Journal of Intelligent & Fuzzy Systems 31 (2016) 1731–1743
DOI:10.3233/JIFS-152094
IOS Press

A bio-inspired scheduler for minimizing makespan and flowtime of computational mechanics applications on federated clouds

Elina Pacini[a,b,d,*], Cristian Mateos[c,d], Carlos García Garino[a,e], Claudio Careglio[a,e] and Aníbal Mirasso[e]
[a]*ITIC Research Institute – UNCuyo University, Mendoza, Argentina*
[b]*Facultad de Ciencias Exactas y Naturales – UNCuyo University, Mendoza, Argentina*
[c]*ISISTAN Research Institute – UNICEN University, Tandil, Buenos Aires, Argentina*
[d]*Consejo Nacional de Investigaciones Científicas y Técnicas, Argentina*
[e]*Facultad de Ingeniería – UNCuyo University, Mendoza, Argentina*

Abstract. Computational Mechanics (CM) concerns the use of computational methods to study phenomena under the principles of mechanics. A representative CM application is parameter sweep experiments (PSEs), which involves the execution of many CPU-intensive jobs and thus computing environments such as Clouds must be used. We focus on federated Clouds, where PSEs are processed via virtual machines (VM) that are lauched in hosts belonging to different datacenters, minimizing both the makespan and flowtime. Scheduling is performed at three levels: a) broker, where datacenters are selected based on their network latencies via three policies, b) infrastructure, where two bio-inspired schedulers based on Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) for VM-host mapping in a datacenter are implemented, and c)VM, where jobs are assigned into the preallocated VMs based on job priorities. Simulated experiments performed with job data from two real PSEs show that our scheduling approach allows for a more agile job handling while reducing PSE makespan and flowtime.

Keywords: Cloud computing, computational mechanics, scheduling, ant colony optimization, particle swarm optimization

1. Introduction

CM involves the use of computational approaches to characterize and simulate physical events and engineering systems governed by the laws of mechanics. PSEs are CM simulations that require performing repeated analyses, where certain input parameters are varied among those defining the problem of interest. PSE users need a computing environment that delivers large amounts of computational power over a long period of time, such as Clouds [4]. However, since in single-datacenter Clouds resource availability might be limited, the option of obtaining extra resources from an arrangement of Cloud providers –or *federating Clouds*– is an appealing solution [4].

For efficiently executing PSEs in federated Clouds it is necessary to properly manage physical resources from geographically distributed datacenters. Therefore, job scheduling should be performed at three levels [24]. At the broker level, scheduling strategies are used for selecting datacenters considering factors such as network interconnections or monetary cost of allocating VMs on hosts. At the infrastructure

*Corresponding author. Elina Pacini. Tel./Fax: +54 (261) 4291000; E-mail: epacini@uncu.edu.ar.

level, the VMs are allocated on the available hosts of the previously selected datacenters. Lastly, at the VM level, jobs are assigned for execution into the allocated VMs. However, scheduling is NP-complete, and the fact that federated Cloud scheduling spans these three levels makes the problem even more challenging.

Swarm Intelligence (SI) metaheuristics have been proposed to solve combinatorial optimization problems –such as job/VM scheduling– by simulating the collective behavior of social insects swarms [14]. The most popular SI-based strategies are ACO and PSO. Even when job scheduling in Clouds has been approached using SI [22], very few efforts applying SI have been proposed for federated Clouds [22] though.

We propose a three-level scheduler for federated Clouds that exploits SI and the concept of job priorities. At the broker level, datacenters are selected according to their network latencies. To this end, we consider three policies, Lowest-Latency-Time-First (LLTF), First-Latency-Time-First (FLTF), and Latency-Time-In-Round (LTIR). Then, at the infrastructure level, we explore ACO and PSO for allocating the VMs in a datacenter. Finally, at the VM level, PSE jobs are assigned to the preallocated VMs by using a priority-based policy, as in [20].

We aim to minimize *makespan*, i.e., the total execution time of all jobs, and *weighted flowtime*, i.e., the weighted sum of job finish times minus job start times. As in [29], where it is shown that the application of scheduling strategies both at the infrastructure level and VM level within a single-datacenter Cloud improves resource usage, here we show how the use of scheduling strategies at the three levels improve the overall system performance in federated Clouds.

Unlike previous works [23, 25], where schedulers take advantage of a single datacenter, here we extend our scheduler for operating in federated Clouds with heterogeneous hosts, which is a common scenario. In addition, we deepen the experimental analysis by incorporating a new PSE from CM [3] and we evaluate the performance of our approach using the PSE studied in [10]. This has enabled more realistic experimental conditions given by jobs that are much more CPU-intensive, and variable job execution times [10]. Unlike [23, 25], where we used a simple FIFO policy at the VM level, we incorporate the priority-based policy.

Experiments performed with job execution data extracted from these two real-world PSEs suggest that the SI schedulers at the infrastructure level, in combination with broker-level policies and the VM-level priority-based policy, deliver competitive makespan and weighted flowtime. Since VM scheduling is highly challenging and heavily contributes to the overall performance in Cloud scheduling [30], for comparison purposes, we used the same three policies at the broker level and the priority-based policy at the VM level in combination with two alternative schedulers based on Genetic Algorithms (GA) [1] and Best Effort (BE).

Section 2 surveys and analyses relevant related works. Section 3 presents our proposal and the involved techniques at each level. Then, in Section 4 we present the performed experiments. Finally, Section 5 concludes the paper and discusses future extensions.

2. Related work

SI techniques, specially ACO and PSO, have been the focus of many research studies for solving combinatorial optimization problems [6, 17, 28]. Specifically, these techniques have been increasingly applied to distributed job scheduling [28, 31]. However, no efforts aimed to the three scheduling levels in federated Clouds where the authors also consider SI exist.

First, our approach differs from those in the literature since we have considered SI-based strategies at the infrastructure level. In other works of our own [21, 23] we have presented SI-based schedulers focused on the infrastructure level, but they target single--datacenter Clouds. Then, in [24] we extended the scheduler to operate in federated Clouds composed of homogeneous datacenters. Another important distinction of this work with respect to [24] is that we have considered a priority-based policy at the VM level.

Second, works found in the literature are focused on one Cloud level and do not consider the three levels as we propose in this work. Most current Cloud brokers do not provide advanced capabilities to make automatic decisions, about how to efficiently distribute the different VMs and jobs of an application among providers [30]. Discusses a Cloud brokering approach based on integer programming that restricts the deployment of VMs across multiple heterogeneous datacenters according to some placement constraints (e.g., Clouds to deploy the VMs) defined by the user. Users can also steer the VM allocation by specifying maximum budget and minimum performance, or other constraints (load balance, hardware

configuration of VMs, etc.). In [15, 16] different strategies at the broker level to optimize the scheduling of jobs across multiple providers are proposed. [15] introduces a multi-objective genetic algorithm (MO-GA) for job scheduling to optimize three objectives namely, energy consumption, CO_2 emission, and the generated profit of distributed datacenters. In [16] the scheduler performs a deployment of the jobs among datacenters by optimizing a particular cost function based on optimization criteria (e.g., monetary cost or performance) and user constraints (e.g., budget, performance, VMs types).

