HPCLatAm 2013, pp. 1-12 (full paper)
Session: Evolutionary Computation & Scheduling
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

Dynamic Scheduling based on Particle Swarm Optimization for Cloud-based Scientific Experiments

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

[1] ITIC and Facultad de Ingeniería - UNCuyo University. Mendoza, Argentina.
{epacini,cgarcia}@itu.uncu.edu.ar
[2] ISISTAN-CONICET - UNICEN University. Tandil, Buenos Aires, Argentina
cmateos@conicet.gov.ar

Abstract. Parameter Sweep Experiments (PSEs) allow scientists to perform simulations by running the same code with different input data, which results in many CPU-intensive jobs and thus computing environments such as Clouds must be used. Our goal is to study private Clouds to execute scientific experiments coming from multiple users, i.e., our work focuses on the Infrastructure as a Service (IaaS) model where custom Virtual Machines (VM) are launched in appropriate hosts available in a Cloud. Then, correctly scheduling Cloud hosts is very important and it is necessary to develop efficient scheduling strategies to appropriately allocate VMs to physical resources. Here, scheduling is however challenging due to its inherent NP-completeness. We describe and evaluate a Cloud scheduler based on Particle Swarm Optimization (PSO). The main performance metrics to study are the number of Cloud users that the scheduler is able to successfully serve, and the total number of created VMs, in online (non-batch) scheduling scenarios. Besides, the number of intra-Cloud network messages sent are evaluated. Simulated experiments performed using CloudSim and a job data from real scientific problems show that our scheduler succeeds in balancing the studied metrics compared to schedulers based on Random assignment and Genetic Algorithms.

1 Introduction

Cloud Computing [4] suits well in executing scientific experiments, because of its promise of provisioning infinite resources. Within a Cloud, resources can be effectively and dynamically managed using virtualization technologies. Moreover, Cloud providers offer their services according to three fundamental models: infrastructure, platform, and software as services. Although the use of Clouds finds its roots in IT environments, the idea is gradually entering scientific and academic ones [16].

This work is focused on the Infrastructure as a Service (IaaS) model, whereby users request virtual machines (VM) to the Cloud, which are then associated to physical resources. However, in order to achieve the best performance, VMs have to fully utilize the physical resources. To perform this, scheduling the processing units of physical Cloud resources is an important issue and it is necessary to develop efficient scheduling strategies to appropriately allocate the VMs in physical resources. Here, *scheduling* refers to the way VMs are allocated to run on the available computing resources. However, scheduling is an NP-complete problem and therefore it is not trivial from an

algorithmic perspective. In this context, scheduling may also refer to two goals, namely delivering efficient high performance computing (HPC) or supporting high throughput computing (HTC). HPC focuses on decreasing job execution time whereas HTC aims at increasing the processing capacity of the system. As will be shown, our proposed scheduler attempts to balance both aspects.

Swarm Intelligence (SI) [3] has received increasing attention among researchers, and refers to the collective behavior that emerges from social insects swarms to collectively solve complex problems. Hence, researchers have proposed algorithms for combinatorial optimization problems [3]. Moreover, scheduling in Clouds is also a combinatorial optimization problem, and hence schedulers exploiting SI have been proposed.

Unlike our previous work [14], where an ant colony optimization (ACO) was used in batch scheduling scenarios, this paper proposes a PSO scheduler to allocate VMs to physical Cloud resources, and evaluates its performance in an online Cloud (non-batch) scenario as the one in [16], where multiple users connect to the Cloud at different times to execute their PSEs. Experiments have been conduced to evaluate the number of serviced users (which relates to throughput) among all users that are connected to the Cloud, and the total number of VMs that are allocated by the scheduler (which relates to response time). The more the users served, the more the executed PSEs, and hence throughput increases. Moreover, when more VMs can be allocated, more physical resources can be taken advantage of, and hence PSE execution time decreases.

