SI-based Scheduling of Parameter Sweep Experiments on Federated Clouds

Elina Pacini[1,3], Cristian Mateos[2,3], and Carlos García Garino[1]

[1] ITIC - UNCuyo University. Mendoza, Mendoza, Argentina.
{epacini,cgarcia}@itu.uncu.edu.ar
[2] ISISTAN - UNICEN University. Tandil, Buenos Aires, Argentina.
cmateos@conicet.gov.ar
[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET).

Abstract. Scientists and engineers often require huge amounts of computing power to execute their experiments. This work focuses on the federated Cloud model, where custom virtual machines (VM) are launched in appropriate hosts belonging to different providers to execute scientific experiments and minimize response time. Here, scheduling is performed at three levels. First, at the *broker level*, datacenters are selected by their network latencies via three policies – Lowest-Latency-Time-First, First-Latency-Time-First, and Latency-Time-In-Round–. Second, at the *infrastructure level*, two Cloud VM schedulers based on Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) for mapping VMs to appropriate datacenter hosts are implemented. Finally, at the *VM level*, jobs are assigned for execution into the preallocated VMs. Simulated experiments show that the combination of policies at the broker level with ACO and PSO succeed in reducing the response time compared to using the broker level policies combined with Genetic Algorithms.

1 Introduction

Scientific computing is a field that applies Computer Science to solve typical scientific problems. A representative example of scientific experiments is parameter sweep experiments (PSEs) [13]. Running PSEs involves managing many independent jobs, since the experiments are executed under multiple initial configurations a large number of times, to locate a particular point in the parameter space that satisfies certain user criteria. Indeed, users relying on PSEs need a computing environment that delivers large amounts of computational power over a long period of time. A kind of parallel environment that has gained momentum is represented by Clouds [14].

Executing PSEs on Clouds is not free from the well-known scheduling problem, i.e., it is necessary to develop efficient scheduling strategies to appropriately allocate the jobs and reduce the associated computation time. Moreover, in federated Clouds [3] it is necessary to properly manage physical resources, when they are part of geographically distributed datacenters. Therefore, for the efficient execution of jobs in federated Clouds, scheduling should be performed at three levels. Firstly, at the broker level, scheduling strategies are used for selecting datacenters taking into account issues such as network interconnections or monetary cost of allocating VMs on hosts that compose

them. Secondly, at the infrastructure level, by using a VM scheduler, the VMs are allocated on the available hosts belonging to the previously selected datacenters. Lastly, at the VM level, by using job scheduling techniques, jobs are assigned for execution into allocated virtual resources. However, scheduling is in general an NP-Complete [21] problem and therefore it is not trivial from an algorithmic standpoint. Besides, in this context, the necessity of scheduling algorithms spans the three levels.

In the last ten years, Swarm Intelligence (SI) has received increasing attention among researchers. SI refers to the collective behavior that emerges from a swarm of social insects [9]. Inspired by these capabilities, researchers have proposed algorithms or theories for combinatorial optimization problems, where the most popular SI-based strategies are Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). Moreover, job scheduling in Clouds is also a combinatorial optimization problem, and schedulers in this line that exploit SI have been proposed.

Existing efforts which address SI have not being studied in the context of federated Clouds. In this paper, unlike previous works of our own [16,18] where we proposed a two-level scheduler for Clouds composed of a single datacenter, in this work we extend the scheduler for operating in federated Clouds. To this end, the scheduler operates at three levels. Firstly, by means of a policy that operates at the broker level, datacenters are selected according to their network interconnections and latencies. Indeed, the network latencies among datacenters can contribute to negatively affect the response time delivered to the user. 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 have explored ACO and PSO for allocating the VMs into the physical resources of a datacenter. To 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 actually messages sent to hosts over the network to obtain information regarding their availability. The number of queries to be performed by each algorithm and the latencies of datacenters also influence the response time to the user. Finally, at the VM level, PSE-jobs are assigned to the preallocated VMs by using FIFO, as in [18]. Briefly, in this paper we include the broker level and evaluate how decisions taken both at the broker level and infrastructure level influence the response time.

Simulated experiments performed with job data extracted from a real-world PSE [6] involving a viscoplastic problem suggest that the SI schedulers at the infrastructure level, in combination with these policies at the broker level and FIFO at the VM level, deliver competitive performance with respect to the response time. Experiments were performed by using the CloudSim [2] simulator. To set the basis for comparison, and since VM scheduling is highly challenging and heavily contributes to the overall performance in Cloud scheduling [20], we used the same three policies at the broker level and FIFO at the VM level in combination with a scheduler based on Genetic Algorithms (GA) [1].