Agostinho et al. [1] uses at the broker level the Dijkstra algorithm [26] to select the datacenter with the lowest monetary cost, and a GA for allocating VMs to hosts. Although [1] targets the broker and the infrastructure levels, the goal was not to reduce makespan, which for scientific applications allows users to accelerate result processing [23]. In [5] an ACO scheduler to distribute jobs in VMs is proposed, minimizing the makespan and improve load balancing in the VMs. However, the work focus on assigning jobs assuming the existence of pre-allocated VMs. So far [5] is the only work using SI for federated Clouds, but SI was applied at the VM level and not at the infrastructure level.

With respect to works which address the scheduling problem at the infrastructure level using SI as we propose in this work, we can mention [7, 8]. In [7] the authors proposed a VM scheduler based on ACO to perform the dynamical placement of VMs according to the current load on physical machines, minimizing energy consumption. The work targets single-datacenter Clouds. In [8] the authors proposed a multi-objective ACO for the VM allocation problem to simultaneously maximize total resource utilization and minimize power consumption. Moreover, in [12] the authors proposed a PSO algorithm to efficiently map VM instances into physical machines while reducing energy consumption. This algorithm makes the best possible use of the power saving states of idle physical machines and instantaneous workload on the operational physical machines. However, although in these works SI-based algorithms at the infrastructure level have been used, the schedulers do not target federated Clouds. Besides, these works do not consider flowtime, and only in [12] makespan is considered.

From the works discussed, most of them take into account only one of the Cloud scheduling levels, without considering metrics such as makespan and weighted flowtime, which difficult their applicability to execute PSEs in federated Clouds.

3. Approach overview

The goal of our scheduler is to minimize the makespan and weighted flowtime of a set of PSE jobs, when the jobs are executed in a federated Cloud with heterogeneous hosts. Makespan is the period of time in which a user requests VMs to execute its PSE, until all the PSE jobs finish their execution. Flowtime is the total time that a job spends in the system, i.e., waiting time plus effective processing time. Moreover, we have considered weighted flowtime, where the sum of times for a job are weighted according to the job priority.

The proposed scheduler associates a *qualitative* priority represented as an integer value for each one of the jobs of a PSE. When designing a PSE, each job is fed with a particular value for the *ith* variable of the model being studied. Hence, job execution times can be very different, since running the same solver against many input values might yield dissimilar execution times as well. This is very undesirable since, unless the scheduler knows some job information, the user can not process/visualize the outputs of the whole PSE until all jobs finish. Thus, giving higher (or lower) priority to jobs that are supposed to take longer to finish may help in improving output processing [11, 20].

In this work, priorities are provided by a disciplinary user, who has knowledge about the problem to be solved from a modeling perspective, and therefore can estimate the time requirements of each job relative to the rest of the jobs in a PSE. Once the disciplinary user has identified the experiments that might require more execution time, a simple tagging strategy is applied to assign a "category" (number) to each job. These categories represent the priority degree of a job with respect to the others in the same PSE. For usability reasons, the number of categories are reduced to three, i.e., high priority, medium priority or low priority.

Formally, a PSE is a set of $N = 1, 2, ..., n$ independent jobs, which are executed on m Cloud machines. Each job j has an associated priority value, which is represented by a weight w_j. This priority value is taken into account by the scheduler to determine the order in which jobs will be executed at the VM level. The scheduler processes the jobs with higher priority (or heavier) first. The larger the estimated size of a job in terms of execution time, the higher priority weight the user should associate to the job. The makespan of a job j in schedule S can be denoted by $C_j(S)$ and hence the makespan is $C_{max}(S) = max_j C_j(S)$.

Furthermore, the total weighted flowtime is calculated as $\sum_j^n (C_j(S) - A_j(S)).w_j$, where C_j is the completion time of job j, A_j is the starting execution time of job j and w_j is the weight associated to job j.

The proposed scheduler proceeds as follows. Firstly, at the broker level, a datacenter is selected based on a policy that considers the datacenter which provides the lowest communication latency to a broker when this latter asks about the availability of physical resources. Latency is due to delays by packets moving over the various networks between the end user computer and the distributed datacenters. One way to mitigate the effects of such latencies is to choose a datacenter with a fast internal network and plenty of capacity. In this work, each broker has three available policies to select datacenters, i.e., *LFTF, FLTR and LTIR*. Secondly, at the infrastructure level, by means of a SI-based VM scheduler which implements *ACO and PSO*, user VMs are allocated in the physical resources (i.e., hosts) belonging to the selected datacenter at the broker level. When there are no available hosts in the datacenter to allocate the VMs, a new datacenter is selected at the broker level. Finally, at the VM level, jobs are assigned to the preallocated VMs through a *Job Priority Policy*.

3.1. Scheduler at the Broker level

For executing jobs in federated Clouds, a broker is created for each user that connects to the Cloud, which knows the datacenters of the federation. The scheduler at the broker level is executed to select the first datacenter to allocate the VMs, which are managed by the scheduler implemented at the infrastructure level. Furthermore, the scheduler at this level can decide to deploy the VMs in a remote datacenter when there are insufficient physical resources in the datacenter where the VM creation was issued. As mentioned, the policies studied at this level are LLTF, FLTF and LTIR.

LLTF maintains a list of all network interconnected datacenters sorted by their latencies. This policy selects the datacenter with the lowest latency. Then, whenever a datacenter has no more physical resources to allocate VMs, the algorithm selects the next datacenter in the list with low latency.

FLTF selects the first datacenter from a list sorted randomly, containing all network interconnected datacenters to which a user can access and allocate his/her VMs. When the selected datacenter has no more available physical resources to allocate VMs, the algorithm selects the next datacenter in the list.

Lastly, LTIR maintains a list of all network interconnected datacenters that make up the Cloud, sorted by increasing latency, and assigns each VM required by the user to a datacenter from the list in a circular order.

3.2. Scheduler at the infrastructure level

3.2.1. Variant based on ACO

In this algorithm, each ant works independently and represents a VM "looking" for the best host to which it can be allocated. When a VM is issued in a datacenter, an ant is initialized. Figure 1 shows the activity diagram of the algorithm. In the first activity, the *step* parameter keeps track of the number of steps carried out by an ant, and *maxStep* is a predefined number of steps, i.e., the completion criterion. Since datacenters have different numbers of hosts, *maxSteps* varies depending on the datacenter being explored according to a user-defined percentage value. Then, a list of all suitable hosts belonging to the selected datacenter in which the VM can be allocated is obtained (*Get suitable hosts*). A host is suitable if it has an amount of processing power, storage, memory and bandwidth greater than or equal to that of required by the unallocated VM. The ant is initialized in one of the obtained hosts, randomly. Then, a local table containing information on the load of each host is initialized (*Initialize LoadTable*). Each host has associated a single *LoadTable*.

In each iteration, the ant collects the load information of the host that is visiting (*Get load information*) and adds this information to its private load table (*Add load to LoadTable*). Here, load refers to the total CPU utilization within a host and is calculated 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.