To quantify this trade-off, we use a weighted metric in which the results obtained from different scheduling algorithms have been normalized and weighted to determine, from the evaluated algorithms, which one better balances the aforementioned metrics. For this, two weights have been assigned to the individual metrics, i.e., a weigh for the number of serviced users (*wSU*) and a weight for the number of created VMs (*wVMs*). Then, we use a mixed HTC/HPC scenario by assigning weights (0.5, 0.5) with the aim of balancing the two basic metrics. Besides, we measure network resources consumed. Experiments performed by using the CloudSim toolkit [5], together with job data extracted from a real-world PSE and other scheduling policies for Clouds show that our PSO performs competitively with respect to the used metrics.

Section 2 gives preliminary concepts. Section 3 surveys related works. Section 4 describes the PSO scheduler, while Section 5 evaluates it. Section 6 concludes the paper.

2 Background

2.1 Cloud Computing basics

Clouds [4] refers to a set of technologies that offer computing services through a network. A related important feature is the ability to scale up and down the computing infrastructure according to resource requirements.

Central to any Cloud is the concept of *virtualization*, i.e., the capability of a software system of emulating various operating systems on a single machine. By means of this support, users exploit Clouds by requesting them VMs that emulate any operating system on top of several physical machines, which in turn run a host operating system. Particularly, for scientific applications, the use of virtualization has shown to provide

many useful benefits, including user-customization of system software and services, and the value of Clouds has been already recognized within the scientific community [21].

While Clouds help scientific users to run complex applications, job and VM management is a key concern that must be addressed. Broadly, job scheduling is a mechanism that maps jobs/VMs to appropriate resources to execute, and the delivered efficiency will directly affect the performance of the whole distributed environment. Moreover, Cloud resources need to be scheduled at two levels (Fig. 1): Infrastructure-level and VM-level. At the Infrastructure-level, one or more Cloud infrastructures are created and through a VM scheduler the VMs are allocated into real hardware. Then, at the VM-level, by using job scheduling techniques, jobs are assigned for execution into allocated virtual resources. Fig. 1 illustrates a Cloud where one or more scientific users connect via a network and require the creation of a number of VMs for executing their PSEs.

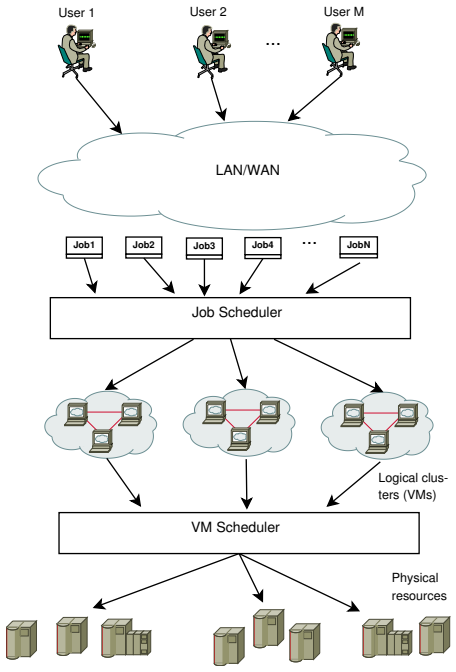

Fig. 1: High-level view of a Cloud

We focus on the Infrastructure-level in order to more efficiently solve the allocation of VMs to physical resources in a dynamic, multi-user Cloud. However, job/VM scheduling is NP-complete, and therefore approximation heuristics are necessary.

2.2 Particle Swarm Optimization (PSO)

PSO [3] is a population-based optimization technique that finds solution to a problem in a search space by modeling and predicting insect social behavior in the presence of objectives. 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.

Under PSO, multiple candidate solutions –called particles– coexist and indirectly collaborate simultaneously. Each particle "flies" in the problem search space looking for the optimal position to land. A particle adjusts its position as time passes according to its own experience as well as according to the experience of neighbor particles. Moreover, particles are essentially described by two characteristics: the particle position, which defines where the particle is located with respect to other solutions in the search space, and the particle velocity, which defines the direction and how fast the particle should move to improve its fitness. The fitness of a particle is a number representing how close a particle is to the optimum point compared to other particles in the search space.