The rest of the paper is as follows. Section 2 gives some background necessary to understand the concepts underpinning our scheduler. Section 3 surveys relevant related works. Section 4 presents our proposal. Section 5 presents the experimental evaluation. Section 6 concludes the paper and discusses future prospective extensions.

2 Background

Clouds [14] are the current emerging trend in delivering IT services, and offer to end-users a variety of services covering the entire computing stack. Scientists in general and PSEs users in particular can completely customize their execution environment, thus deploying the most appropriate setup for their experiments. Another related important feature is the ability to scale up and down the computing infrastructure according to PSEs resource requirements.

In the next subsections we describe the federated Cloud basics (Subsection 2.1), and introduce the classical SI-based algorithms (Subsection 2.2), the core optimization techniques of the schedulers implemented in this work at the infrastructure level.

2.1 Federated Clouds

Federated Clouds [15] consist of infrastructures with physical resources belonging to different Cloud providers. A federated Cloud could involve different architectures and levels of coupling among federated datacenters. Federated Clouds also make use of *brokers* to meet the needs of their participating organizations. A broker is an entity which keeps a queue of requests from a particular user that need to be provisioned by a datacenter. In the context of this work, where a user runs PSEs, only one broker is associated with that user.

Clouds allow the dynamic scaling of users applications by the provisioning of computing resources via *machine images*, or VMs. In order to achieve good performance, VMs have to fully utilize its services and resources by adapting to the Cloud dynamically. Proper allocation of resources must be guaranteed so as to improve resource usefulness [15].

For running applications in a Cloud, resources are scheduled at three levels (Figure 1): Broker level, Infrastructure level, and VM level. At the broker level, different policies can be implemented in order to serve users. Some examples are policies considering the influence of network interconnections among Cloud datacenters or monetary cost of hosts that compose them [1]. Furthermore, the scheduler at this level can decide to deploy the VMs in a remote Cloud when there are insufficient physical resources in the datacenter where the VM creation was issued. Secondly, once a datacenter/provider has been selected by a broker, at the infrastructure level, the VMs are allocated into real hardware through a VM scheduler. Finally, at the VM level, by using job scheduling techniques, jobs are assigned for execution into virtual resources (the allocated VMs). Figure 1 illustrates a Cloud where one or more users are connected via a network and require the creation of a number of VMs for executing their experiments, i.e., a set of jobs. As can be seen in the Figure 1, a broker is created for each user that connects to the Cloud. Each broker knows who are the providers that are part of the federation through network interconnections –the relation of each broker is colored with green and blue dotted lines–. In addition, the Figure 1 illustrates how jobs sent by *User N* are executed in the datacenter of *Cloud Provider 2*. At the right of this provider –inside the dotted Cloud– the intra-datacenter scheduling activities are depicted, i.e., at the infrastructure level and the VM level.

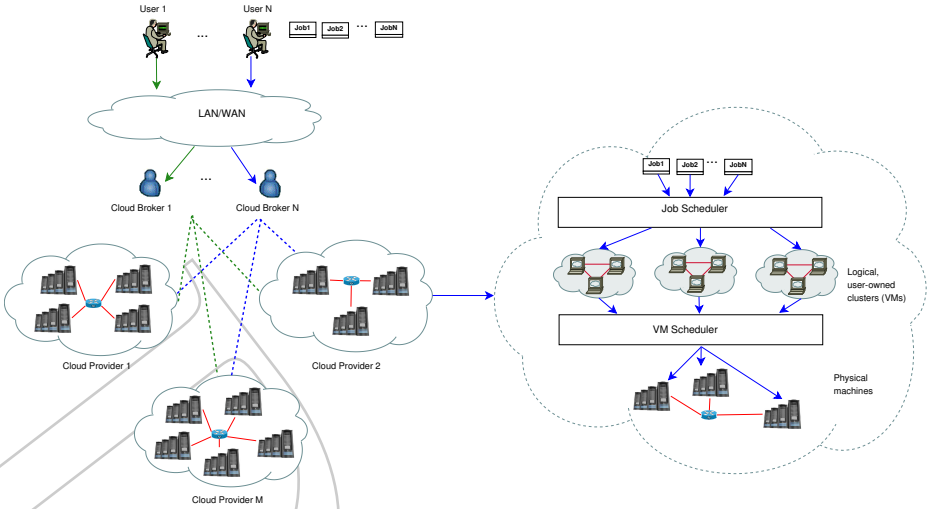

Fig. 1: High-level view of a federated Cloud.

