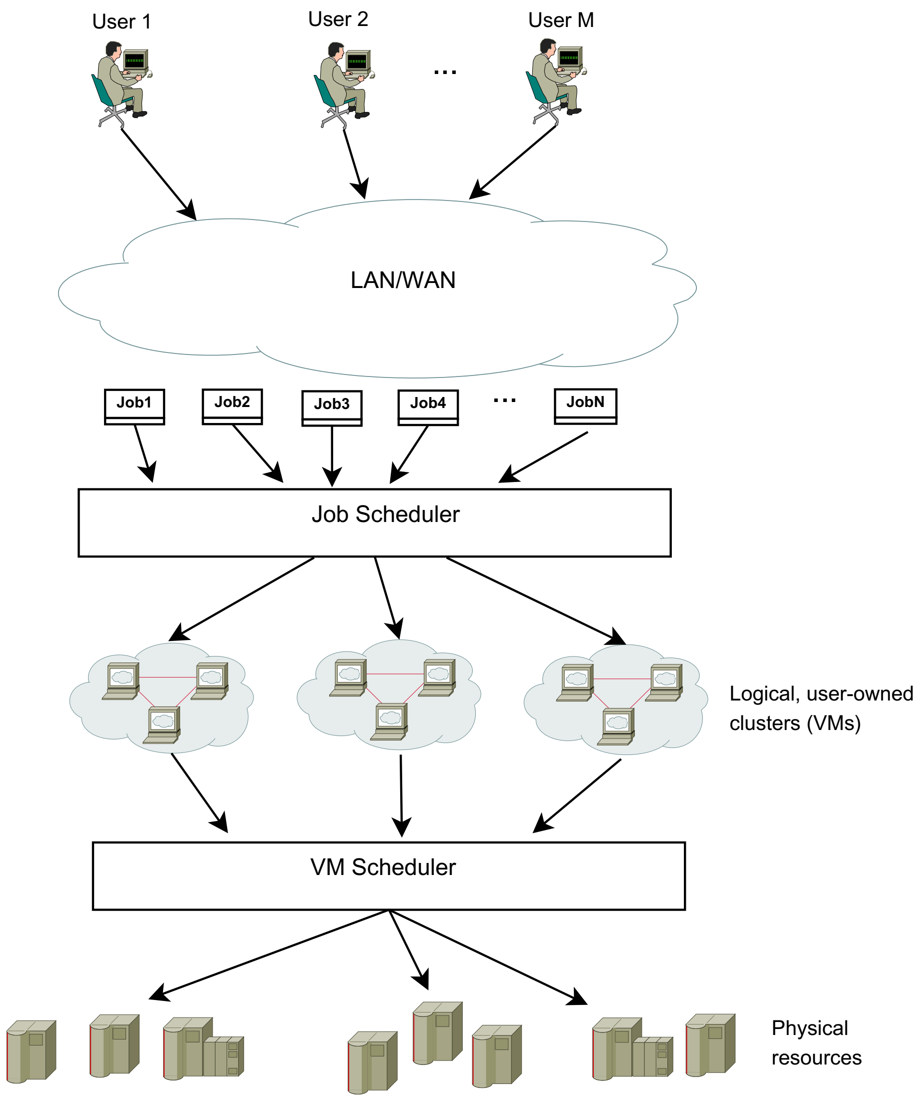

Fig. 2. High-level view of a Cloud.

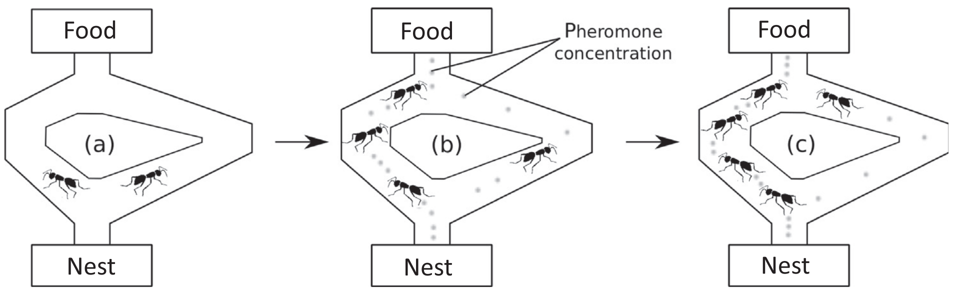

Fig. 3. Adaptive behavior of ants.

reach the food source through the slower path will find attractive to return to the nest using the shortest path. Eventually, most ants will choose the left path as shown in Fig. 3(c).

The above behavior of real ants has inspired ACO. One of its main ideas is exploiting the indirect communication among the individuals of an ant colony. Intuitively, this mechanism is based on an analogy with the above mentioned trails of pheromone which real ants use for communication. ACO employs pheromone trails as a kind of distributed numerical information which is modified by ants to reflect their accumulated experience while solving a particular problem. At each execution step, ants compute a set of feasible moves and select the best one (according to some probabilistic rules) to carry out all the tour. The transition probability for moving from a place to another is based on the heuristic information and pheromone trail level of the move. The higher the value of the pheromone and the heuristic information, the more profitable it is to select this move and resume the search.

All ACO algorithms adapt the algorithm scheme explained next. After initializing the pheromone trails and control parameters, a main loop is repeated until a stopping criterion is met (e.g., a certain number of iterations to perform or a given time limit without improving the result). In this loop, ants construct feasible solutions and update the associated pheromone trails. Furthermore, partial problem solutions are seen as *nodes* (an abstraction for the location of an ant): each ant starts to travel from a random node and moves from a node i to another node j of the partial solution. At each step, the ant k computes a set of feasible solutions to its current node and moves according to a probability distribution. For an ant k the probability p_{ij}^k to move from a node i to a node j is:

$$\begin{cases} p_{ij}^k = \frac{\tau_{ij}\cdot\eta_{ij}}{\sum_{q\in allowed_k}\tau_{iq}\eta_{iq}} & \text{if } j \in allowed_k \\ p_{ij}^k = 0 & otherwise \end{cases} \quad (1)$$

where η_{ij} is the attractiveness of the move as computed by some heuristic information indicating a prior desirability of that move. τ_{ij} is the pheromone trail level of the move, indicating how profitable it has been in the past to make that particular move (it represents therefore a posterior indication of the desirability of that move). Finally, $allowed_k$ is the set of remaining feasible nodes.

The higher the pheromone value and the heuristic information, the more profitable it is to include j in the partial solution. The initial pheromone level is a positive integer τ_0. In nature, there is not any pheromone on the ground at the beginning (i.e., $\tau_0 = 0$). However, the ACO algorithm requires $\tau_0 > 0$, otherwise the probability to chose the next state would be $p_{ij}^k = 0$ and the search process would stop from the beginning. Furthermore, the pheromone level of the elements of the solutions is changed by applying the following update rule:

$$\tau_{ij} \leftarrow \rho.\tau_{ij} + \Delta\tau_{ij} \quad (2)$$

where $0 < \rho < 1$ models pheromone evaporation and $\Delta\tau_{ij}$ represents additional added pheromone. Normally, the quantity of the added pheromone depends on the quality of the solution.

2.3. Computational Mechanics Parameter Sweep Experiments (PSEs)

From the seminal paper on Computational Mechanics due to Bathe and Oden [34], many advances can indeed be addressed. Non linear solid mechanics in general, and Finite Strain Plasticity in particular, have been benefited from the works of Simo and Ortiz [40–42]. In the literature different problems can be found where it is important to study the sensitivity of results in terms of changes of variable data. For instance, García Garino et al. [20] have discussed the sensitivity of results of the necking problem of circular cylindrical bars in terms of applied imperfections.

A concrete example of a PSE is the one presented by Careglio et al. [13], which consists in analyzing the influence of size and type of geometric imperfections in the response of a simple tensile test on steel bars subject to large deformations. To conduct the study, the authors numerically simulate the test by varying some parameters of interest, namely using different sizes and types of geometric imperfections. By varying these parameters, several study cases were obtained, which was necessary to analyze and run on different machines in parallel. More recently, García Garino et al. [21] have discussed a large strain viscoplastic constitutive model. A plane strain plate with a central circular hole under imposed displacements stretching the plate has been studied. Different values were considered for viscosity and other constitutive model parameters in order to adjust the model response. As can be seen in Fig. 4 rather different deformation patterns have been found for different values of viscosity η.

Consequently, different results can be expected for the different values of constitutive parameters considered, which in practice can led to significantly different CPU times in order to complete the execution of the associated numerical simulations. Even in the case of static assignation of computing resources [19,30], a rather complex scheduling problem has to be solved. Particularly, the simulations in this work are based on a large strain elastoplastic/elastoviscoplastic constitutive model written in terms of internal variables theory and a hyperelastic free energy function [18,21], following the ideas of Simo, Ortiz and co-authors [40–42]. It is important to mention that only few works devoted to Finite Elements on Cloud Computing infrastructures can be found in the literature [3,19,30,36,55].

3. Proposed scheduler

Our scheduler deals with the problem described next. A number of users are connected to the Cloud at different times to execute their PSEs, and each user requests to the Cloud the creation of v VMs. A PSE is formally defined as a set of $N = 1, 2, \ldots, n$ independent jobs, where each job corresponds to a particular value for a variable of the model being studied by the PSE. The jobs are distributed and executed on the v VMs created by the corresponding user. Since the total number of VMs required by all users is usually greater than the number of Cloud physical resources (i.e., hosts), a strategy that achieves a good use of these physical resources is needed. This strategy is implemented at the Infrastructure-level by means of a support that allocates user VMs to hosts. Moreover, a strategy for assigning user jobs to allocated VMs is also necessary (currently we use FIFO).