The basic PSO algorithm, which minimizes an objective function $f(x)$ of a variable vector x defined on a n-dimensional space, uses a swarm of m particles. Each parti-

cle i of the swarm is associated with a position in a continuous n-dimensional search space. Similarly, the velocity is also an n-dimensional vector. Denoting with x_i^k and v_i^k respectively the position and velocity of particle i at iteration k of the PSO algorithm, the following equations are used to iteratively modify the velocities of the particles and positions: $v_i^{k+1} = w * v_i^k + c1 * r1 * (pbest_i - x_i^k) + c2 * r2 * (gbest - x_i^k)$, and $x_i^{k+1} = x_i^k + v_i^{k+1}$, where v_i^{k+1} represents the distance to be traveled by the particle from its current position in the k^{th} iteration, x_i^{k+1}represents the particle position in the k^{th} iteration, w is the *inertia* parameter that weights the previous particles velocity, i.e., w controls the impact of previous historical values of particle velocities on its current velocity, *pbest* represents its best personal position (i.e. its experience), and *gbest* represents the best position among all particles in the population. Parameters *c1* and *c2* are positive constant parameters called acceleration coefficients which control the maximum step size of the particle and determine the relative "pull" of *pbest* and *gbest*. The parameter *c1* is a factor determining how much the particle is influenced by the nostalgia or memory of his best location, and *c2* is a factor determining how much the particle is socially influenced by the rest of the swarm. Parameters *r1* and *r2* are two random numbers uniformly distributed in [0, 1] that are used to weight the velocity toward the particle personal best – $(pbest_i - x_i^k)$– and toward the global best solution –$(gbest - x_i^k)$– found so far by the swarm.

According to the velocity equation, a particle decides where to move next, considering its own experience, which is the memory of its best past position, and the experience of the most successful particle in the swarm. The new particle position is determined by adding to the particle current position the new velocity computed.

3 Related work

The last decade has witnessed an astonishingly amount of research in SI [13,18,19]. As shown in recent surveys [20,22], SI has been increasingly applied to distributed job scheduling. However, with regard to scheduling in Cloud environments, very few works can be found to date [15]. Moreover, to the best of our knowledge, no effort aimed to job scheduling based on SI for online Clouds have been proposed and evaluated. By online we mean non-batch scenarios, i.e., where the jobs to be executed in the Cloud is not available beforehand. In most related works, SI techniques are used to specifically solve job scheduling, i.e., they determine how the jobs are assigned to VMs, but few efforts are aimed at solving VM scheduling, i.e., how to allocate VMs to physical resources. Among these two groups we can mention the following works.

In [17] propose a PSO algorithm to schedule jobs that are targeted at paid Clouds, i.e., those that charge users for CPU, storage and network usage, to minimize monetary cost. The algorithm considers both job computation costs and job data transfer costs. Moreover, this approach is based on static resource allocation, which forces users to feed the scheduler with the estimated running times of jobs on the set of Cloud resources to be used. Another relevant approaches are [11,24]. In [11] the authors propose a PSO algorithm to solve the problem of load balancing in VMs. The goal of this work was to minimize the execution time of the jobs. In the work [24] an improved PSO is proposed to reduce job average execution time and increase the rate availability of resources. Finally, in [9] the authors propose a novel self-adaptive Particle Swarm Opti-

mization scheduler to map efficiently a set of VM instances onto a set of Cloud physical resources and reduce energy consumption. Indeed, energy consumption has become a crucial problem [10], on one hand because it has started to limit further performance growth due to expensive electricity bills, and also by the environmental impact in terms of carbon dioxide (CO2) emissions caused by high energy consumption.

Another relevant approaches, but based on ACO, are the works in [6,2,23]. In [6], the authors propose a scheduler to compute the placement of VMs according to the current load of the physical resources and minimize the energy consumption. The authors claim that, from the business perspective, reducing the energy consumption can lead to immense cost reductions. Moreover, the higher power consumption, the higher heat dissipation, and therefore the probability of hardware failures increases. In [2,23] the authors have proposed ACO-based Cloud schedulers for mapping jobs-VMs. The goal was to minimize makespan and maximize load balancing, respectively. Makespan is the maximum execution time of a set of jobs. Flowtime is the sum of job finish times minus job arrive times of a set of jobs. An interesting aspect of [2] is that it was evaluated using real Cloud platforms (Google App Engine and Microsoft Live Mesh), whereas the other work was evaluated through simulations. However, during the experiments, [2] used only 25 jobs and a Cloud comprising 5 machines.