2.2 SI techniques for Cloud scheduling

Broadly, SI techniques [9] have shown to be useful in optimization problems. The advantage of these techniques derives from their ability to explore solutions in large search spaces in a very efficient way. The most popular SI-based techniques are ACO and PSO. ACO [9] arises from the way real ants behave in nature, i.e., from the observation of ant colonies when they search the shortest paths to reach a food source from their nest. In nature, real ants move randomly from one place to another to search for food, and upon finding food and returning to their nest each ant leaves a *pheromone* that lures other working ants to the same course. When more and more ants choose the same path, the pheromone trail is reinforced and even more ants will further choose it. Over time the shortest paths will be intensified by the pheromone faster since the ants will both reach the food source and travel back to their nest at a faster rate.

On the other hand, PSO [9] is a population-based technique that finds solution to a problem in a search space by modeling and predicting insect social behavior in the presence of objectives. In the algorithm the general term "particle" is used to represent birds, bees or any other individuals who exhibit social behavior as group and interact with each other. An example based on nature to illustrate the algorithm is as follows: a group of bees flies over the countryside looking for flowers. Their goal is to find as many flowers as possible. At the beginning, bees do not have knowledge of the field and fly to random locations with random velocities looking for flowers. Each bee has the capability of remember the places where it saw the most flowers, and moreover, somehow knows the places where other bees have found a high density of flowers. These two pieces of information –*nostalgia* and *social knowledge*– are used by the bees to continually modify their trajectory, i.e., each bee alters its path between the two directions to fly somewhere between the two points and find a greater density of flowers. Occasionally, a bee may fly over a place with more flowers than any other place found

previously by other bees in the swarm. If this happens the whole swarm is attracted towards this new direction.

3 Related work

In this paper we address the scheduling of scientific application in federated Clouds in order to minimize the response time considering the influence of the network latencies among datacenters. Our approach differs from those presented in literature for federated Cloud, where the authors have not considered SI-based strategies at the infrastructure level. In previous works of our own [16,18] we have presented SI-based schedulers focused on the infrastructure level. However, it is important to note that in these works the schedulers operate at two levels for Clouds composed of a single datacenter. The remaining works found in literature are focused on one level and do not evaluate the three levels such as we propose in this work.

Among these works we can mention [5,10,20]. In [5] the authors summarize some VM allocation policies based on linear programming for different Cloud federation architectures. Then, in [10] scheduling strategies at the broker level based on different optimization criteria (e.g., monetary cost optimization or performance optimization) and different user constraints (e.g., budget, performance, VMs types) were proposed. Moreover, in [20], the scheduler restricts the deployment of VMs according to some placement constraints (e.g., Clouds to deploy the VMs) defined by the user.

Two works that deserve special attention are [1,4]. In [1], the authors used at the broker level, a Dijkstra algorithm to select the datacenter with lower monetary cost, and a GA for allocating the VMs at the infrastructure level. Although in this work the authors target the broker and the infrastructure levels, the goal was to reduce the monetary costs without considering the response time. For scientific applications in general, the response time is very important [18]. Moreover, in [4] the authors proposed an ACO scheduler based on load balancing to perform efficient distribution of jobs by finding the best VM to execute jobs. The aim of this work was minimizing the makespan and improve load balancing in the VMs. Makespan is the maximum execution time of a set of jobs. To the best of our knowledge, this is the only work in literature in which the authors have considered the use of SI for federated Clouds. However, it is important to note that ACO was implemented at the VM level and not at the infrastructure level.

With respect to works which address the scheduling problem at the infrastructure level –intra-datacenter– using SI-based strategies as we propose in this work, few efforts have been found [17]. However, in these related works, it is important to note that SI techniques are used to solve the job scheduling problem, i.e., determining how the jobs are assigned to pre-allocated VMs, and few efforts have aimed to solve VM scheduling problems to date [17]. It is worth noting that, from the related works found, most of them have been proposed for Clouds taking into account only one of the scheduling levels without considering SI for allocating VMs, or Clouds composed by a single datacenter where only scheduling of jobs (and not VMs) is addressed. The next Section explains our approach, which considers the three levels described in Subsection 2.1.

4 Proposed Scheduler

The goal of our scheduler is to minimize the response time of a set of PSE jobs. Response time is the period of time between a user makes a request to the Cloud and gets the answer, i.e., the period of time in which a user requests a number of VMs to execute its PSE, and the time in which all the entire PSE-jobs finish their execution. Conceptually, the scheduling problem to tackle down can be formulated as follows. A PSE is formally defined as a set of $N = 1, 2, ..., n$ 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 issued by the user. With the goal of minimizing the response time, the need to implement strategies to select the appropriate datacenters in which to place the VMs arises. For example, the most suitable datacenter might be the one that 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 geographically distributed Cloud datacenters. One way to mitigate the effects of such latencies is to choose a datacenter which operates with a fast and efficient internal network and plenty of capacity.