To implement the Infrastructure-level strategy, AntZ, the algorithm proposed in [29] to solve the problem of load balancing in Grid environments has been adapted to be used in Clouds (see Algorithm 1). AntZ combines the idea of how ants cluster objects with their ability to leave pheromone trails on their paths so that it can be a guide for other ants passing their way.

Algorithm 1. ACO-based allocation algorithm for individual VMs

```
Procedure ACOallocationPolicy (vm,hostList)
Begin
    initializeLoadTable()
    ant = getAntPool(vm)
    if (ant==null) then
        suitableHosts = getSuitableHostsForVm(hostList,vm)
        ant = new Ant(vm,suitableHosts)
        antPool.add(vm,ant)
    end if
    repeat
        ant.AntAlgorithm()
    until ant.isFinish()
    allocatedHost = hostList.get(ant.getHost())
    if (!allocatedHost.allocateVM(ant.getVM()))
        repeat
            ACOallocationPolicy(ant.getVM(),hostList)
            numberOfRetries—
        until successful or numberOfRetries==0
End
```

In our adapted algorithm (see Algorithm 1), each ant works independently and represents a VM “looking” for the best host to which it can be allocated. When a VM is created, an ant is initialized. A master table containing information on the load of each host is initialized (`initializeLoadTable()`). Subsequently, if an ant associated to the VM that is executing the algorithm already exists, the ant is obtained from a pool of ants through the `getAntPool(vm)` method. If the VM does not exist in the ant pool, then a

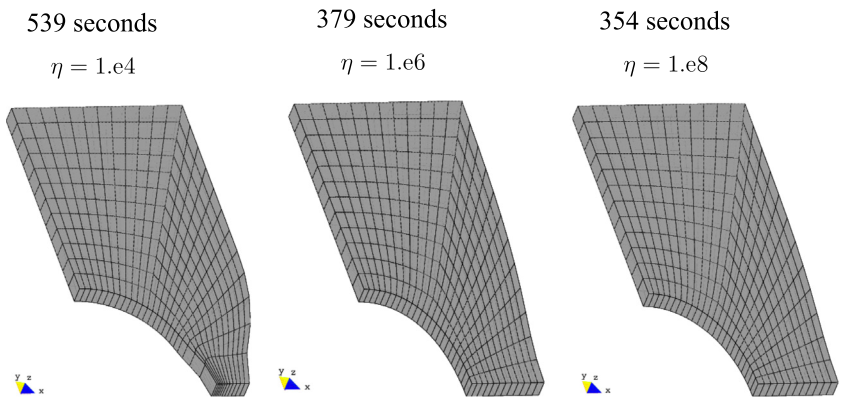

Fig. 4. Deformed shapes for 2 *m* stretching: sensitivity of results in terms of viscosity parameter value.

new ant is created. To do this, first, a list of all suitable hosts in which can be allocated the VM is obtained. A host is suitable if it has an amount of processing power, memory and bandwidth greater than or equal to that of required by the unallocated VM.

Algorithm 2. ACO-specific logic: Core logic

```
Procedure AntAlgorithm()
Begin
    step = 1
    networkMessages = 0
    initialize()
    While (step < maxSteps) do
      currentLoad = getHostLoadInformation()
      AntHistory.add(currentLoad)
      localLoadTable.update()
      if (currentLoad = 0.0)
        break
      else
      if (random() < mutationRate) then
        nextHost = randomlyChooseNextStep()
      else
        nextHost = chooseNextStep()
      end if
      mutationRate = mutationRate-decayRate
      networkMessages = networkMessages + 1
      step = step + 1
      moveTo(nextHost)
    end while
    deliverVMtoHost()
End
```

Each working ant and its associated VM are added to the ant pool (`antPool.add(vm,ant)`) and the ACO-specific mechanism starts to operate (see Algorithm 2). In each iteration of the sub-algorithm, the ant collects the load information of the host that is visiting and adds this information to its private load history. The ant then updates a load information table of visited hosts (`localLoadTable.update()`), which is maintained in each host. This table contains information of the own load of an ant, as well as load information of other hosts, which were added to the table when other ants visited the host. Here, load refers to the total CPU utilization within a host and is calculated taking into account the number of VMs that are executing at a given time in each physical host. To calculate the load, the original AntZ algorithm receives the number of jobs that are executing in the resource in which the load is being calculated, and it is calculated taking into account the amount available of million instructions per second (MIPS) in each CPU. MIPS is a metric that indicates how fast a computer processor runs. In our scheduler, the load is calculated on each host taking into account the CPU utilization made by all the VMs that are executing on each host. This metric is useful for an ant to choose the least loaded host to allocate its VM.

When an ant moves from one host to another it has two choices: moving to a random host using a constant probability or *mutation rate*, or using the load table information of the current host (`chooseNextStep()`). The mutation rate decreases with a *decay rate*factor as time passes, thus, the ant will be more dependent on load information than to random choice. This process is repeated until the finishing criterion is met. The completion criterion is equal to a predefined number of steps (*maxSteps*). Finally, the ant delivers its VM to the current host and finishes its task. Due to the fact that each step performed by an ant involves moving through the network, we have added a control to minimize the number of steps that an ant performs: every time an ant visits a host that has not yet allocated VMs, then the ant allocates its associated VM to it directly without performing further steps. The number of messages sent over the network by an ant to hosts to obtain information regarding their availability is accumulated every time an ant takes a step.

When the ant has not completed its work, i.e., the ant cannot allocate its associated VM to a host, then an exponential back-off strategy may be activated. The allocation of each failing VM in the queue is re-attempted every *s* seconds and retried *n* times.

Algorithm 3. ACO-specific logic: The ChooseNextStep procedure

```
Procedure ChooseNextStep()
Begin
    bestHost = currentHost
    bestLoad = currentLoad
    for each entry in hostList
      if (entry.load < bestLoad) then
        bestHost = entry.host
      else if (entry.load = bestLoad) then
        if (random.next < probability) then
          bestHost = entry.host
        end if
      end if
    end for
End
```

Every time an ant visits a host, it updates the host load information table with the information of other hosts, but at the same time the ant collects the information already provided by the table of that host, if any. The load information table acts as a pheromone trail that an ant leaves while it is moving, to guide other ants to choose better paths rather than wandering randomly in the Cloud. Entries of each local table represent the hosts that ants have visited on their way to deliver their VMs together with load information.

When an ant processes the information from a load table in a host via the Algorithm 3, the ant selects the lightest loaded host in the table, i.e., each entry of the load information table is evaluated and compared with the current load of the visited host. If the load of the visited host is smaller than any other host stored in the load information table, the ant chooses the host with the smallest load. On the other hand, if the load of the visited host is equal to any host in the load information table, the ant chooses any of these hosts randomly.

Algorithm 4. The SubmitJobsToVMs procedure

```
Procedure SubmitJobsToVMs(jobList)
Begin
    vmIndex = 0
    while (jobList.size() > 0)
        job = jobList.getNextJob()
        vm = getVMsList(vmIndex)
        vm.scheduleJobToVM(job)
        totalVMs = getVMsList().size()
        vmIndex = Mod(vmIndex + 1,totalVMs)
        jobList.remove(job)
    end while
End
```

Once the VMs have been allocated to physical resources, the scheduler proceeds to assign the jobs to these VMs. To do this, jobs are assigned to VMs according to the Algorithm 4. This represents the second scheduling level of the scheduler proposed as a whole. This sub-algorithm uses two lists, one containing the jobs that have been sent by the user, i.e., a PSE, and the other list contains all user VMs that are already allocated to a physical resource and hence are ready to execute jobs. The algorithm iterates the list of all jobs – `jobList` – and then, through `getNextJob()` method retrieves jobs by a FIFO policy. Each time a job is obtained from `jobList`, it is submitted to be executed in a VM in a round robin fashion. The VM where the job is executed is obtained through the method `getVMsList(vmIndex)`. Internally, the algorithm maintains a queue for each VM that contains its list of jobs to be executed. The procedure is repeated until all jobs have been submitted for execution, i.e., when the `jobList` is empty.

4. Evaluation

To assess the effectiveness of our proposal in a non-batch Cloud environment where multiple users dynamically connect to request VMs, we processed a real case study for solving a well-known benchmark problem discussed for instance in [21]. Methodologically, we first executed the problem in a real single machine by varying an individual problem parameter by using a finite element solver, called SOGDE [18], in order to gather real job processing times and input/output data sizes. By means of the generated job data, we performed the experimental setup by configuring the CloudSim simulation toolkit. Details on the experimental methodology are provided in Section 4.1. After that, we compared our proposal with some Cloud scheduling alternatives in terms of the metrics of interest. The results are explained in Section 4.4.

4.1. Experimental methodology

A classical benchmark problem cited in the literature, see [2] for instance, involves studying a plane strain plate with a central circular hole (see Fig. 5). The dimensions of the plate were 18×10 m, with $R = 5$ m. On the other hand, material constants considered were $E = 2.1 \times 10^5$ Mpa, $\nu = 0.3$, $\sigma_y = 240$ Mpa and $H = 0$. A linear Perzyna viscoplastic model with $m = 1$ and $n = \infty$ was considered. The 3D finite element mesh used had 1152 elements and H1/P0 elements were chosen. Imposed displacements (at $y = 18$ m) were applied until a final displacement of 2000 mm was reached in 400 equal time steps of 0.05 mm each. Lastly, $\Delta t = 1$ has been set for all the time steps. Unlike previous studies of our own [13], in which a geometry parameter – particularly imperfection – was chosen to generate the PSE jobs, in this case a material parameter was selected as the variation parameter. Then, 25 different viscosity values for the η parameter were considered, namely $x \cdot 10^y$ Mpa, with $x = 1$–5 and 7 and $y = 4$–7, plus $1 \cdot 10^8$ Mpa. Useful and introductory details on viscoplastic theory and numerical implementation can be found in [21]. The tests were solved using the SOGDE 3D finite element solver software [18].