Since each ant step involves moving through the intra-datacenter network to get the availability of the hosts from the selected datacenter, it incurs in latencies. We have added a control to minimize the number of steps performed by an ant: every time an ant visits a host that has not allocated VMs yet, the ant allocates its associated VM to it directly without performing further steps (*Deliver VM to host*). The smaller the number messages sent to the hosts through the network, the smaller the impact of the latencies in the makespan and flowtime given to the user.

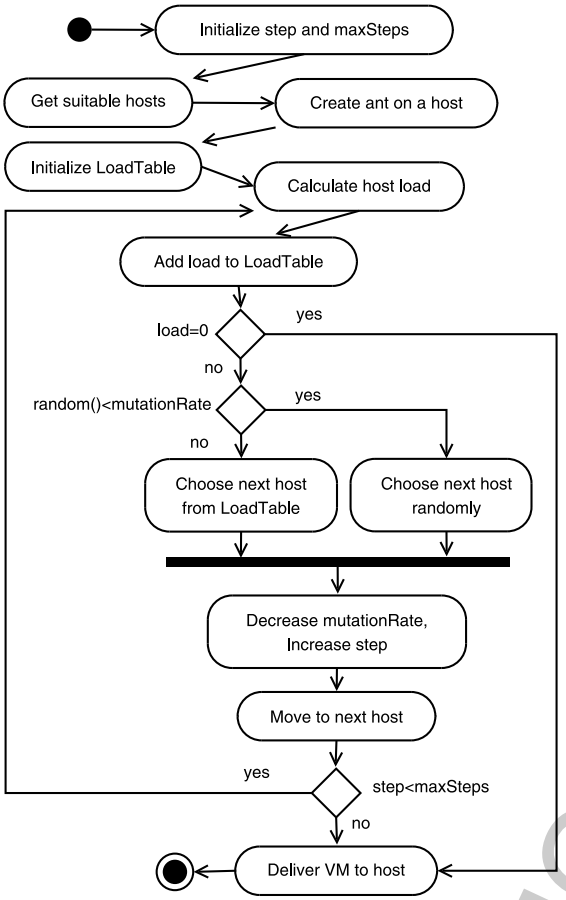


Fig. 1. ACO-based allocation algorithm for individual VMs.

Later, if the host has allocated VMs, the ant looks for the least loaded host. The ant might move either selecting a random host using a constant probability or *mutation rate*, or using the load table of the current host. The mutation rate decreases with a *decay rate* factor as time passes, thus, the ant will be less dependent on random choice. Upon visiting a host, the ant updates the host load table with the information of other hosts in the datacenter, and collects the information provided by the table of that host, if any. The load table acts as a pheromone trail that an ant leaves to guide other ants to choose better paths rather than wandering randomly. Entries of each load table are the hosts that ants have visited on their way to deliver their VMs together with their load information. When an ant reads the information from a load table in a host, the ant chooses the least loaded host in the table. If the load of the visited host is smaller than any other host in the load information table, the ant chooses the host with the smallest load. This process is repeated until *step=maxStep*. Finally, the ant delivers its VM to the current host.

3.2.2. *Variant based on PSO*

Here, each particle represents a VM looking for the best host to execute. An example based on nature to illustrate PSO is as follows: some bees fly over the countryside looking for as many flowers as possible. Initially, bees do not have knowledge of the field and fly to random locations and velocities. Each bee can remember the places where it saw the most flowers, and somehow knows the places where other bees have found a high density of flowers. These two pieces of information are used by the bees to continually modify their trajectory to find a greater density of flowers.

Each VM is considered a bee and each host represent locations in the field with different density of flowers. shows the activity diagram of the algorithm. Like the maxSteps parameter of ACO, the size of the particle neighborhood –*neighborhoodSize*– varies depending on the datacenter which is being explored according to a predefined percentage value. Every time a user requires a VM, a particle is initialized in a random host of the selected datacenter. The density of flowers of each host is its load as in ACO, i.e., the *Calculate host load* activity. This definition helps to search in the load search space –in the field of flowers– and try to minimize the load. The smaller the load on a host, the better the flower concentration. Subsequently, the neighborhood of each particle is obtained and it is composed by the remaining hosts in a datacenter excluding the one in which the particle is initialized. Each one of the neighbors –hosts– that compose the neighborhood are selected randomly.

In each iteration of the algorithm (see Fig. 2), the particle moves to the neighbors of its current host in search of a host with a lower load. Similarly to ACO, since each move a particle performs involves traveling through the intra-datacenter network, a control to minimize the number of moves that a particle performs have been added: every time a particle moves from the associated host to a neighbor host that has not allocated VMs yet, the particle allocates its associated VM to it immediately (*Deliver VM to host*). If the host load is not equal to zero, then the particle moves to the host with a greater velocity. The velocity is defined by the load difference between the host to which the particle has been previously assigned with respect to its other neighboring hosts. If any of the hosts in the neighborhood is less loaded than the

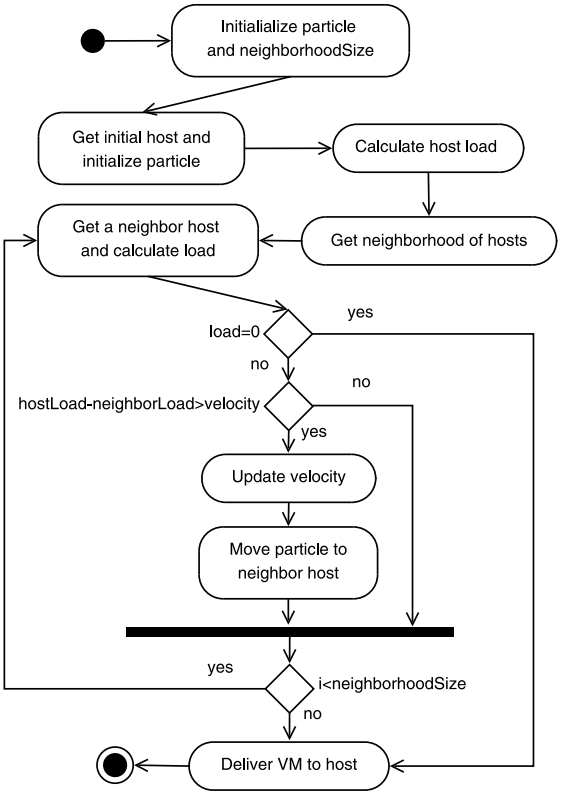


Fig. 2. PSO-based allocation algorithm for individual VMs.

original host, then the particle is moved to the neighbor host with a greater velocity. Due to particles move through hosts of their neighborhood into a datacenter in search of a host with the lower load, the algorithm reaches a local optimum quickly. Thus, each particle makes a move from their associated host to one of its neighbors, which has the minimum load among all. If all its neighbors are busier than the associated host itself, the particle is not moved from the current host. Finally, the particle delivers its associated VM to the host with the lower load among their neighbors.