The proposed scheduler proceeds as follows. Firstly, at the broker level, a datacenter is selected by a policy that takes into account network interconnections and/or network latencies. Secondly, at the infrastructure level, by means of a VM scheduler, 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, a policy for assigning user jobs to allocated VMs is also used (currently we use FIFO).

4.1 Scheduler at the Broker level

The scheduler at the broker level is executed both to select the first datacenter to allocate the VMs, which are managed by the scheduler implemented at the infrastructure level, as well as each time such datacenter is not able to perform the allocation of VMs anymore. At present, the policies implemented at this level are:

- Lowest-Latency-Time-First (LLTF), maintains a list of all network interconnected datacenters sorted by their latencies. Each time a user requires a number the VMs to execute their PSE, this policy is responsible for selecting first the datacenter with the lowest latency in the list. Then, whenever a datacenter has no more physical resources to allocate VMs, then the algorithm selects the next datacenter in the list.
- First-Latency-Time-First (FLTF) selects the first datacenter from a list sorted randomly, containing all network interconnected datacenters to which a user can access and allocate his VMs. When the selected datacenter has no more available physical resources to allocate VMs, the algorithm selects the next datacenter in the list.
- Latency-Time-In-Round (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.

4.2 Scheduler at the Infrastructure level

To implement the Infrastructure level policy, the SI algorithms proposed in [18] are used. Below we describe these algorithms.

Scheduler based on Ant Colony Optimization 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 created in a datacenter, an ant is initialized and starts to work. A master table containing information on the load of each host in the selected datacenter is initialized. Subsequently, if an ant associated to the VM that is executing the algorithm already exists, the ant is obtained from a pool of ants. If the VM does not exist in the ant pool, then a new ant is created. To do this, first, a list of all suitable hosts belonging to the selected datacenter in which can be allocated the VM is obtained.

Algorithm 1 ACO-based Cloud scheduler: Core logic

```
Procedure AntAlgorithm()
Begin
  step=1
  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
    step=step+1
    moveTo(nextHost)
  end while
  deliverVMtoHost()
End
```

Then, the working ant and its associated VM is added to the ant pool and the ACO-specific logic starts to operate (see Algorithm 1). In each iteration, 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 of the datacenter, 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.

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. When an ant reads the information from a load table in a host, the ant chooses 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 provided in the load information table, the ant chooses the host with the smallest load. 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. Since each step performed by an ant involves moving through the intra-datacenter 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 allocated VMs yet, the ant allocates its associated VM to it directly without performing further steps. Every time an ant sends a message through the intra-datacenter network to obtain information regarding the availability of the hosts from the selected datacenter latencies are produced. The smaller the number messages sent to the hosts through the network, the smaller the impact of the latencies in the response time given to the user.

Every time an ant visits a host, it updates the host load information table with the information of other hosts in the datacenter, 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 are the hosts that ants have visited on their way to deliver their VMs together with their load information.

Scheduler based on Particle Swarm Optimization In this algorithm, each particle works independently and represents a VM looking for the best host –in the selected datacenter at the broker level– to which it can be allocated. Following the analogy from the example of bees in Subsection 2.2, each VM is considered a bee and each host represent locations in the field with different density of flowers. When a VM is created, a particle is initialized in a random host, i.e., in a random place in the field. The density of flowers of each host is determined by its load.

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. This means that the host has more available resources to allocate a VM. In the algorithm (see Algorithm 2), every time a user requires a VM, a particle is initialized in a random host of the selected datacenter (`getInitialHost()`). Each particle in the search space takes a position according to the load of the host in which is initialized through the `calculateTotalLoad(hostId)` method. Load refers to the total CPU utilization within a host and is calculated as well as ACO. The neighborhood of each particle is composed by the remaining hosts in a datacenter excluding the one in which the particle is initialized. The neighborhood of that particle is obtained through `getNeighbors(hostId,neighborSize)`. Each one of the neighbors –hosts– that compose the neighborhood are selected randomly. Moreover, the size of the particle neighborhood is a parameter defined by the user.

In each iteration of the algorithm, the particle moves to the neighbors of its current host in search of a host with a lower load. The velocity of each particle 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 has a lower load than the original host, then the particle is moved to the neighbor host with a greater velocity. Taking into account that the 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 and finishes its task.