After that, we employed a single real machine to run the parameter sweep experiment by varying the viscosity parameter η as indicated and measuring the execution time for the 25 different experiments, which resulted in 25 input files with different input configurations and 25 output files. The experiment were processed using an AMD Athlon(tm) 64 X2 Dual Core Processor 3600+ machine, 2 GB of RAM, equipped with the Ubuntu 12.04 operating system. The information regarding machine processing power was obtained from the native benchmarking support of Linux and is expressed in MIPS. The machine has 4008.64 MIPS. It is worth noting that only one core was used during the experiments, since SOGDE supports sequential program execution.

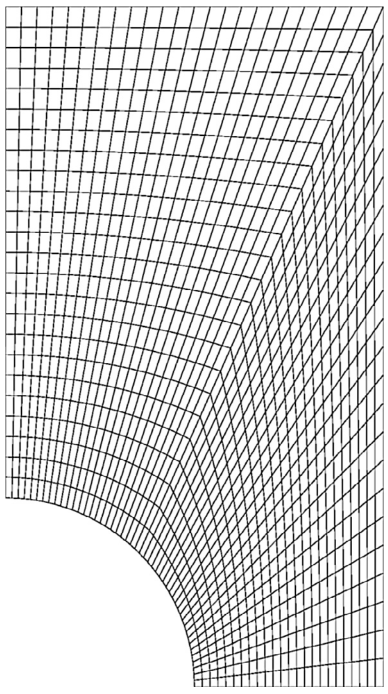

Fig. 5. Plane strain plate: Finite element mesh of 1152 elements.

Once the execution times were obtained from the real machine, we approximated for each experiment the number of executed instructions by the following formula $NI_i = mipsCPU * T_i$, where NI_i is the number of million instructions associated to job i, *mipsCPU* is the processing power of the CPU of our real machine measured in MIPS, and T_i is the time that took to run job$i$ on the real machine. For example, for a job taking 539 s to execute, the approximated number of instructions for the job was 2,160,657 MI (Million Instructions). Resulting jobs execution times and lengths are shown in Table 1.

After gathering real job data, the CloudSim simulation toolkit [12] was configured with a Cloud composed of a single machine ("host" in CloudSim terminology) with similar characteristics as the real machine where the above experiments were performed. We used more cores ("processing elements" or "PEs" in CloudSim), each one with the same processing power than the real machine, and more memory capacity. Table 2 shows the characteristics of the configured host and virtual machine. Once configured, we checked that the execution times obtained by the simulation coincided or were close to real times for each independent job performed on the real machine. The results were successful in the sense that one experiment (i.e., a variation in the value of η) took 539 s to be solved in the real machine, while in the simulated machine the elapsed time was 539.086 s. The range of differences between simulated times and real times for all jobs was [0.054–0.086]. Once the execution times have been validated for a single machine on CloudSim, a new simulation scenario was set. This new scenario consisted of a datacenter with 10 hosts, where each had the same hardware capabilities as the host in Table 2. Then, each user connecting to the Cloud requests v VMs to execute their PSE. Each VM has the characteristics specified in Table 2. This is a moderately-sized, homogeneous datacenter likely to be found in many real scenarios [30].

To evaluate the performance in the simulated Cloud we have modeled an online Cloud scenario in which new users connect to the Cloud every 600 s. We have set to 600 s the connection gap of users to have an approximate time to the longest job execution time which was 539 s (see Table 1, first job). The aim was to establish a time that does not completely saturate the system with the load, and further that the load is manageable. Furthermore, each user requires the creation of 10 VMs in which they run their PSE –a set of 10 * 25 jobs–. This is, the real base job set comprising 25 jobs that was obtained by varying the value of η was cloned to obtain more jobs. The number of users who connect to the Cloud varies as $u = 10, 20, \ldots, 120$, and since each user executes one PSE –100 jobs–, the total number of jobs to execute is $n = 100 * u$ at each time. Likewise, the total number of requested VMs is $m = 10 * u$ at each time.

Each job, called *Cloudlet* by CloudSim, had the characteristics shown in Table 3, where the Length parameter is the number of instructions to be executed by a Cloudlet, which varied between 1,362,938 and 2,160,657 MIPS (see Table 1). Each Cloudlet required only one PE since as explained above real jobs are not multi-threaded. Input size and Output size are the input file size and output file size, respectively, measured in bytes.

Table 1
Real jobs execution times and lengths.

Parameter η	Execution time (s)	Length (MI)
1.10^4	539	2,160,657
2.10^4	458	1,835,957
3.10^4	454	1,819,923
4.10^4	436	1,747,767
5.10^4	399	1,599,447
7.10^4	395	1,583,413
1.10^5	401	1,607,465
2.10^5	356	1,427,076
3.10^5	365	1,463,154
4.10^5	361	1,447,119
5.10^5	381	1,527,292
7.10^5	375	1,503,240
1.10^6	379	1,495,223
2.10^6	340	1,362,938
3.10^6	342	1,370,955
4.10^6	344	1,378,972
5.10^6	367	1,471,171
7.10^6	357	1,431,084
1.10^7	359	1,439,102
2.10^7	354	1,419,059
3.10^7	350	1,403,024
4.10^7	351	1,407,033
5.10^7	351	1,407,033
7.10^7	355	1,423,067
1.10^8	354	1,419,059

Table 2
Simulated Cloud machines characteristics. Host parameters (left) and VM parameters (right).

Host parameters	Value	VM parameters	Value
Processing power	4008 MIPS	Processing power	4008 MIPS
RAM	4 Gbytes	RAM	512 Mbytes
Storage	400 Gbytes	Machine image Size	100 Gbytes
Bandwidth	100 Mbps	Bandwidth	25 Mbps
PEs	4	PEs	1
		VMM (Virtual Machine Monitor)	Xen

4.2. Alternative schedulers considered

Here, we report the results when executing PSEs submitted by multiple users in the simulated Cloud using our two-level scheduler and alternative Cloud scheduling policies for assigning VMs to hosts. Due to their high CPU requirements, and the fact that each VM requires only one PE, we assumed a 1–1 job-VM execution model, i.e., jobs within a VM waiting queue are executed one at a time by competing for CPU time with other jobs from other VMs in the same hosts. In other words, a time-shared CPU scheduling policy was used, which ensures fairness. Although our scheduler is independent from the SI technique exploited at the Infrastructure-level, it will be referred as "ACO" for simplicity. Moreover, our proposed algorithm is compared against another three schedulers:

- Random allocation, a scheduling algorithm in which the VMs requested by the different users are assigned randomly to different physical resources. Although this algorithm does not provide an elaborated criterion to allocate the VMs to physical resources, it provides a good benchmark to evaluate how our scheduler performs compared to random assignment.

Table 3
Cloudlet configuration used in the experiments.

Cloudlet parameters	Value
Length (MIPS)	1,362,938–2,160,657
PEs	1
Input size (bytes)	291,738
Output size (bytes)	5,662,310

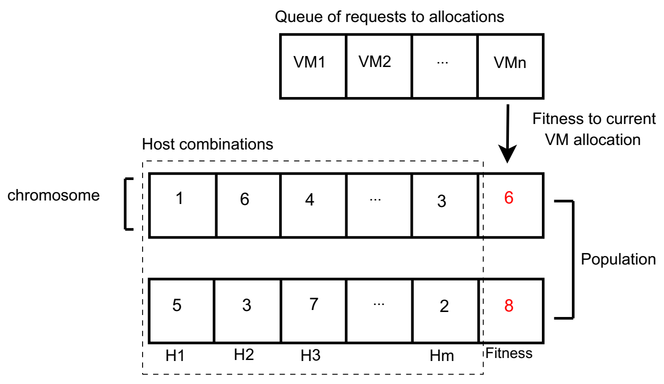

Fig. 6. Genetic encoding of VM scheduling to physical hosts.

- A Cloud scheduler based on Genetic Algorithm (GA) proposed in [1], in which the population structure is represented as the set of physical resources that compose a datacenter, as illustrated in Fig. 6. Each chromosome is an individual in the population that represents a part of the searching space. Each gene (field in a chromosome) is a host in the Cloud, and the last field in this structure is the *fitness* field, which is updated in each chromosome for each VM allocation request. The fitness field indicates the result of the fitness function and it is calculated as the inverse of the accumulated load of all hosts composing the chromosome. The load in each is host is calculated taking into account the number of VMs that are executing in it. A chromosome with higher fitness indicates that its associated set of hosts has the most free CPUs to perform the current allocation. Each chromosome keeps combinations of hosts and the fitness of the current allocation.

Algorithm 5. Genetic algorithm pseudo-code

```
Procedure GAalgorithm()
Begin
    generation = 1
    P = createPopulation(sizePopulation)
    evaluatePopulation(P)
    While (generation < maxGenerations) do
        P2 = select(P)
        reproduce(P2)
        evaluate(P2)
        P = renewPopulation(P,P2)
        generation = generation + 1
    end while
    i = 1
    While (i < sizePopulation) do
        localSort(P)
    end while
    j = 1
    While (j < sizePopulation) do
        globalSort(P)
    end while
End
```

The Algorithm 5 shows the pseudo-code of this scheduler. The initiation step of the population (`createPopulation(size)`) outputs the set of hosts available in the Cloud. This is, as illustrated in Fig. 6, the population is represented as a set of hosts. Each chromosome keeps combinations of hosts and its associated fitness. This fitness value is updated every time a VM is requested by an user to indicate the suitability of the hosts in each chromosome (`evaluatePopulation(P)`). In each generation, a new population *P2* originated from the initial population *P* is formed by selecting chromosomes using a Roulette method [27] (`select(P)`), given a probability of selection proportional to the chromosome fitness. This *P2* population is recombined using a uniform crossover (`reproduce(P2)`) with the aim of exploring more possible hosts with better fitness than the current allocation. The evaluation step (`evaluate(P2)`) is done over the *P2* population to update the fitness field of this new recombined population. Chromosomes with low fitness in *P* are replaced by the better individuals in *P2* (`renewPopulation(P,P2)`). Thus, the algorithm preserves the best individuals to increase the probability of a better allocation. At the end of generations, two sorting steps are done: one local (`localSort(P)`) to provide a sorted list of hosts in the chromosome with higher fitness, and a global sort (`globalSort(P)`), to provide a sorted list of individuals with better fitness. The allocation of VMs will begin in the first host of the first chromosome. If this host is not able to perform this operation, the next host in the chromosome with better fitness is selected.
In our experiments, the GA-specific parameters were set to the following values: *chromosome size* = *8*, *population size* = *10* and *number of iterations* = *10*. In [1] the authors have set the chromosome size equal to the number of available hosts, but in this paper we have reduced this number in order to reduce the network consumption, and because in a population of size 10 still almost all chromosomes are reached to consider all available hosts.