3.3. Scheduler at the VM level

This scheduler uses two lists, one with 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 machine and hence are ready to execute jobs. The scheduler iterates the list of all jobs –*jobList*– and then, through *Get job by priority from jobList* activity retrieves jobs according to their priority value, this means, jobs with the highest priority first, then jobs with medium priority value, and finally jobs with low priority. 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 *Get VM form VMList* activity. Internally, the scheduler maintains a queue for each VM that contains its list of jobs to execute. Every time a job is obtained from the *jobList* to be executed, it is also deleted from the list. The procedure is repeated until all jobs have been submitted for execution.

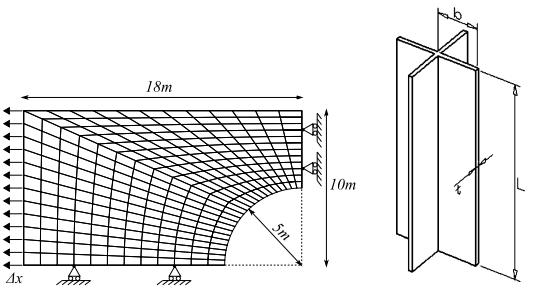
4. Evaluation

We processed two PSEs by using a finite element software to gather real job processing times (subsection 4.1). Based on these, we instantiated the CloudSim simulator [2] (subsection 4.2). Lastly, we compared our proposal with some alternatives for assigning VMs to hosts (subsection 4.3).

4.1. Case studies

A classical benchmark problem [10] involves studying a plane strain plate with a central circular hole (PSE-1). The geometry of the plate is shown in Fig. 3a. The 3D finite element mesh employed had 1,152 elements. To generate the PSE jobs, a material parameter –viscosity η– was selected as the variation parameter. Then, 25 different values for η were considered: $x.10^y$ Mpa, with $x = 1, 2, 3, 4, 5, 7$ and $y = 4, 5, 6, 7$, plus 1.10^8 Mpa.

The second problem (PSE-2) was the elastoplastic buckling behavior of cruciform columns [18]. The geometry of the column is shown in Fig. 3b. The total number of finite elements of the mesh was 2,176. Moreover, 30 different angle values for the α parameter were considered, namely $\alpha_n = \alpha_{n-1} + 0.25$, with $\alpha_0 = 0.5$ and $n = 1, 2, ..., 30$.



(a) Plane strain plate (b) Cruciform column

Fig. 3. Studied PSEs: Geometry.

Table 1
Simulated Cloud machines characteristics

Characteristic	D1	D2	D3	D4	D5
Power (MIPS)	7,200	5,618	8,036	6,600	6,185
RAM (GB)	32	12	16	12	16
# of cores	8	6	8	8	6

Table 2
Job priorities

Job priority	Value in MI (from-to)	
	Plane strain plate	Cruciform column
Low; $w_j = 1$	1,300,291-1,379,496	13,359,331-14,798,231
Medium; $w_j = 2$	1,405,898-1,570,909	15,055,649-15,398,873
High; $w_j = 3$	1,689,718-2,712,789	15,405,474-15,603,487

4.2. CloudSim instantiation

After establishing the parameters for each PSE, we used a single computer to run the experiments. The execution of 25 PSE-jobs (plane strain plate) and the 30 PSE-jobs (cruciform column) resulted, respectively, in 25 and 30 input/output files with different input configurations for each PSE. Both PSEs were solved using the SOGDE finite element solver [9]. Once the execution times were obtained using this computer, we approximated for each experiment the number of executed CPU instructions via $NI_i = mipsCPU * T_i$, where NI_i is the number of Million Instructions (MI) associated to a job *i*, $mipsCPU$ is the processing power of the CPU of our real computer in MIPS, and T_i is the time that took to run the job *i* on the real computer.

The experimental scenario consists of a Cloud composed of 5 heterogeneous datacenters. The network topology is defined in the BRITE [13] format. A BRITE file is used by CloudSim to define the different nodes that compose a commonly-found federation (i.e., datacenters, brokers) and their network connections. Each datacenter is composed of 20 physical machines and, as in other works in the literature [19, 27], has an associated latency of 1.5, 0.8, 1, 0.15 and 2 seconds, respectively. The characteristics of the machines that compose datacenters *D1*, *D2, D3, D4* and *D5* are shown in Table 1. In all cases, 1 Gbps networks were used. Moreover, an user requests 100 VMs to execute its PSE. Each VM has the same characteristics as a *t2.small* instance of Amazon EC2 (1 core), since the SOGDE code is a monolithic application, and therefore, jobs need only one core to be executed.

Each job had between 1,333,293 and 2,712,789 MI for the plane strain plate and between 13,359,331 and 15,603,487 for the cruciform column. The experiments of the plane strain plate had input files of 291.7 Kbytes, and 249.9 Kbytes for those of the cruciform column. A similar distinction applies to the output file sizes. Table 2 shows the priorities assigned to jobs. Then, for each PSE, we evaluated their performance as we increased the number of jobs to be performed, i.e., $25 * i$ jobs and $30 * i$ jobs with $i = 40, 80, ..., 400$. This is, the base job set comprising 25 jobs of the plane strain plate PSE obtained by varying η, and the base job set comprising 30 jobs of the cruciform column PSE by varying α, were cloned to obtain larger sets.

4.3. Performed experiments

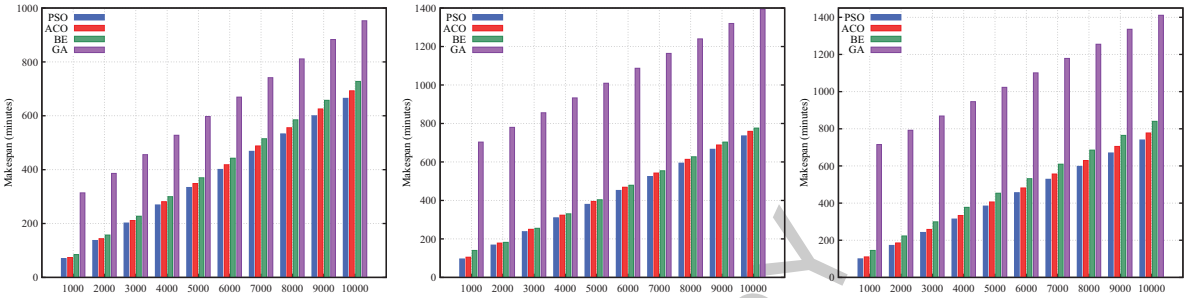
Due to their high CPU requirements, jobs within a VM compete for CPU time with other jobs from other VMs in the same hosts. In other words, a time-shared CPU scheduling policy was used, which ensures fairness. Particularly, we study/combine the three policies for selecting datacenters at the broker level, and at the infrastructure level we use ACO and PSO while comparing them against Best Effort (BE) and the scheduler based on Genetic Algorithm (GA) from [1].

BE chooses the host with less cores in use upon allocating a VM. To do this, the broker sends a message to all hosts in the selected datacenter to know their state. In the GA, the population is the set of physical resources in a datacenter. Each chromosome (individual) represents a part of the search space. Each gene is a host in a datacenter, and the *fitness* field indicates the suitability of the hosts in each chromosome. Fitness is calculated as the inverse of the accumulated load of all hosts composing the chromosome. Load is calculated considering the number of VMs executing in a host. A chromosome with higher fitness is always desirable.