Algorithm 2 PSO-based Cloud scheduler: Core logic

```
Procedure PSOallocationPolicy(vm,hostList)
Begin
  particle = new Particle(vm, hostList)
  initialHostId = particle.getInitialHost()
  currentPositionLoad = particle.calculateTotalLoad(initialHostId)
  neighbours = particle.getNeighbours(initialHostId, neighbourSize)
  While (i < neighbours.size()) do
       neighbourId = neighbours.get(i)
    destPositionLoad = particle.calculateTotalLoad(neighbourId)
      if(destPositionLoad == 0)
        currentPositionLoad = destPositionLoad
        destHostId = neighbours.get(i)
        i=neighbours.size()
      end if
      if(currentPositionLoad - destPositionLoad > velocity)
        velocity = currentPositionLoad - destPositionLoad
        currentPositionLoad = destPositionLoad
        destHostId = neighbours.get(i)
      end if
  i=i+1
  end while
  allocatedHost=hostList.get(destHostId)
  if(!allocatedHost.allocateVM(vm)
       PSOallocationPolicy(vm,hostList)
End
```

Since each move a particle performs involves traveling through the intra-datacenter network, similarly to ACO, 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. The smaller the number messages sent to the hosts through the network by a particle, the smaller the impact of the latency in the response time given to the user.

4.3 Scheduler at the VM level

Once the VMs have been allocated to physical resources at the Infrastructure level, the job scheduler proceeds to assign the jobs to these VMs. 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, and then retrieves jobs by a FIFO policy. Each time a job is obtained from the list, it is submitted to be executed in a VM in a round robin fashion. 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. 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. This is, a time-shared CPU scheduling policy was used, since it is a good alternative for executing CPU-intensive jobs in terms of fairness.

5 Evaluation

To assess the effectiveness of our proposal, we processed a real case study for solving a well-known benchmark problem [6]. Details on the experimental methodology are provided in Section 5.1. After that, we compared our proposal with a GA in terms of the metric of interest in this paper, i.e., response time. The results are explained in Subsection 5.2.

5.1 Experimental Methodology

A plane strain plate with a central circular hole, see reference [6] and therein is studied. The dimensions of the plate were 18 x 10 m, with $R = 5$ m. The 3D finite element mesh used had 1,152 elements. To generate the PSE jobs, a material parameter –viscosity η– was selected as the variation parameter. Then, 25 different viscosity values for the η parameter were considered, namely $x.10^y$ Mpa, with $x = 1, 2, 3, 4, 5, 7$ and $y = 4, 5, 6, 7$, plus 1.10^8 Mpa. Introductory details on viscoplastic theory and numerical implementation can be found in [6].

After establishing the problem parameters, we employed a single 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 tests were solved using the SOGDE finite element solver software [7]. 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 to be executed by or associated to a 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 the job i on the real machine. For example, for a job taking 539 seconds to execute, the approximated number of instructions was 2,160,657 MI (Million Instructions). By means of the generated job data, we instantiated the CloudSim toolkit [2].

The experimental scenario consists of a Cloud composed of 5 datacenters. The network topology is defined in the Boston university Representative Internet Topology

gEnerator (BRITE) [8] format. BRITE is a file used by CloudSim which defines the different nodes that compose a commonly-found federation (e.g., datacenters, brokers) and the network connections among them. This file is then used to calculate latencies in network traffic. Then, each datacenter is composed of 10 physical resources –or "host" in CloudSim terminology–. The characteristics of hosts are 4,008 MIPS (processing power), 4 GBytes (RAM), 400 GBytes (storage), 100 Mbps (bandwidth), and 4 CPUs. Furthermore, each datacenter has an associated latency of 0.8, 1.5, 0.5, 0.15, 2.8 seconds, respectively. These latencies have been assigned taking into account other works proposed in the literature [12,19].

Moreover, a user requests 100 VMs to execute its PSE. Each VM has one virtual CPU of 4,008 MIPS, 512 Mbyte of RAM, a machine image size of 100 Gbytes and a bandwidth of 25 Mbps. For further details about the job data gathering and the CloudSim instantiation process, please refer to [13,18].

In this work, we evaluated the performance of the user PSE-jobs as we increased the number of jobs to be performed from 1,000 to 10,000. This is, the base job set comprising 25 jobs that was obtained by varying the value of η was cloned to obtain larger sets. Each job was determined by a length parameter or the number of instructions to be executed by the job, which varied between 1,362,938 and 2,160,657 MI. Moreover, another parameter was PEs, or the number of processing elements (cores) required to perform each individual job. Each job required one PE since jobs are sequential (not multi-threaded). Finally, the experiments had input files of 291,738 bytes and output files of 5,662,310 bytes.