- An ideal scheduler, which achieves the best possible allocation of VMs to physical resources in terms of the studied metrics. To allocate all the VMs, the scheduler uses a back-off strategy until it is able to serve all users. The number of enough retries to serve all users and create all requested VMs was 20. This scheduler has been implemented in this way to obtain the ideal values to which all its competitors, including ACO, should be compared against.

In our ACO scheduler, we have set the ACO-specific parameters – i.e., mutation rate, decay rate and maximum steps – with values within the range of values studied in [29]: *mutation rate* = *0.6*, *decay rate* = *0.1* and *maximum steps* = *8*. In all cases, the considered algorithms use the same policy for handling jobs within VMs (i.e., FIFO with round robin), and the VMs allocated to a single host (i.e., time-shared [37]). Using time-shared means there is not limit on the number of VMs a host can handle in terms of CPU time. A host can, however, reject the creation of a new VM because of insufficient RAM or disk space.

4.3. Evaluation metrics

In our previous works [19,30], a batch scenario was considered and the goal was minimizing the flowtime and makespan of all jobs submitted by one user. Flowtime is the sum of job finish times minus job start times of a set of jobs. Makespan is the maximum execution time of a set of jobs. Here, an *online Cloud* [36] scenario is employed. An online Cloud is a Cloud which is available all the time and to which different users connect at different times to submit their experiments. The experiments have been performed with the aim of measuring the trade-off between the number of serviced users by the Cloud – among all users that are connected to the Cloud – and the total number of created VMs among all users. The basis for these metrics is that the more the number of serviced users, the higher the end-user throughput, and the greater the number of created VMs, the greater the parallelism and therefore the lower the flowtime [30]. The number of serviced users increases every time the scheduler successfully allocates any of the requested VMs. A user is considered "serviced" if the scheduler can create at least one VM from its requirement jobs.

Based on these two metrics, we derived a *weighted metric*, by which the results obtained from the different algorithms have been normalized and weighted with numerical weights. The normalized values for each metric and each user group U connected to the Cloud are computed as:

$$NormalValueU_{i=10,20,\ldots,120} = 1 - \left(\frac{Max(valueU_i) - valueU_i}{Max(valueU_i) - Min(valueU_i)}\right) \quad (3)$$

where *valueU* represents the obtained value for each one of the basic metrics – serviced users and created VMs – and for each user group connected to the Cloud, *Max(valueU)*and *Min(valueU)* are the maximum and minimum values, respectively, for each basic metric among all the algorithms – ACO, Random, GA, Ideal – and for each user group connected to the Cloud. Moreover, the weighted metric is computed as:

$$WeightedMetricU_{i=10,\ldots,120} = (weightSU * NormalSU_i + weightVMs * NormalVMsU_i) \quad (4)$$

where *weightSU* is the weight applied to the number of serviced users by the Cloud (*NormalSU*) and *weightVMs* weighs the total number of created VMs (*NormalVMs*). Based on these, three weights combinations have been used. Each pair of weight combinations (weightSU, weightVMs) represent a different scenario. We evaluate pure HTC/HPC scenarios by assigning the weight combinations (1, 0)/(0, 1) (Sections 4.4.1 and 4.4.2, respectively), and a mixed scenario by assigning the weights (0.50, 0.50) with the aim of achieving a balance between the number of serviced users and the number of created VMs (Section 4.4.3).

Finally, we study in more detail how the number of serviced users and the number of created VMs behaves when using the exponential back-off strategy to retry the allocation of failing VMs. The back-off strategy is activated every time a VM fails in their first attempt to creation, and retries allocating the VM based on an exponential function. The number of retries is equal to 3. In our previous work [36] we determined that 3 retries is a reasonable number when reallocating VMs. More retries does not lead to more successful VM allocations, and the schedulers become more inefficient.

4.4. Experimental results

Irrespective of the metric, in this work we show average results, which arise from averaging 20 times the execution of each algorithm. Previously, to select the appropriate number of executions for reporting the results, experiments were performed with different numbers of executions: 15, 20, 25 and 30. Although the more the number of executions, the more accurate the obtained results, deviations in the order of 1/10,000 were obtained in the results when the number of executions increased between 15 and 30. For example, with 15 executions, the standard deviation with respect to the average results of 30 executions (the most accurate results) for the metric *serviced users* varied between 0 and 0.13, and for 20 executions varied between 0 and 0.06. On the other hand, for the metric *created VMs* the standard deviation varied between 0.02 and 1.56 for 15 executions, and between 0.26 and 1.40 for 20 executions.

4.4.1. Pure HTC scenario – weight combination (1, 0)

In HTC [8] environments, the main challenge is how to maximize the amount of resources accessible to its users. These computing paradigm is more suited for running multiple independent jobs on multiple computing resources at the same time. The HTC field is more interested in how many jobs can be completed over a long period of time (throughput) instead of how fast an individual job can complete. In this sense, we consider a pure HTC scenario as the one that prioritizes to serve as many users as possible, i.e., the number of users that can be actually serviced from those connected to the Cloud. In this context, "serviced" means those users for which at least one VM can be allocated. Moreover, the reason that the schedulers cannot serve some users that connect to the Cloud is because the attempt to create all VMs requested by certain users fail. This means that, if the scheduler fails to create any VMs requested by an user, then this user is considered *not served*.

Table 4 shows the weighted metric for the different algorithms. We have assigned the weights combinations (weightSU, weightVMs) = (1, 0). First column illustrate the number of user trying to connect to the Cloud, and moreover, each row represents a different scenario. Specifically, the first row represents the situation where up to 10 users are connected but not all can be serviced. In the second row up to 20 users are connected, whereas in the third row up to 30 users connect, and so on. Then, the second column indicates the range of times, in minutes, between each user connects to the Cloud and actually issues the creation of their VMs. For example, in the first row, the range of connection times [0–90] represents the scenario in which the first user connects to the Cloud at time 0, the second user connects 10 min after, and so on until the last user is connected (90 min after the first one). Finally, the last four columns shows the results of the weighted metric for each one of the algorithms.

As shown in Table 4, among all approaches, Random is the algorithm that serves more users with respect ACO and GA. However, as we will show in the next subsection, Random is the algorithm that creates less VMs. It is important to note that, while Random serves many users, it is in general not fair with the response times for users, producing a very large flowtime [30]. The reason behind this is that the Random algorithm assigns the VMs to physical resources randomly, and many of the creations of the VMs requested by users might fail. There are situations where for a single user Random is able to create only one VM where all jobs of the user are executed. This situation means that the user must wait too long to complete their jobs and thus loses the benefit of using a Cloud. Note that the weighted metric is always zero for GA because it is the less efficient algorithm in the number of users that achieves to serve.

Then, we evaluated the different algorithms when the back-off strategy is used. Table 5 shows again the weighted metric for this experiment. As shown, Random again is the scheduler that serve more users. Note that the values of the weighted metric for the number of serviced users are lower when using the back-off strategy. This happens because the back-off strategy starts to try reallocating VMs from the first VM that failed in their first attempt to creation. Moreover, generally, the VMs that failed correspond to the first users connected to the Cloud. The back-off strategy serves fewer users but with a greater number of VMs for each one of them (see Section 4.4.2). The reason is because when the schedulers are able to create a greater number of VMs for the first users connected to the Cloud, the availability of physical resources decreases earlier, and as a consequence, fewer users can be serviced.

Table 6 shows the gains obtained in the number of serviced users using the back-off strategy for each algorithm with respect

Table 4
Weighted metric with weight combinations (1,0).

Users connected to the Cloud	Range of connection times (min)	Without retries of VM creation			
		ACO	GA	Random	Ideal
10	[0–90]	0.62	0	0.84	1
20	[0–190]	0.55	0	0.58	1
30	[0–290]	0.37	0	0.56	1
40	[0–390]	0.33	0	0.45	1
50	[0–490]	0.32	0	0.39	1
60	[0–590]	0.31	0	0.38	1
70	[0–690]	0.29	0	0.33	1
80	[0–790]	0.25	0	0.31	1
90	[0–890]	0.22	0	0.28	1
100	[0–990]	0.20	0	0.26	1
110	[0–1090]	0.20	0	0.25	1
120	[0–1190]	0.19	0	0.24	1

Table 5
Weighted metric with weight combinations (1,0) and back-off strategy.