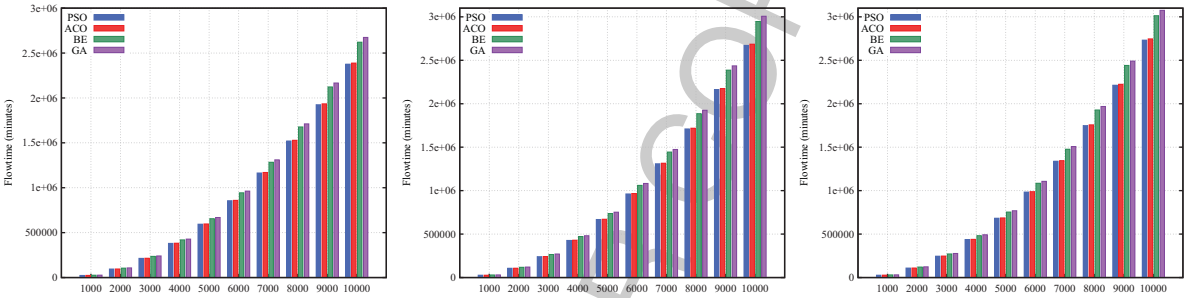
The specific-parameter of each algorithm (e.g., neighborhood size in PSO, maximum steps in ACO and chromosome size in GA), has been configured so as to explore up to 60% of the number of hosts of each datacenter. Furthermore, in the ACO algorithm we have set the mutation rate and decay rate parameters to 0.6 and 0.1, respectively, and the GA population size is 100. For simplicity, we will refer to *"weighted flowtime"* as *"flowtime"*. In all cases, the competing policies both a the broker level and the infrastructure level were also complemented with the VM-level priority-based policy for handling jobs within VMs.

4.3.1. Effects of number of jobs

Figures 4 and 5 compare the makespan and flowtime for each one the policies at the broker level (LLTF, FLTF, LTIR) and all the considered scheduling algorithms (PSO, ACO, BE, GA) at the infrastructure level. Irrespective of the PSE,

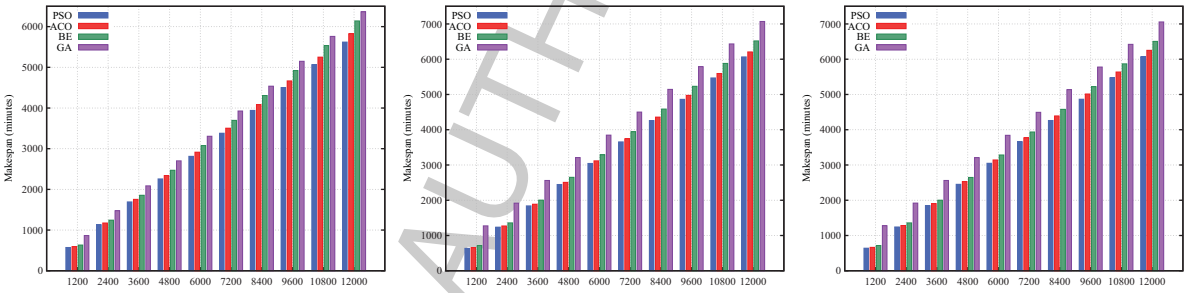


(a) Makespan. From left to right: LLTF, FLTF and LTIR

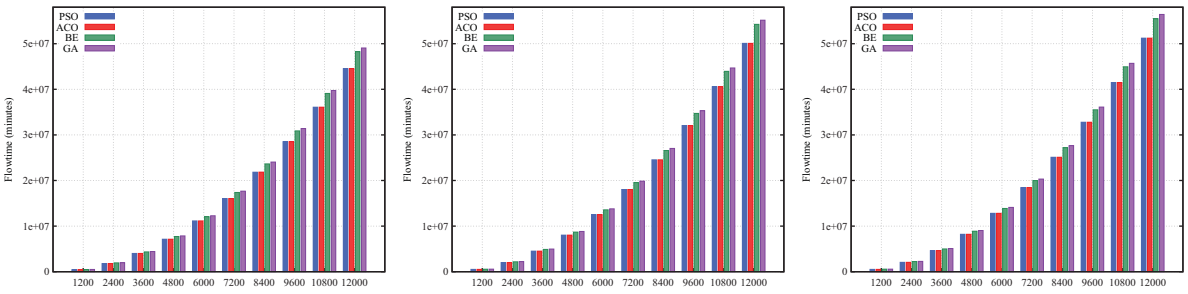


(b) Flowtime. From left to right: LLTF, FLTF and LTIR

Fig. 4. PSE-1: Results as the number of jobs increases.



(a) Makespan. From left to right: LLTF, FLTF and LTIR



(b) Flowtime. From left to right: LLTF, FLTF and LTIR

Fig. 5. PSE-2: Results as the number of jobs increases.

makespan and flowtime presented linear and exponential tendencies, respectively.

At the broker level, when LLTF is used in combination with PSO, ACO, BE and GA, both the makespan and flowtime decrease w.r.t. FLTF and LTIR policies. This happens because most VMs are allocated in datacenters with lower latencies, and therefore they have less influence in the makespan when PSO, ACO, BE and GA send network messages to the hosts to inquire about their availability. For example, the makespan for PSE-1, when the LLTF policy is used and the number of jobs to execute increases to 10,000 (first subfigure in 4a), the makespan is 664.75, 692.81, 727.51 and 952.65 minutes, for PSO, ACO, BE and GA, respectively. Then, when FLTF is used (second subfigure in 4a) the makespan for PSO, ACO, BE and GA reaches 732.57, 759.23, 775.94 and 1,394.67 minutes, respectively, when 10,000 jobs are executed. When LTIR is used, the makespan is 740.74, 777.60, 840.05 and 1,411.05 minutes (third subfigure in 4a), when executing 10,000 jobs via PSO, ACO, BE and GA, respectively. Like the makespan, the flowtime is lower for all the algorithms when LLTF is used. Briefly, the average gain obtained by LLTF with regard to FLTF is 17.05% and about 19.41% compared to LTIR.

Likewise, for the PSE-2, when LLTF is used and the number of jobs to be executed is 12,000 (first subfigure in 5a), the makespan is 5,620.53, 5,825.82, 6,140.81 and 6,364.92 minutes, for PSO, ACO, BE and GA, respectively. Then, when FLTF is used (second subfigure in 5a) the makespan for PSO, ACO, BE and GA reaches 6,064.37, 6,205.22, 6,504.55 and 7,056.55 minutes, respectively, when 12,000 jobs are executed. When LTIR is used, the makespan is 6,068.54, 6,251.19, 6,539.15 and 7,075.16 minutes (third subfigure in 5a), when the number of jobs was 12,000, and for PSO, ACO, BE and GA, respectively. The average gain from using LLTF with regard to FLTF is 7.28% and about 7.64% compared to LTIR.