5.2 Performed experiments

In this subsection we report results obtained through our proposed three level scheduler. Particularly, at the infrastructure level we compare to another alternative scheduler based on GA proposed in [1], which has been previously evaluated via CloudSim as well. The population structure is represented as the set of physical resources that compose a datacenter and each chromosome is an individual in the population that represents a part of the searching space. Each gene (field in a chromosome) is a physical resource in a datacenter, and the last field in this structure is the *fitness* field, which indicate the suitability of the hosts in each chromosome.

In our experiments, the GA-specific parameters were set to the following values: *chromosome size = 8*, *population size = 10* and *number of iterations = 10*. Moreover, we have set the ACO-specific parameters to values within the range of values studied in [11]: *mutation rate = 0.6*, *decay rate = 0.1* and *maximum steps = 8*, and the PSO-specific parameter *neighbourhood size = 8*. Since the number of hosts that compose each datacenter is equal to 10, a specific parameter values (i.e., maxSteps in ACO, neighborhood in PSO and chromosome size in GA) equal to 8, means exploring a percentage of the 80% of the number of hosts for each datacenter.

Figure 2 compares the obtained results for all the considered scheduling algorithms (ACO, PSO, GA) and each one the policies at the broker level (LLTF, FLTF, LTIR) in subfigures a), b) and c), respectively. Graphically, it can be seen that the response time presents a linear tendency in all cases. As shown in the subfigures included in Figure 2,

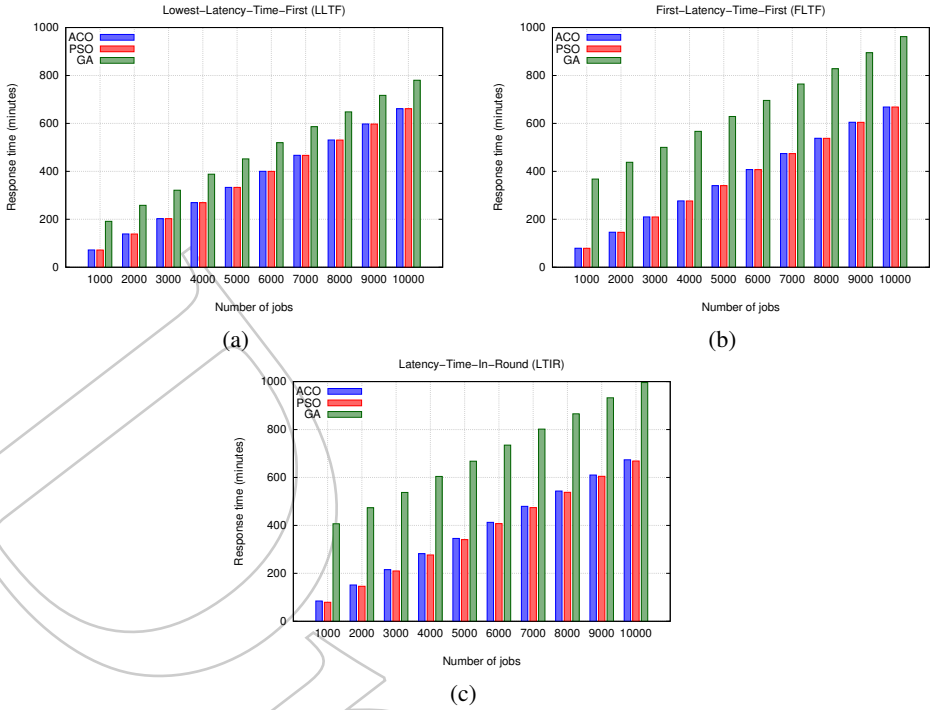

Fig. 2: Response time as the number of jobs increases

regardless of the policy used at the broker level, GA is the algorithm that produces the greatest response time to the user with respect to ACO and PSO.

Since the GA algorithm contains a population size of 10 and chromosome sizes of 8, for each datacenter in which it tries to create the VMs, 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. The number of messages sent is equals to the number of host within each chromosome multiplied by the population size. The number of messages to send through the network for each algorithm directly impacts the response time to the user. This is because for each message sent to query about hosts availability, latencies from datacenters affect the answers.