Users connected to the Cloud	Range of connection times (min)	With 3 retries of VM creation			
		ACO	GA	Random	Ideal
10	[0 – 90]	0.43	0	0.43	1
20	[0 – 190]	0.43	0	0.36	1
30	[0 – 290]	0.25	0	0.26	1
40	[0 – 390]	0.25	0	0.23	1
50	[0 – 490]	0.20	0	0.25	1
60	[0 – 590]	0.18	0	0.19	1
70	[0 – 690]	0.19	0	0.20	1
80	[0 – 790]	0.15	0	0.17	1
90	[0 – 890]	0.15	0	0.16	1
100	[0 – 990]	0.13	0	0.16	1
110	[0 – 1090]	0.10	0	0.16	1
120	[0 – 1190]	0.12	0	0.16	1

Table 6
Serviced users: % Gain when using the back-off strategy.

Users connected to the Cloud	% Gain ACO	% Gain GA	% Gain random	% Gain ideal
10	−8.09	−3.57	−15.52	N/A
20	−8.00	−5.65	−12.70	N/A
30	−8.74	−4.13	−18.54	N/A
40	−6.43	−3.32	−14.18	N/A
50	−8.73	−2.39	−9.58	N/A
60	−8.74	−0.98	−11.93	N/A
70	−7.51	−1.74	−9.09	N/A
80	−6.64	−1.07	−9.36	N/A
90	−5.17	−1.15	−8.40	N/A
100	−6.11	−1.47	−7.55	N/A
110	−7.26	−1.26	−7.22	N/A
120	−5.67	−1.01	−6.43	N/A

$$\%GainUsers_{u=10,20,\ldots,120} = 100 - \frac{(numberServicedUsersWithoutRetries_u(ACO, GA, Random) \, * \, 100)}{(numberServicedUsersWithRetries_u(ACO, GA, Random))} \quad (5)$$

to not using it. The gains are calculated considering the number of serviced users using the back-off strategy and for each group of users $u = 10, 20, \ldots, 120$:

As shown in Table 6 all gains are negative numbers, which means that instead of gains, losses are obtained in the number of serviced users when the back-off strategy is used. Some observations are that, when 10 users join to the Cloud the loss of ACO to use retries of VM creation is 8.09%, the loss of GA is 3.57% and the loss of Random is 15.52%. For the ideal scheduler, gains are not shown as the algorithm already reached the best possible values without the strategy. The highest losses in terms of number of serviced users for all algorithms were obtained when the numbers of users that try to connect to the Cloud were from 10 to 60 users. Random and GA present the higher and the lower losses, respectively, in the number of serviced users. In the next subsection, it can be seen that though the strategy of back-off serves fewer users, it is able to create more VMs.

4.4.2. Pure HPC scenario – weight combination (0, 1)

HPC [8] environments are those who are evaluated in terms of executed floating-point operations per seconds, and hence their most important goal is to achieve the greater performance. Therefore, in this work we consider that the greater number of VMs, the greater number of operations per second that can be executed for PSEs, achieving better response times. Then, in this subsection we evaluate the performance of a pure HPC scenario which only gives importance to the number of created VMs. This is equivalent to assign the weights combinations of the weighted metric such as (weightSU, weightVMs) = (0, 1).

As shown in Table 7, among all approaches, excluding the ideal scheduler, GA is the algorithm that creates more VMs. This is because the population size is equal to 10, and each chromosome contains 7 different hosts, so after 10 iterations GA always finds the hosts with better fitness, and can thus allocate more VMs to the first users who connect to the Cloud. However, as we shown in previous subsection, GA is the algorithm that serves the smaller number of users. Note that the weighted metric is always zero for Random because it is the less efficient algorithm in the number of VMs that achieves to create.

The creation of some VMs fails at the moment an user issues the creation, due to all physical resources are already fully busy with VMs belonging to other users, i.e., because the scheduler not found an available resource where allocate the VM. Depending on the algorithm and according to the results, some schedulers are able to find to some extent a host with free resources to which at least one VM per user is allocated. It is for this reason that in this work we have also incorporated the back-off mechanism that attempts to improve the number of VMs that are created for each user.

Next, we evaluate the impact of the back-off strategy for each algorithm in the number of VMs that each one is able to allocate. Table 8 shows the same weighted metric values for each algorithm. Again, GA is the scheduler that achieves to create more VMs, but GA is not the algorithm that achieves greater improvement levels when using the back-off strategy. Table 9 shows the gains obtained in the number of created VMs using the back-off strategy for each algorithm with respect not using it. The gain is calculated considering the number of created VMs for each algorithm using back-off strategy and for each group of users $u = 10, 20, \ldots, 120$:

$$\%GainVMs_{u=10,20,\ldots,120} = 100 - \frac{(numberCreatedVMsWithoutRetries_u(ACO, GA, Random) * 100)}{(numberCreatedVMsWithRetries_u(ACO, GA, Random))} \qquad (6)$$

As shown in Table 9, when the first 10 users join the Cloud the gain of ACO due to using the back-off strategy is 3.50%, the gain of GA is 0.93% and the gain of Random is 12.15%. The highest gains in terms of the number of created VMs for all algorithms were achieved when the number of users connected to the Cloud were in the range of 60–120. Random presents higher gains since it was the algorithm that created a lower number of VMs than ACO and GA without using retries. It is for this reason that by using the strategy the algorithm has more chances of finding a free host to assign a VM. However, the number of created VMs by Random is quite lower compared to ACO and GA. The second position of gains is presented by ACO and GA remains in the last place, but it is important to note that GA is the algorithm that achieves to create more VMs, and therefore, the best scheduler to use in a pure HPC Cloud environment, at least under the experimental conditions described so far.

4.4.3. Mixed HTC/HPC scenario – the weight combination (0.50, 0.50)

Since throughput is often the primary limiting factor in many scientific and engineering efforts, and moreover, many scientists and engineers are interested in obtaining their results as soon as possible, it is important to achieve the best possible balance between the two previous scenarios, i.e., HTC and HPC. In this subsection we evaluate the performance of a mixed computing scenario with the weights combinations (weightSU, weightVMs) = (0.50, 0.50). The higher the value of the weighted metric, the better the balance provided by an algorithm with respect to its competitors. Some observations are that when the VMs are created without retries as in Table 10, in all cases the weighted metric is more favorable to ACO, resulting in better values with respect to Random and GA.

As shown in previous subsections, ACO achieves to serve a greater number of users than GA, and create a greater number of VMs with respect to Random. But, as can be seen in this scenario, our ACO scheduler offers the best balance with respect to the number of serviced users and the total number of created VMs than GA and Random. Regardless of the weighted metric, a greater throughput in terms of serviced users and a greater number of created VMs involves greater parallelism, and therefore, a greater rate of jobs per unit time can be executed. Again, the Ideal scheduler has only been treated in these experiments as a fictitious algorithm that

Table 7
Weighted metric with weight combinations (0, 1).

Users connected to the Cloud	Range of connection times (min)	Without retrying VM creation			
		ACO	GA	Random	Ideal
10	[0–90]	0.28	0.32	0	1
20	[0–190]	0.32	0.33	0	1
30	[0–290]	0.28	0.34	0	1
40	[0–390]	0.28	0.35	0	1
50	[0–490]	0.26	0.35	0	1
60	[0–590]	0.27	0.39	0	1
70	[0–690]	0.28	0.36	0	1
80	[0–790]	0.26	0.38	0	1
90	[0–890]	0.30	0.38	0	1
100	[0–990]	0.27	0.39	0	1
110	[0–1090]	0.26	0.39	0	1
120	[0–1190]	0.26	0.38	0	1

Table 8
Weighted metric with weight combinations (0,1) and back-off strategy.

Users connected to the Cloud	Range of connection times (min)	With 3 retries of VM creation			
		ACO	GA	Random	Ideal
10	[0–90]	0.20	0.20	0	1
20	[0–190]	0.29	0.30	0	1
30	[0–290]	0.26	0.29	0	1
40	[0–390]	0.30	0.32	0	1
50	[0–490]	0.27	0.32	0	1
60	[0–590]	0.29	0.36	0	1
70	[0–690]	0.30	0.32	0	1
80	[0–790]	0.31	0.36	0	1
90	[0–890]	0.34	0.36	0	1
100	[0–990]	0.31	0.36	0	1
110	[0–1090]	0.31	0.37	0	1
120	[0–1190]	0.34	0.36	0	1

Table 9
Number of VMs: % Gain when using the back-off strategy.

Users connected to the Cloud	% Gain ACO	% Gain GA	% Gain random	% Gain ideal
10	3.50	0.93	12.15	N/A
20	8.31	9.17	21.27	N/A
30	10.67	6.08	22.93	N/A
40	15.18	8.45	26.63	N/A
50	15.85	8.38	28.42	N/A
60	18.22	8.19	31.14	N/A
70	18.57	8.56	33.33	N/A
80	19.87	8.97	30.42	N/A
90	19.94	9.56	34.59	N/A
100	21.16	8.75	35.90	N/A
110	22.35	8.48	35.80	N/A
120	24.35	8.09	33.50	N/A

Table 10
Weighted metric with weight combinations (0.50,0.50).

Users connected to the Cloud	Range of connection times (min)	Without retries of VM creation			
		ACO	GA	Random	Ideal
10	[0 – 90]	0.45	0.16	0.42	1
20	[0 – 190]	0.44	0.16	0.29	1
30	[0 – 290]	0.32	0.17	0.28	1
40	[0 – 390]	0.30	0.18	0.22	1
50	[0 – 490]	0.29	0.18	0.19	1
60	[0 – 590]	0.29	0.19	0.19	1
70	[0 – 690]	0.28	0.18	0.17	1
80	[0 – 790]	0.25	0.19	0.15	1
90	[0 – 890]	0.26	0.19	0.14	1
100	[0 – 990]	0.24	0.19	0.13	1
110	[0 – 1090]	0.23	0.20	0.13	1
120	[0 – 1190]	0.23	0.19	0.12	1

gets ideal results, but is taken as a reference to determine how far each competitor is from the former.