Secondly, among all the infrastructure-level algorithms and regardless the broker-level policy used, Figs. 4 and 5 show that our proposed PSO and ACO performed well compared to its competitors, being PSO the one achieving the best performance. Each algorithm sends a different number of messages to the hosts to query about their availability and allocate the VMs. ACO and particularly PSO make less use of network resources than BE and GA. The number of messages to send by PSO and ACO depends of the neighborhood size and the maximum number of ant steps, respectively, which is equals to the 60% of a datacenter size.

$$Gain = \frac{\left[\sum_{j=PSEbaseSet*i} \frac{(makespan_j(BE,GA) - makespan_j(PSO,ACO)}{(makespan_j(BE,GA))}\right]}{10} \quad (1)$$

In addition, when PSO and ACO find an idle host, they allocate the current VM and does not make any further move towards the hosts. For each VM allocation, contrarily, BE sends one message to each host in the selected datacenter. Finally, GA is the one producing the greatest makespan and flowtime. Since GA has a population size of 100 and chromosome sizes of 12 –60% of a datacenter size–, to calculate the fitness value, the algorithm sends one message for each host of the chromosome to obtain the chromosome containing the best fitness value. The number of messages sent by GA depends on both the number of host within each chromosome and the population size.

The reason why, regardless of the broker-level policy used, PSO provides the shortest makespan and flowtime, it is because PSO does not repeat the visited hosts in each allocation of a VM. Each particle visits each host in its neighborhood, looking for the host with the lowest load. This increases the chances of PSO of finding an unloaded host, thereby reducing the total number of moves. Moreover, in the ACO algorithm an ant might visit some hosts more than once because ACO uses a random function in the early steps to choose the host to which it performs the movement.

Table 3 shows the average gain of PSO and ACO regarding BE and GA. The average gains have been calculated for all algorithms in combination with the LLTF policy at the broker level, i.e., the policy through which the lowest makespan and flowtime were obtained. The average makespan gains of PSO and ACO are calculated through Equation (1), where *PSEbaseSet* is 25 jobs (PSE-1) or *PSEbaseSet* is 30 jobs (PSE-2), and $i = 40, 80, ..., 400$. Average flowtime gains are calculated similarly. As can be seen, PSO is the algorithm which achieves the best average

Table 3
Using LLTF: Avg. Gains (%) of PSO and ACO w.r.t. to BE and GA

Schedulers	PSE-1		PSE-2	
	Makespan	Flowtime	Makespan	Flowtime
PSO vs BE	10.62	9.32	7.58	6.69
PSO vs GA	46.45	11.24	27.09	9.52
ACO vs BE	7.55	8.86	5.34	6.59
ACO vs GA	44.07	10.79	25.54	9.21

Table 4
VM-level priority-based policy and LLTF at the broker level

Scheduler	Plane strain plate (PSE-1)	
	Makespan (mins.)	Flowtime (mins.)
PSO	4,383.8	10,587,351.5
PSO (priority)	3,678.2	9,148,671.1
Gain (0-100%)	16.1	13.5
ACO	4,578.5	11,053,206.6
ACO (priority)	3,852.3	9,600,038.8
Gain (0-100%)	15.8	13.1
BE	4,810.4	12,053,184.3
BE (priority)	4,081.5	10,604,335.1
Gain (0-100%)	15.1	12.9
GA	7,423.1	12,175,173.9
GA (priority)	6,360.2	10,750,393.5
Gain (0-100%)	14.3	12.7
Scheduler	Cruciform column (PSE-2)	
	Makespan (mins.)	Flowtime (mins.)
PSO	31,425.1	178,976,228.8
PSO (priority)	30,973.2	171,488,815.9
Gain (0-100%)	1.4	4.2
ACO	32,568.6	184,956,815.2
ACO (priority)	32,107.1	177,668,477.9
Gain (0-100%)	1.4	3.9
BE	34,342.1	203,054,005.5
BE (priority)	33,875.5	196,188,548.2
Gain (0-100%)	1.3	3.4
GA	36,674.5	206,369,403.4
GA (priority)	36,172.7	199,477,078.2
Gain (0-100%)	1.3	3.3

gains with respect to BE and GA for both PSEs: the average gains with respect to BE for the (makespan, flowtime) are (10.62%, 9.32%) and (7.58%, 6.69%) for the PSE-1 and PSE-2, respectively. The average gains of PSO for the (makespan, flowtime) with respect to GA are (46.45%, 11.24%) and (27.09%, 9.52%) for PSE-1 and PSE-2, respectively. From Table 3, it can also be noted that the average gains of ACO with respect to BE and GA are lower than those obtained by PSO, for both PSEs. Besides, all the average gains obtained by PSO and ACO for the PSE-1 are greater than the PSE-2 for both metrics and two policy-aware scheduling alternatives. This is since jobs of PSE-2 are more CPU-intensive than that of PSE-1, and then the latencies produced upon creating the virtual infrastructure (which is composed of equal number of VMs for both PSEs) affect less the makespan and the flowtime.

Thirdly, the competing schedulers were also complemented with the VM-level policy for handling jobs within VMs. As shown in Table 4, regardless the VM allocation policy used or the executed PSE, considering job priority information yielded important gains with respect to the accumulated makespan and flowtime: $acumMk = \sum_{j=PSEbaseSet*i} makespan_j$.

The makespan and flowtime from the execution of $25 * i$ jobs (PSE-1) or $30 * i$ jobs (PSE-2), with $i = 40, 80, ..., 400$ of various priorities. The gains for each scheduler were calculated via $\frac{(a_j - b_j)}{a_j} * 100$, where $j = PSEbaseSet * i$, $a = acumMk(withoutPriority)$ and $b = acumMk(withPriority)$.

Once again, the gains have been calculated for all algorithms in combination with the LLTF policy at the broker level, i.e., the policy through which the best results were obtained. For example, for the PSE-1, the obtained gains compared to not considering priorities were in the range of 14.3-16.1% and 12.7-13.5% for the makespan and flowtime, respectively. For PSE-2, the obtained makespan and flowtime gains were in the range of 1.3-1.4% and 3.3-4.2%, respectively. This demonstrates that considering job priorities at the VM level is beneficial. Moreover, for both PSEs the greatest gains for both metrics were obtained by PSO. However, a remark from Table 4 is that the obtained gains by PSE-2 are considerably lower than the obtained by PSE-1. When we executed the two PSEs by varying for each case study the η and α parameters, the jobs execution times –or lengths– comprising each PSE had different variability among them. For example, the average execution time of jobs comprising PSE-1 is equal to 235.12 seconds, and the standard deviation is 46.32 seconds. This means that the PSE-1 jobs have a deviation of 19.70% with respect to the average execution time. On the other hand, the average execution time of jobs from PSE-2 is equals to 2,275.76 seconds, and the standard deviation is 95.08 seconds, or 4.17% of the average execution time. Hence, this characteristic among the jobs lengths involves different effects when using the priority-based policy, i.e., the greater variability among the jobs lengths, the greater gains are obtained (e.g., for PSE-1).