The proposed ACO and PSO, however, make less use of network resources than GA, being in some cases PSO the one which sends less network messages. The number of messages to send by ACO depends of the maximum number of steps that an ant carries out to allocate its associated VM. For example, when the maximum number of steps is equals to 8, ACO sends a maximum of 8 messages per VM allocation. Moreover, when ACO finds an idle host, it allocates the current VM and does not perform any further step. This reduces the total number of network messages sent. On the other hand, the number of network messages to send by PSO depends of the neighborhood size, which is also equals to 8, i.e., PSO sends a maximum of 8 messages per VM

allocation. Furthermore, like ACO, when PSO finds an idle host, it allocates the current VM and does not make any further move. This also reduces the total number of network messages sent, and therefore, the total latency that influences the user response time.

Another observations from the subfigures included in Figure 2 are that when the LLTF policy is used combined with PSO, ACO and GA, the response time decreases with respect to FLTF and LTIR policies. This happens because most VMs are created in datacenters with lower latencies. For example, when the LLTF policy is used and the number of jobs to be executed is increased from 1,000 to 10,000 (subfigure 2a), the response time varied between (71–661), (72–661), and (191–780) minutes, for PSO, ACO and GA, respectively. On the other hand, when the FLTF policy is used at the broker level (subfigure 2b), the response time for PSO, ACO, and GA, when the number of jobs was increased from 1,000 to 10,000, varied between (79–668), (79–668), and (367–962) minutes. Finally, when the LTIR policy is used, the response time rose from (79– 668), (84–673), and (409–996) minutes, when the number of jobs was increased from 1,000 to 10,000, and for PSO, ACO and GA, respectively.

As can be seen, the response times for ACO and PSO are close when FLTF and LTIR are used at the broker level. The reason is because both algorithms reduce the number of queries to the hosts when LTIR is used. This is because when ACO and PSO find an idle host, they not make any further move, and due to the fact that LTIR explores all datacenters (in a circular order for each VM to be allocated), it has more chance of finding an underloaded hosts where to allocate the VMs. However, if the user requests the execution of a larger number of VMs, the latencies of datacenters will have more influence in the response time when LTIR is used instead of FLTF.

Finally, the gains of PSO and ACO with respect to GA, when LLTF is used at the broker level varied between 15% and 62%. When FLTF was used, gains varied between 30% and 78%. Lastly, when LTIR was used, gains varied between 32% and 80%. As can be seen, the greatest gains were obtained when LTIR was used at the broker level. The is because, since GA sent a greater number of network messages to the hosts than PSO and ACO, the inter-datacenter latencies had more influence on the response time.

6 Conclusions

One popular kind of scientific experiments are PSEs, which involve running many CPU-intensive independent jobs. These jobs must be efficiently processed –i.e., scheduled– in the different computing resources of a distributed environment such as the ones provided by Cloud. The growing popularity of Cloud environments has increased the attention in the research of resource allocation mechanisms across datacenters. Federated Clouds potentially provide plenty of resources to users, specially when the number of VMs required by a user exceeds the maximum that can be provided by a single provider or datacenter. Then, job/VM scheduling plays a fundamental role.

Recently, SI-inspired algorithms have received increasing attention in the Cloud research community for dealing with VM and job scheduling. In this work, we described two schedulers –based on ACO and PSO– for the efficient allocation of VMs in a datacenter combined with three strategies –LLTF, FLTF and LTIR– that consider network information for selecting datacenters. Simulated experiments performed with

CloudSim and real PSE job data suggest that our PSO and ACO schedulers provide better response times to the user than GA. In addition, when PSO, ACO and GA are combined with LLTF, the response time is the lowest for all of them w.r.t. FLTF and LTIR, being LTIR the most influential on the response time.

We are extending this work in several directions. We will explore the ideas exposed in this paper in the context of other bio-inspired techniques such as Artificial Bee Colony (ABC), which is also extensively used to solve combinatorial optimization problems. Another issue which deserves attention is to consider other Cloud scenarios [15] with heterogeneous physical resources belonging to different Cloud providers.

Due to multi-tenancy, in Clouds it is necessary to provide distributed scheduling mechanisms for allocating resources to a number of independent users' VMs/jobs along with time constraints. For this, we plan to implement a Cloud scheduler based on SI techniques in order to *fairly* schedule users' VMs/jobs based on different optimization criteria (e.g., cost, execution times, etc.).

Finally, another interesting issue consists of providing more elaborated dynamic optimization capabilities, enabling the dynamic reallocation (migration) of VMs from one physical machine to another to meet a specific optimization criteria such as improving the response time, reducing the number of physical resources in use for minimizing energy consumption, or balancing the workload of all resources to avoid resources saturation and performance slowdown. In addition, the user could also specify constraints for the scheduler decisions such as hardware (amount of CPU, memory, bandwidth, etc.), platform (type of hypervisor, operating system, etc.), location (geographical restrictions), among others.