Next, and as in previous subsections, we aggregately evaluate the number of serviced users and the number of created VMs reached by the algorithms when using the back-off strategy. Table 11 shows that ACO again achieves the best balance with respect to its competitors in all cases. Although none of the schedulers were able to improve the number of serviced users (see Table 6), all of them were able to improve the number of created VMs (see Table 9). Due to the fact that ACO is in the second place in both gain tables, the weighted metric makes ACO the algorithm that achieves the best balance of the proposed metrics, and also turns it in the best approach to use for mixed high-computing Cloud scenarios, at least under the discussed experimental conditions.

4.4.4. Evaluation of the number of network messages sent and conclusions

In this subsection we summarize the results obtained in the previous scenarios, and evaluate the number of network messages sent by each one of the studied schedulers. To achieve allocate the VMs into hosts, each scheduler must make a different number of "queries" to hosts to determine their availability upon each VM allocation attempt. These queries are performed through messages sent to hosts over the network to obtain information regarding their availability. This process has been modeled in CloudSim by counting the number of messages sent to hosts every time an user request the allocation of a VM.

Fig. 7 illustrates the number of network messages sent to hosts by each algorithm to allocate the VMs. The Ideal scheduler needs to send messages to hosts every time a VM is allocated to know the hosts states and to decide where to allocate the VM. Moreover,

Table 11
Weighted metric with weight combinations (0.50, 0.50) and back-off strategy.

Users connected to the Cloud	Range of connection times (min)	With 3 retries of VM creation			
		ACO	GA	Random	Ideal
10	[0 – 90]	0.31	0.10	0.21	1
20	[0 – 190]	0.36	0.15	0.18	1
30	[0 – 290]	0.26	0.15	0.13	1
40	[0 – 390]	0.27	0.16	0.11	1
50	[0 – 490]	0.24	0.16	0.13	1
60	[0 – 590]	0.24	0.18	0.09	1
70	[0 – 690]	0.24	0.16	0.10	1
80	[0 – 790]	0.23	0.18	0.09	1
90	[0 – 890]	0.25	0.18	0.08	1
100	[0 – 990]	0.22	0.18	0.08	1
110	[0 – 1090]	0.21	0.18	0.08	1
120	[0 – 1190]	0.23	0.18	0.08	1

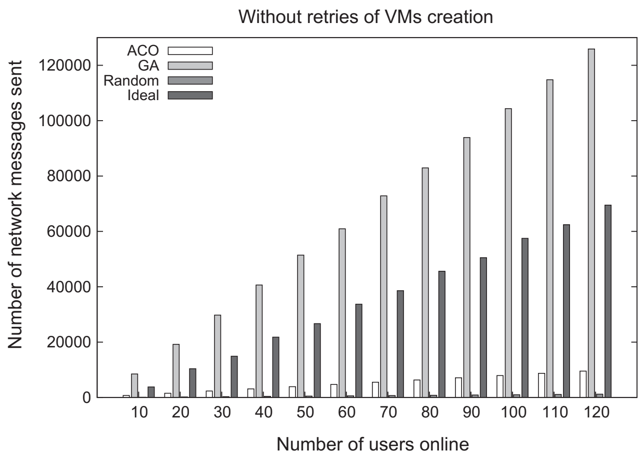

Fig. 7. Results as the number of users increases: number of network messages.

as mentioned earlier, Ideal always performs a number of creation retries until all users are served, which makes the number of messages sent even higher. It is important to note, however, that the Ideal algorithm implementation was executed using the back-off strategy with a number of retries equal to 20 to obtain the ideal values to reach. The number of network messages sent to hosts rose from 3800 to 69,500 when the number of users connected to the Cloud went from 10 to 120.

On the other hand, since the GA algorithm contains a population size of 10 and chromosome sizes of 8 (7 genes for hosts plus one gene for the fitness value), to calculate the fitness function, the algorithm sends one message for each host of the chromosome to know its availability and obtain the chromosome containing the best fitness value. This is, the VM is allocated to a host belonging to the chromosome with the best fitness value. The number of messages to send is equal to the number of host within each chromosome multiplied by the population size. As shown in Fig. 7, GA is the algorithm that makes greater use of network resources in respect to the other algorithms. The number of network messages sent to hosts varied from 8497.50 to 125897.50 when the number of connected users was increased from 10 to 120.

The last competitor in this work is Random, which sends only one network message to a random host for each attempt of VM creation and is the algorithm that makes the lowest network resource usage. The number of network messages rose from 100 to 1200 when the number of connected users to the Cloud went from 10 to 120.

Our ACO algorithm, however, makes less use of the network resources than GA and the Ideal scheduler. Due to the fact that we configure the maximum number of steps that an ant carries out to allocate a single VM to *8*, ACO sends a maximum of *8* messages per VM allocation. Moreover, when ACO finds an unloaded host, it allocates the current VM and does not perform any further step. This reduces the total number of network messages to send. The number of network messages sent by ACO to hosts rose from 730 to 9524.70 when the number of users connected to the Cloud went from 10 to 120. One point in favor is that, unlike Ideal and GA, ACO sent messages in the order of 100–1000 as Random did. Furthermore, the time taken to allocate 10 user VMs (i.e., sending at most 80 network messages) represents a very small fraction of the time each user PSE took, which was around 158 min of effective computing time. This fraction would be even smaller in Clouds with high-speed network connections.

Finally, when we configured the schedulers to operate with the back-off strategy the number of network messages increased. This happens because for each VM that failed in their first attempt to creation, the schedulers must be reactivated to allocate these VMs for a maximum of 3 times. Fig. 8 illustrates the number of network messages sent by each algorithm using the back-off mechanism. The number of network messages sent by GA to hosts varied from 22,782.80 to 376,585.80 when the number of connected users was increased from 10 to 120. In the case of Random the number of network messages also increased from [211.00–3476.60] when the number of users connected to the Cloud varied from 10 to 120. Finally, our ACO algorithm makes again less use of

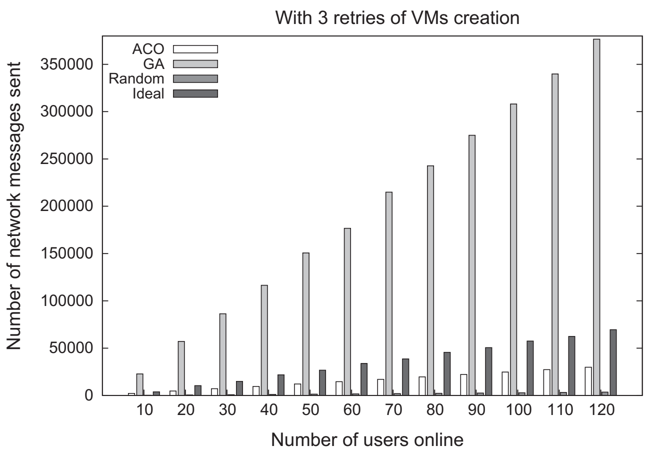

Fig. 8. Results as the number of users online increases: number of network messages.

the network resources than GA and the Ideal scheduler when using back-off strategy. The network messages rose from 2150.1 to 29,848.6 when the number of user connected to the Cloud went from 10 to 120.

To conclude, it is important to note that GA makes a greater use of network resources than ACO, being ACO the scheduler that achieves the best balance for the mixed environment proposed. Furthermore, an important consideration is that although Random sends few network messages, and the use of the network can be very important in some distributed environments, in most Clouds network interconnections are fast. Moreover, as we shown previously, Random is a very inefficient algorithm in terms of performance because it creates few VMs, and moreover, as we described in our previous work [30], Random gets the worse performance in terms of makespan and flowtime in batch scenarios. These results are encouraging because they indicate that ACO is close to obtaining the best possible solution balancing all the employed evaluation metrics and making a reasonable use of the network resources.

5. Related work

Studying SI techniques, specially ACO [39], has been the focus of a lot of research in the last ten years. A recent work [46] describes how ACO has been exploited to solve classical industrial scheduling problems. In this work the authors conclude and suggest, based on the basis of the literature reviewed, that ACO is a very viable approach to solve scheduling problems in general. Moreover, the authors were able to derive certain guidelines for the implementation of ACO algorithms. Furthermore, as evidenced by other surveys [54,47], these techniques have been applied to distributed job scheduling.

However, with regard to scheduling in Cloud environments, very few works can be found to date [35]. Moreover, to the best of our knowledge, no effort aimed to job scheduling based on SI for online Clouds where a large number of users are connected to submit their experiments has been proposed. By online we mean non-batch scenarios, i.e., where the jobs to be executed in the Cloud is not available beforehand. In these related works, it is important to note that, the most SI techniques are used to solve the job scheduling problem, i.e., determining how the jobs are assigned to VMs, and few efforts have aimed to solve VM scheduling problems, or how to allocate VMs to physical resources. Among them we can mention a recent survey from Huang et al. [24], which summarizes different methods to improve job execution performance, including dynamic resource allocation strategies based on the law of failure, dynamic resource assignment on the basis of credibility, Ant Colony Optimization algorithms for resource allocation, optimized genetic algorithm with dual fitness, among others. On the other hand, the objectives to optimize considered by the authors are suitable when the execution of a set of jobs belong to the same user, but when a large number of users make requests to the Cloud, fair mechanisms and normalized evaluation metrics such as those discussed in this work are not considered.

Moreover, the works in [6,56] propose ACO-based Cloud schedulers minimizing makespan and maximizing load balancing, respectively. An interesting aspect of [6] is that it was evaluated using real Cloud platforms (Google App Engine and Microsoft Live Mesh), whereas the other work was evaluated through simulations. During the experiments, [6] used only 25 jobs and a Cloud comprising 5 machines, while in [56] despite to be simulated the authors have not provided all the information needed to reproduce their experiments. Interestingly, like our work, these two efforts support dynamic resource allocation, i.e., the scheduler does not need to know the details of the jobs to allocate and the available resources.