4.3.2. Effects of number of cloud machines

We varied the number of hosts while keeping the number of jobs to execute fixed with the aim of assessing the *horizontal scalability* of the algorithms. For simplicity, we focus on the best performing combination from the previous round of experiments: LLTF at the broker level, combined with the four scheduling alternatives at the infrastructure level, plus the priority-based policy for mapping jobs. The number of hosts of each datacenter was $10 * i$ hosts with $i = 1, 2, ..., 10$ and the number of VMs in each case is increased accordingly as $100 * i$ VMs with $i = 1, 2, ..., 10$. The specific parameter value of each

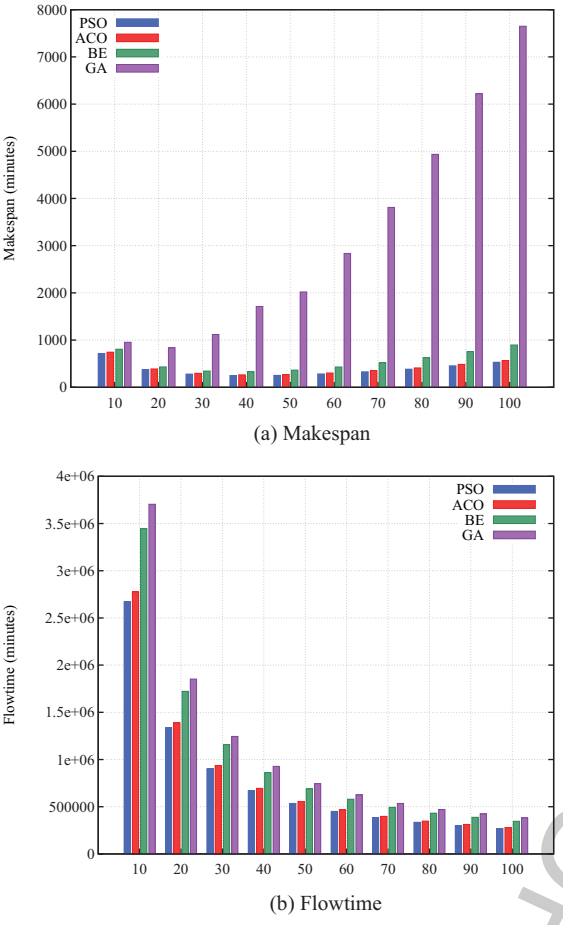


Fig. 6. PSE-1: Results as the number of hosts increases.

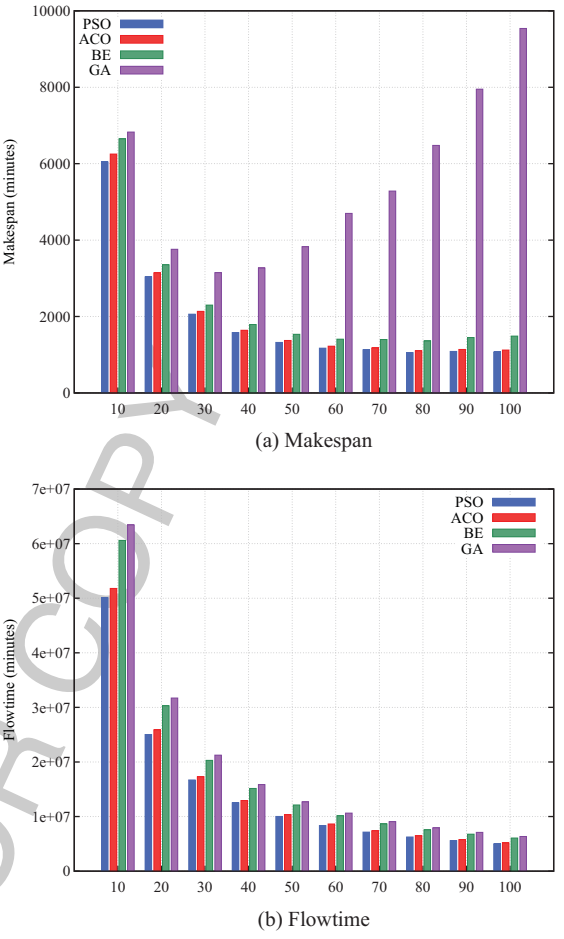


Fig. 7. PSE-2: Results as the number of hosts increases.

algorithm also increases properly to explore the 60% of each datacenter as the Cloud size increases. The number of jobs were set to 10,000 for PSE-1 and 12,000 for PSE-2.

The goal of this experiment is not to study the number of hosts to which the makespan and flowtime curves converge, which is in fact not generalizable, but to quantify the influence of the network latencies in the completion time.

Figures 6 and 7 show the results. With respect to makespan, subfigures 6a and 7a show that the network latencies have more influence in the completion time of PSE-1 than PSE-2. For example, for PSE-1, the makespan of PSO, ACO and BE decreases when the number of hosts is increased from 10 to 40, and the makespan of GA decreases only up to 20 hosts. These decreases in the makespan lead to gains of 65.36%, 64.61%, 59.08% and 12.09% for PSO, ACO, BE and GA, respectively. For PSE-2, the makespan of PSO, ACO and BE decreases until each datacenter reaches 80 hosts, and the makespan of GA decreases until the number of hosts is increased from 10 to 30 hosts. Gains in the makespan for PSO, ACO, BE and GA are 82.55%, 82.35%, 79.54% and 53.88%, respectively. As can be seen, the obtained gains for the PSE-2 are greater than the gains for the PSE-1. This is because the jobs included in PSE-2 are much more CPU-intensive than the jobs in PSE-1, and therefore, the latencies have comparatively less influence on the makespan when the size of the Cloud is increased.

When the number of hosts of each datacenter is greater than 40 (PSO, ACO and BE), and greater than 20 (GA), the makespan is much higher (subfigure 6a). The same applies to PSE-2 in subfigure 7a, where the makespan decreases when the number of hosts is greater than 80 (PSO, ACO, BE and BE), and greater than 30 (GA). This is since the greater the size of each datacenter, the greater the number of network messages sent by the algorithms to manage

VM allocations. With respect to flowtime, irrespective of the PSE and number of hosts, the flowtime in all cases (subfigures 6b and 7b). The greater the number of hosts, the better the distribution of jobs among these hosts.

5. Conclusions

PSEs involve running many CPU-intensive jobs, which must be scheduled in environments such as federated Clouds. However, job scheduling is NP-complete. Particularly, SI-based schedulers have received increasing attention in the Cloud research community [22], but no effort minimizing both the makespan and flowtime in *federated Clouds* exists.

We proposed a three level Cloud scheduler based on SI for executing CM applications on federated Clouds. We studied at the broker level three policies that consider network information for selecting datacenters. At the infrastructure level, we studied two SI strategies for the allocation of VMs to hosts in a selected datacenter. Finally, at the VM level, we consider a priority-based job allocation policy. We have shown how the scheduling decisions at each level affects the overall performance. Our PSO and ACO schedulers perform better than BE and GA. Particularly, when PSO, ACO, BE and GA are combined with LLTF, both the makespan and flowtime are minimized w.r.t. FLTF and LTIR.