Acknowledgments

We acknowledge the financial support provided by ANPCyT through grants PICT-2012-0045 and PICT-2012-2731, and National University of Cuyo project 06B/308. The first author acknowledges her Ph.D. fellowship granted by 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). pp. 598–604. DASC 11, IEEE Computer Socienty, Washington, DC, USA (12-14 December 2011)
2. 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. Software: Practice & Experience 41(1), 23–50 (January 2011)
3. Celesti, A., Fazio, M., Villari, M., Puliafito, A.: Virtual machine provisioning through satellite communications in federated Cloud environments. Future Generation Computer Systems 28(1), 85–93 (2012)
4. de Oliveira, G., Ribeiro, E., Ferreira, D., Araújo, A., Holanda, M., Walter, M.: ACOsched: a scheduling algorithm in a federated Cloud infrastructure for bioinformatics applications. In: International Conference on Bioinformatics and Biomedicine. pp. 8–14. IEEE (2013)

5. Gahlawat, M., Sharma, P.: Survey of virtual machine placement in federated Clouds. In: International Advance Computing Conference (IACC). pp. 735–738. IEEE (2014)
6. García Garino, C., Ribero Vairo, M., Andía Fagés, S., Mirasso, A., Ponthot, J.P.: Numerical simulation of finite strain viscoplastic problems. Journal of Computational and Applied Mathematics 246, 174–184 (Jul 2013)
7. García Garino, C., Gabaldón, F., Goicolea, J.M.: Finite element simulation of the simple tension test in metals. Finite Elements in Analysis and Design 42(13), 1187–1197 (2006)
8. Jung, J., Jung, S., Kim, T., Chung, T.: A study on the Cloud simulation with a network topology generator. In: World Academy of Science, Engineering & Technology. vol. 6, pp. 303–306 (2012)
9. Kennedy, J.: Swarm Intelligence. In: Zomaya, A. (ed.) Handbook of Nature-Inspired and Innovative Computing, pp. 187–219. Springer US (2006)
10. Lucas-Simarro, J., Moreno-Vozmediano, R., Montero, R., Llorente, I.: Scheduling strategies for optimal service deployment across multiple clouds. Future Generation Computer Systems 29(6), 1431 – 1441 (2013), including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems
11. Ludwig, S., Moallem, A.: Swarm Intelligence approaches for Grid load balancing. Journal of Grid Computing 9(3), 279–301 (2011)
12. Malik, S., Huet, F., Caromel, D.: Latency based group discovery algorithm for network aware Cloud scheduling. Future Generation Computer Systems 31, 28 – 39 (2014)
13. Mateos, C., Pacini, E., García Garino, C.: An ACO-inspired algorithm for minimizing weighted flowtime in Cloud-based parameter sweep experiments. Advances in Engineering Software 56, 38–50 (2013)
14. Mauch, V., Kunze, M., Hillenbrand, M.: High performance cloud computing. Future Generation Computer Systems 29(6), 1408 – 1416 (2013), including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems
15. Moreno Vozmediano, R., Montero, R., Llorente, I.: IaaS Cloud architecture: FromvVirtualized datacenters to federated Cloud infrastructures. IEEE Computer 45(12), 65–72 (2012)
16. Pacini, E., Mateos, C., García Garino, C.: Dynamic scheduling of scientific experiments on Clouds using Ant Colony Optimization. In: Topping, B. H. V. and Iványi, P. (ed.) Proceedings of the Third International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering. Civil-Comp Press, Stirlingshire, United Kingdom (2013), paper 33
17. Pacini, E., Mateos, C., García Garino, C.: Distributed job scheduling based on Swarm Intelligence: A survey. Computers & Electrical Engineering 40(1), 252–269 (2014), 40th-year commemorative issue
18. Pacini, E., Mateos, C., García Garino, C.: Multi-objective Swarm Intelligence schedulers for online scientific Clouds. Special Issue on Cloud Computing. Computing pp. 1–28 (2014)
19. Somasundaram, T., Govindarajan, K.: CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science Cloud. Future Generation Computer Systems 34, 47 – 65 (2014)
20. Tordsson, J., Montero, R., Moreno Vozmediano, R., Llorente, I.: Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Generation Computer Systems 28(2), 358 – 367 (2012)
21. Woeginger, G.: Exact algorithms for NP-Hard problems: A survey. In: Junger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization - Eureka, You Shrink!, Lecture Notes in Computer Science, vol. 2570, pp. 185–207. Springer (2003)