An approach based on Particle Swarm Optimization (PSO), another popular SI technique, is proposed in [38]. PSO is inspired by the behavior of bird flocks, bee swarms and fish schools. Contrary to [6,56] and our scheduler, the approach is based on static resource allocation, which forces users to feed the scheduler with the estimated running times of jobs on the set of Cloud resources to be used. [38] is on the other hand targeted at paid Clouds, where users pay for the physical resources they use. As such, the work only minimizes monetary cost, and does not consider other metrics throughput or response time metrics.

Following to the above discussed approaches, another type of SI technique found in the literature is honey bee or bee colonies [15,45]. The work proposed in [15] aims to achieve load balancing across virtual machines of a Cloud for maximizing throughput. Also, this algorithm balances the priorities of jobs on the machines in such a way that the amount of waiting time of the jobs in the queue is minimal. On the other hand, the work in [45] proposes a mechanism to efficiently schedule data-oriented jobs onto Grid nodes and replicate data files on storage nodes with the objectives of minimizing both the makespan and the total datafile transfer time.

Finally, the works in [26,22] address the problem of job scheduling in Clouds while reducing energy consumption, which is a crucial problem [28] mainly because the environmental impact in terms of carbon dioxide (CO2) emissions caused by high energy consumption. [26] focuses however only on achieving competitive makespan. On the other hand, in [22] a new scheduling policy that models and manages a virtualized datacenter is proposed. It focuses on the allocation of VMs in datacenter nodes according to multiple facets to optimize the provider's profit. In particular, it considers energy efficiency, virtualization overheads, and SLA violation penalties, and supports the outsourcing to external providers.

It is worth noting that all the mentioned works ignore multiple users, rendering difficult their applicability to execute scientific experiments in online, shared Cloud environments.

6. Conclusions

Supporting experiments in engineering and scientific groups usually involves running a large amount of independent jobs, which requires a lot of computing power. These jobs must be efficiently processed in the different computing resources of a distributed environment such as the ones provided by Cloud. Consequently, job scheduling in this context indeed plays a fundamental role.

In recent years, SI has been received increasing attention in the research community. SI refers to the collective behavior that emerges from a swarm of social insects, which helps in solving complex combinatorial optimization problems. Particularly, ACO is an heuristic algorithm inspired by the behavior of real ants for solving such problems, which can be reduced to find good paths through graphs. Moreover, Cloud job scheduling is an NP-complete optimization problem, and many schedulers based on SI have been proposed. Basically, researchers have introduced changes to the traditional bio-inspired techniques to achieve different Cloud scheduling goals [35].

However, existing efforts do not address in general *online* environments where multiple users connect to scientific Clouds to execute their scientific experiments. On the other hand, to the best of our knowledge, no effort aimed at balancing the number of serviced users in a Cloud and the total number of created VMs by the scheduler exists. Indeed, the greater the number of serviced users, the better the throughput, and the more the created VMs, the higher the achieved parallelism. More parallelism means executing a greater number of jobs, and hence a more agile human processing of PSE job results. More serviced users means a more fair assignment of Cloud computing resources.

In this work, we have described a two-level Cloud scheduler based on SI, particularly Ant Colony Optimization, that operates under the IaaS model and pays special attention to the balance both throughput and response time – a mixed HTC–HPC scenario – in an online Cloud, i.e., a scenario in which several users are connected to the Cloud at different times. Moreover, both the number of serviced users and the total number of created VMs are important.

By means of simulated experiments performed with the CloudSim simulation toolkit and real PSE job data, we have evaluated three different scenarios through the use of a weighted metric by assigning different weights combinations. We evaluated two pure HTC and HPC scenarios by assigning the weights combinations (1, 0)/(0, 1) respectively, and a mixed scenario by assigning the weights (0.50, 0.50) with the aim of reaching a balance between the number of serviced users and the number of created VMs.

Depending on the scenario, the different algorithms behave better or worse. For example, when a pure HTC scenario is considered, Random is the algorithm that achieves the best performance in terms of throughput (serviced users), but as we have shown in our previous work [30], Random accomplishes in general very poor performance in terms of flowtime and makespan. On the other hand, when a pure HPC scenario is considered, GA proves to be the algorithm that achieves better performance in terms of response time. This happens because GA is the algorithm that creates a larger number of VMs. However, we also shown that GA serves the least number of users. Finally, when we evaluated the performance for a mixed HTC–HPC scenario with the aim to balance both the number of serviced users and the number of created VMs, our ACO algorithm is the one that best balances these two metrics with respect to Random and GA. Moreover, another observation from the experimental results is that by the use of a back-off strategy that retries the creation of VMs that have failed in their first attempt to creation, improvements in the number of created VMs were obtained.

In this paper, we have also evaluated the number of network messages sent to the host by each one of the studied schedulers to allocate the VMs. Results have shown that GA makes the highest use of network resources. Random sends less network messages than ACO, but this latter is the scheduler that achieves the best balance for the mixed environment proposed in this paper. Moreover, although Random is the algorithm which sends the least amount of network messages, it is a very inefficient algorithm in terms of performance because it creates few VMs and gets the worse performance in terms of makespan and flowtime in batch scenarios [30]. Nevertheless, we will in the future focus on heuristics for ACO to further reduce network consumption.

We are currently implementing another scheduler based on SI, specifically an adaptation of the Particle Swarm Optimization Grird scheduler proposed in [29], to explore the ideas exposed in this paper. Second, we plan to materialize the resulting schedulers on top of a real Cloud platform, such as OpenNebula (http://opennebula.org/), which is designed for extensibility. Third, we will consider other Cloud scenarios, e.g., federated Clouds with heterogeneous physical resources belong to different Cloud providers devoted to create an uniform Cloud resource interface to users. Moreover, an interesting research line in federated domains is the study of interconnection capacities – network links – among the domains. Indeed, in [1] a GA-based solution for the problem has been proposed, and therefore we aim at studying the usefulness of ACO and PSO in this context. Finally, we will evaluate and measure how the variation of the parameters of each algorithm (e.g., maxSteps, mutation rate and decay rate in ACO, chromosome size, population size and number of iterations in GA) influence the performance and network consumption. For instance, the more the maxSteps in our ACO scheduler, the more the “migrations” of ants among Cloud hosts, which increases network consumption.

Since our work is focused on the IaaS model where custom VMs are launched to be executed in the hosts available in a datacenter, energy consumption is another important issue. When simpler scheduling policies are used, e.g., Random, the balance between throughput and response time suffers, but CPU usage, access to memory and transfer through the network are less compared to that of more complex policies such as ACO or GA. For example, to maintain the load tables information for ants in ACO, or to maintain the chromosomes for populations in GA, we need those resources. Therefore, to execute many jobs or create a large number of VMs, the accumulated resource usage overhead may be significant, resulting in higher demands for energy. Then, we plan to quantify the trade-off between algorithm performance (as measured by the weighted metric) and energy consumption.

Lastly, an aspect to further explore is solution quality. SI algorithms in general, and ACO in particular, use indirect communication mechanisms to exchange information between entities. These mechanisms – e.g., pheromone update – usually lead to an undesirable “stagnation” effect [31], whereby entities explore the same solution paths from early stages. This, in turn, produces locally optimal solutions and hence overall performance is suboptimal. To deal with this problem, some direct communication mechanisms between entities have been studied [31,9]. For example, in [31] a direct communication for ACO algorithms in which near ants can exchange information is proposed. We will explore these mechanisms in the context of our ACO scheduler. We then expect to increase solution quality and therefore performance in terms of serviced users and created VMs.

Acknowledgments

We acknowledge the financial support provided by ANPCyT through Grants PAE-PICT 2007-02311, PAE-PICT 2007-02312 and National University of Cuyo Project 06/B253. The first author acknowledges her Ph.D. fellowships granted by the PRH-UNCuyo Project and the National Scientific and Technological Research Council (CONICET).

References

[1] Agostinho L, Feliciano G, Olivi L, Cardozo E, Guimaraes E. A bio-inspired approach to provisioning of virtual resources in federated clouds. In: Ninth international conference on dependable, autonomic and secure computing (dasc). DASC 11. Washington (DC, USA): IEEE Computer Society; 2011. p. 598–604.
[2] Alfano G, Angelis FD, Rosati L. General solution procedures in elasto-viscoplasticity. Comput Methods Appl Mech Eng 2001;190(39):5123–47.
[3] Ari I, Muhtaroglu N. Design and implementation of a cloud computing service for finite element analysis. Adv Eng Softw, in press.
[4] Armbrust M, Fox A, Griffithn R, Joseph A, Katz R, Konwinski A, et al. Above the clouds: a Berkeley view of cloud computing. Tech. rep. UCB/EECS-2009-28, EECS Department, University of California; February 2009.
[5] Axelrod R. The dissemination of culture: a model with local convergence and global polarization. J Conflict Resolut 1997;41(2):203–26.
[6] Banerjee S, Mukherjee I, Mahanti P. Cloud computing initiative using modified ant colony framework. In: World academy of science, engineering and technology, WASET, 2009. p. 221–4.
[7] Basney J, Livny M, Mazzanti P. Harnessing the capacity of computational grids for high energy physics. In: Conference on computing in high energy and nuclear physics, Padova, Italy, 7–11, 2000. p. 610–13.
[8] Beck. Alan (1997-06-27, High Throughput Computing: An Interview with Miron Livny, 1997. <http://research.cs.wisc.edu/htcondor/HPCwire.1>.

[9] Beer C, Hendtlass T, Montgomery J. Improving exploration in ant colony optimisation with antennation. In: 2012 IEEE congress on evolutionary computation (CEC), 2012. p. 1–8.

[10] Bonabeau E, Dorigo M, Theraulaz G. Swarm intelligence: from natural to artificial systems. Oxford University Press; 1999.