We will further improve the broker level. Currently, datacenters are selected through simple policies, but we will explore SI-based selection techniques. The idea is not to scope the use of SI to intra-datacenter scheduling, but to design algorithms operating at the wider (broker) level. We will also extend our scheduler to consider other optimization criteria (e.g., monetary cost). In Clouds, different providers might offer VMs with different capacities/pricing. Thus, trade-off situations between makespan/flowtime and monetary costs arise. Lastly, due to multi-tenancy, it is necessary to allocate resources to a number of *independent* users' VMs/jobs. Hence, we will instantiate the levels of our scheduler with proper policies and SI techniques.

Acknowledgments

We acknowledge the funding by ANPCyT (PICT-2012-0045, PICT-2014-1430) and UNCuyo (06/B308).

References

[1] L. Agostinho, G. Feliciano, L. Olivi, E. Cardozo and E. Guimaraes, A Bio-inspired approach to provisioning of virtual resources in federated Clouds, *In 9th DASC*, IEEE, 2011, pp. 598–604.

[2] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose and R. Buyya, Cloudsim: A toolkit for modeling and simulation of Cloud Computing environments and evaluation of resource provisioning algorithms, *Software: Practice & Experience* **41**(1) (2011), 23–50.

[3] C. Careglio, A. Mirasso and C. García Garino, Estudio numérico de una columna crucifome en grandes deformaciones. *Mecánica Computacional*, XXVI:129-143, 2007.

[4] R. Coutinho, L. Drummond, Y. Frota and D. de Oliveira, Optimizing virtual machine allocation for parallel scientific workflows in federated clouds, *Future Generation Computer Systems* **46** (2015), 51–68.

[5] G. de Oliveira, E. Ribeiro, D. Ferreira, A. Araújo, M. Holanda and M. Walter, ACOsched: A scheduling algorithm in a federated Cloud infrastructure for bioinformatics applications, *In 2013 BIBM*, IEEE, 2013, pp. 8–14.

[6] B. Deevena Raju, P. Pandarinath and G. Prasad, An image reconstruction technique based on ipso-dwt under varying crack, *Journal of Intelligent & Fuzzy Systems* **29**(4) (2015), 1643–1652.

[7] E. Feller, L. Rilling and C. Morin, Energy-Aware Ant Colony based workload placement in Clouds, *In 12th GCA*, IEEE, 2011, pp. 26–33.

[8] Y. Gao, H. Guan, Z. Qi, Y. Hou and L. Liu, A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, *Journal of Computer and System Sciences* **79**(8) (2013), 1230–1242.

[9] C. García Garino, F. Gabaldón and J.M. Goicolea, Finite element simulation of the simple tension test in metals, *Finite Elements in Analysis and Design* **42**(13) (2006), 1187–1197.

[10] C. García Garino, M. Ribero Vairo, S. Andía Fagés, A. Mirasso and J.-P. Ponthot, Numerical simulation of finite strain viscoplastic problems, *Journal of Computational and Applied Mathematics* **246** (2013), 174–184.

[11] S. Ghanbari and M. Othman, A priority based job scheduling algorithm in Cloud Computing, *Procedia Engineering* **50** (2012), 778–785.

[12] R. Jeyarani, N. Nagaveni and R. Vasanth Ram, Design and implementation of adaptive power-aware virtual machine provisioner (APA-VMP) using swarm intelligence, *Future Generation Computer Systems* **28**(5) (2012), 811–821.

[13] J. Jung, S. Jung, T. Kim and T. Chung, A study on the Cloud simulation with a network topology generator, *World Academy of Science, Engineering & Technology* **6**(11) (2012), 303–306.

[14] J. Kennedy, Swarm Intelligence, *In Handbook of Nature-Inspired and Innovative Computing*, 2006, pp. 187–219.

[15] Y. Kessaci, N. Melab and E.-G. Talbi, A pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation, *Cluster Computing* **16**(3) (2013), 451–468.

[16] J. Lucas-Simarro, R. Moreno-Vozmediano, R. Montero and I. Llorente, Scheduling strategies for optimal service deployment across multiple clouds, *Future Generation Computer Systems* **29**(6) (2013), 1431–1441.

[17] E. Mahdiyeh, S. Hussain, K. Mohammad and M. Azah, A survey of the state of the art in Particle Swarm Optimization, *Research Journal of Applied Sciences, Engineering and Technology* **4**(9) (2012), 1181–1197.

[18] N. Makris, Plastic torsional buckling of cruciform compression members, *Journal of Engineering Mechanics* **129**(6) (2003), 689–696.
[19] S. Malik, F. Huet and D. Caromel, Latency based group discovery algorithm for network aware Cloud scheduling, *Future Generation Computer Systems* **31** (2014), 28–39.
[20] C. Mateos, E. Pacini and C. García, Garino, An ACO-inspired algorithm for minimizing weighted flowtime in Cloud-based parameter sweep experiments, *Advances in Engineering Software* **56** (2013), 38–50.
[21] E. Pacini, C. Mateos and C. García Garino, Dynamic scheduling of scientific experiments on Clouds using Ant Colony Optimization, *In 3rd PARENG*, 2013.
[22] E. Pacini, C. Mateos and C. García Garino, Distributed job scheduling based on Swarm Intelligence: A survey, *Computers & Electrical Engineering* **40**(1) (2014), 252–269.
[23] E. Pacini, C. Mateos and C. García Garino, Multi-objective Swarm Intelligence schedulers for online scientific Clouds, *Computing* (2014). In Press.
[24] E. Pacini, C. Mateos and C. García Garino, SI-based Scheduling of Parameter Sweep Experiments on Federated Clouds, *In High Performance Computing*, volume 845 of CCIS, 2014, pp. 28–42.
[25] E. Pacini, C. Mateos and C. García, Garino, Balancing throughput and response time in online scientific clouds via ant colony optimization, *Advances in Engineering Software* **84** (2015), 31–47.
[26] A. Sedeño Noda and A. Raith, A dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem, *Computers & Operations Research* **57** (2015), 83–94.
[27] T. Somasundaram and K. Govindarajan, CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science Cloud, *Future Generation Computer Systems* **34** (2014), 47–65.
[28] R. Tavares Neto and M. Godinho Filho, Literature review regarding Ant Colony Optimization applied to scheduling problems: Guidelines for implementation and directions for future research, *Engineering Applications of Artificial Intelligence* **26**(1) (2013), 150–161.
[29] A. Tchana, G. Son, L. Broto, N. DePalma and D. Hagimont, Two levels autonomic resource management in virtualized IaaS, *Future Generation Computer Systems* **29**(6) (2013), 1319–1332.
[30] J. Tordsson, R. Montero, R. Moreno Vozmediano and I. Llorente, Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers, *Future Generation Computer Systems* **28**(2) (2012), 358–367.
[31] Z. Zhan, X. Liu, Y. Gong, J. Zhang, H. Chung and Y. Li, Cloud computing resource scheduling and a survey of its evolutionary approaches, *ACM Computing Surveys* **47**(4) (2015), 1–63.