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

[12] Calheiros R, Ranjan R, Beloglazov A, De Rose C, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw: Pract Exper 2011;41(1):23–50.

[13] Careglio C, Monge D, Pacini E, Mateos C, Mirasso A, García Garino C. Sensibilidad de resultados del ensayo de tracción simple frente a diferentes tamaños y tipos de imperfecciones. In: Dvorkin MGE, Storti M, editors. Mecánica Computacional, vol. XXIX. AMCA; 2010. p. 4181–97.

[14] Cherkasova L, Gupta D, Vahdat A. When virtual is harder than real: resource allocation challenges in virtual machine based it environments. Tech. rep., HP Laboratories. Technical Report HPL-2007-25, Palo Alto; February 2007. <http://www.hpl.hp.com/techreports/2007/HPL-2007-25.html>.

[15] Dhinesh Babu L, Venkata Krishna P. Honey bee behavior inspired load balancing of tasks in cloud computing environments. Appl Soft Comput 2013;13(5):2292–303.

[16] Dorigo M. Optimization, learning and natural algorithms. Phd thesis, Italy (Milano, Italy): Politecnico di Milano; 1992.

[17] Dorigo M, Stützle T. The ant colony optimization metaheuristic: algorithms, applications, and advances. In: Glover F, Kochenberger G, editors. Handbook of metaheuristics. International series in operations research and management science, vol. 57. New York: Springer; 2003. p. 250–85 [chapter 9].

[18] García Garino C, Gabaldón F, Goicolea JM. Finite element simulation of the simple tension test in metals. Finite Elem Anal Des 2006;42(13):1187–97.

[19] García Garino C, Mateos C, Pacini E. Job scheduling of parametric computational mechanics studies on cloud computing infrastructures. In: International advanced research workshop on high performance computing, grid and clouds. Cetraro (Italy); June 2012. <http://www.hpcc.unical.it/hpc2012/pdfs/garciagarino.pdf>.

[20] García Garino C, Mirasso A, Raichman S, Goicolea J. Imperfection sensitivity analysis of necking instability of circular cilyndrical bars. In: Owen DRJ, et al., editors. Computational plasticity: fundamentals and applications, vol. 1, International Center for Numerical Methods in Engineering (CIMNE); 1997. p. 759–64.

[21] García Garino C, Ribero Vairo M, Andía Fagés S, Mirasso A, Ponthot J-P. Numerical simulation of finite strain viscoplastic problems. J Comput Appl Math 2013;246:174–84.

[22] Goiri I, Berral J, Oriol Fitó J, Juliá F, Nou R, Guitart J, et al. Energy-efficient and multifaceted resource management for profit-driven virtualized data centers. Fut Gener Comput Syst 2012;28(5):718–31.

[23] Gulamali M, Mcgough A, Newhouse S, Darlington J. Using ICENI to run parameter sweep applications across multiple Grid resources. In: Global grid forum 10, case studies on grid applications workshop, Berlin, Germany, 2004.

[24] Huang L, Chen H, Hu T. Survey on resource allocation policy and job scheduling algorithms of cloud computing. J Softw 2013;8(2):480–7.

[25] Huang W, Liu J, Abali B, Panda D. A case for high performance computing with virtual machines. In: Proceedings of the 20th annual international conference on supercomputing. ICS '06. New York (NY, USA): ACM; 2006. p. 125–34.

[26] Jeyarani R, Nagaveni N, Vasanth Ram R. Design and implementation of adaptive power-aware virtual machine provisioner (APA-VMP) using swarm intelligence. Fut Gener Comput Syst 2012;28(5):811–21.

[27] Lipowski A, Lipowska D. Roulette-wheel selection via stochastic acceptance. Physica A 2012;391(6):2193–6.

[28] Liu Y, Zhu H. A survey of the research on power management techniques for high-performance systems. Softw Pract Exper 2010;40(11):943–64.

[29] Ludwig S, Moallem A. Swarm intelligence approaches for grid load balancing. J Grid Comput 2011;9(3):279–301.

[30] Mateos C, Pacini E, García Garino C. An ACO-inspired algorithm for minimizing weighted flowtime in cloud-based parameter sweep experiments. Adv Eng Softw 2013;56:38–50.

[31] Mavrovouniotis M, Yang S. Ant colony optimization with direct communication for the traveling salesman problem. In: 2010 UK workshop on computational intelligence (UKCI), 2010. p. 1–6.

[32] Mell P, Grance T. The NIST definition of cloud computing. Nat Inst Stand Technol 2009;53(6):50.

[33] Merkle D, Middendorf M, Schmeck H. Ant colony optimization for resource-constrained project scheduling. IEEE Trans Evol Comput 2002;6(4):333–46.

[34] Oden JT, Bathe KJ. A commentary on computational mechanics. Appl Mech Rev 1978;31(8):1053–8.

[35] Pacini E, Mateos C, García Garino C. Schedulers based on ant colony optimization for parameter sweep experiments in distributed environments. In: Siddhartha Bhattacharyya Dr, Paramartha Dutta Dr, editors. Handbook of research on computational intelligence for engineering, science and business, vol. I. IGI Global; 2012. p. 410–47 [chapter 16].

[36] Pacini E, Mateos C, García Garino C. Dynamic scheduling of scientific experiments on clouds using ant colony optimization. In: Topping BHV, Iványi P, editors. Proceedings of the third international conference on parallel, distributed, grid and cloud computing for engineering. Stirlingshire (UK): Civil-Comp Press; 2013 [paper 33]. <http://dx.doi.org/10.4203/ccp.101.33>.

[37] Pacini E, Ribero M, Mateos C, Mirasso A, García Garino C. Simulation on cloud computing infrastructures of parametric studies of nonlinear solids problems. In: Cipolla-Ficarra FV et al., editors. Advances in new technologies, interactive interfaces and communicability (ADNTIIC 2011). LNCS, vol. 7547. Springer-Verlag; 2011. p. 58–70.

[38] Pandey S, Wu L, Guru S, Buyya R. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: International conference on advanced information networking and applications. IEEE Computer Society; 2010. p. 400–7.

[39] Pedemonte M, Nesmachnow S, Cancela H. A survey on parallel ant colony optimization. Appl Soft Comput 2011;11(8):5181–97.

[40] Simo J. A framework for finite strain elastoplasticity based on maximum plastic dissipation and the multiplicative decomposition: Part I. Continuum formulation. Comput Methods Appl Mech Eng 1988;66:199–219.

[41] Simo J. A framework for finite strain elastoplasticity based on maximum plastic dissipation and the multiplicative decomposition. Part II: Computational aspects. Comput Methods Appl Mech Eng 1988;68(1):1–31.

[42] Simo J, Ortiz M. A unified approach to finite deformation elastoplastic analysis based on the use hiperelastic constitutive equation. Comput Methods Appl Mech Eng 1985;49:221–45.

[43] Sotomayor B, Keahey K, Foster I, Freeman T. Enabling cost-effective resource leases with virtual machines. In: Hot topics session in ACM/IEEE international symposium on high performance distributed computing 2007, Monterey Bay (CA, USA); 2007.

[44] Sun C, Kim B, Yi G, Park H. A model of problem solving environment for integrated bioinformatics solution on grid by using condor. In: Grid and cooperative computing, 2004. p. 935–8.

[45] Taheri J, Lee YC, Zomaya AY, Siegel H. A bee colony based optimization approach for simultaneous job scheduling and data replication in grid environments. Comput Oper Res 2011;40(6):1564–78.

[46] Tavares Neto R, Godinho Filho M. Literature review regarding ant colony optimization applied to scheduling problems: guidelines for implementation and directions for future research. Eng Appl Artif Intell 2013;26(1):150–61.

[47] Tinghuai M, Qiaoqiao Y, Wenjie L, Donghai G, Sungyoung L. Grid task scheduling: algorithm review. IETE Tech Rev 2011;28(2):158–67.

[48] Vaquero L, Rodero-Merino L, Caceres J, Lindner M. A break in the clouds: towards a cloud definition. ACM SIGCOMM Comput Commun Rev 2009;39(1):50–5.

[49] Vecchiola C, Pandey S, Buyya R. High-performance cloud computing: a view of scientific applications. In: Proceedings of the 2009 10th international symposium on pervasive systems, algorithms, and networks. ISPAN 09. Washington (DC, USA): IEEE Computer Society; 2009. p. 4–16.

[50] Wang L, Kunze M, Tao J, von Laszewski G. Towards building a cloud for scientific applications. Adv Eng Softw 2011;42(9):714–22.

[51] Wang L, Tao J, Kunze M, Castellanos AC, Kramer D, Karl W. Scientific cloud computing: early definition and experience. In: 10th IEEE international conference on high performance computing and communications. Washington (DC, USA): IEEE Computer Society; 2008. p. 825–30.

[52] Woeginger G. Exact algorithms for NP-hard problems: a survey. In: Junger M, Reinelt G, Rinaldi G, editors. Combinatorial optimization – Eureka. You Shrink!, Lecture notes in computer science, vol. 2570. Berlin/Heidelberg: Springer; 2003. p. 185–207.

[53] Wozniak J, Striegel A, Salyers D, Izaguirre J. GIPSE: streamlining the management of simulation on the grid. In: 38th Annual simulation symposium, 2005. p. 130–7.

[54] Xhafa F, Abraham A. Computational models and heuristic methods for grid scheduling problems. Fut Gener Comput Syst 2010;26(4):608–21.

[55] Xiaoyong B. High performance computing for finite element in cloud. In: International conference on future computer sciences and application (ICFCSA). IEEE Computer Society; 2011. p. 51–3.

[56] Zehua Z, Xuejie Z. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In: 2nd International conference on industrial mechatronics and automation. IEEE Computer Society; 2010. p. 240–3.