The mentioned do not focus on addressing multiple users, thus rendering difficult their applicability to execute scientific experiments in online, shared Cloud environments. The next Section explains our approach to PSO-inspired Cloud scheduling.

4 Proposed scheduler

We address the scheduling problem where a number of users connect to the Cloud at different times to execute their PSEs, and each user requests the creation of v VMs. A PSE is a set of N independent jobs, each corresponding to a particular value for a variable of the model being studied by the PSE. A user's jobs are distributed and executed on the v VMs created for him. Since the total number of VMs required by all users is usually greater than the number of Cloud physical hosts, a strategy that achieves a good use of these physical resources is needed. This strategy is implemented by means of a support that allocates user VMs to hosts. Moreover, a strategy for assigning user jobs to allocated VMs is also necessary (currently we use FIFO).

To implement the VM allocation part of the scheduler, the Grid scheduler proposed in [12] has been adapted to Clouds. In our adapted algorithm, all hosts belonging to a Cloud are considered a swarm and each host in the Cloud is a particle in this swarm. Following the analogy from the classical PSO, the position of each host in the swarm can be determined by its load (see Algorithm 1). This definition helps to search in the load search space and try to minimize the load. Every time a user requires a VM, it is initialized in a random host (`getInitialHost()`) and each host in the search space takes a position according to its load through the `calculateTotalLoad(hostId)` method. Load refers to the total CPU utilization within a host and is calculated as $load = vmTotalMips/hostTotalMips$, where *vmTotalMips* is an estimation of the amount of processing power used by the VMs that are executing in the host, and *hostTotalMips* is the (hardware-given) total amount of processing power in the host. The neighborhood of

Algorithm 1 PSO-based Cloud scheduler: Core logic

```
Procedure PSOallocationPolicy(vm, hostList)
Begin
  i=0, networkMessages=0, velocity=-1
  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)
      networkMessages++
      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++
  end while
  allocatedHost=hostList.get(destHostId)
  if(!allocatedHost.allocateVM(vm)
      PSOallocationPolicy(vm, hostList)
End
```

that particle is also obtained through `getNeighbors(hostId,neighborSize)`. Each one of the neighbors in the neighborhood are selected randomly as this delivers the best results. The size of the particle neighborhood is a parameter defined by the user.

In each iteration of the algorithm, a particle moves through its neighbors searching for less loaded hosts. The velocity of each particle is defined by the load difference that a host has compared to its other neighbors hosts. If any of the hosts in the neighborhood has a lower load than the original host, then the VM is moved to the neighbor host with a greater velocity. Taking into account that the particles move through hosts of their neighborhood in search of a host with the lower load, the algorithm reaches a local optimum quickly. Thus, each particle makes a move to one of its neighbors, which has the minimum load among all. If all its neighbors are busier than the host itself, the VM 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.

Since each move that a particle performs, involves moving through the network, we consider a tactic to minimize the number of moves: every time a particle moves to a neighbor host that has not allocated VMs yet, the particle allocates its associated VM to it directly without performing further steps. The number of messages sent over the network by a particle to their neighbors hosts to obtain information regarding their availability –load– is accumulated in the `networkMessages` variable.

In our algorithm, there are some issues related to the classical PSO. First, a particle only moves toward its best local neighbor, while in the classical PSO algorithm particles keep track of the best global solutions so far. The velocity of each particle takes the following form: $v_i^{k+1} = (gbest - x_i^k)$. Since we are dealing with a dynamic Cloud

environment, the use of the past experience of each particle is not useful, therefore, zero value is assigned to $c1$ to neutralize the effect of the past history of the particle. Also, the previous velocity should not have effect in the particle decision, therefore, a zero value is assigned to w as well. On the other hand, since we want to use the neighborhood to identify and decide which neighbor represent the best move, we set $c2 = 1$. The formula for updating a particle position is $x_i^{k+1} = x_i^k + v_i^{k+1}$, which is equal to the neighbor load to which the particle have the move, i.e., the position of a particle is its load value, which changes as the particle moves to its neighbors.

5 Evaluation

We have processed a real study case for solving a well-known benchmark problem [7]. The problem involves studying a 18 x 10 m plane strain plate with a central circular hole, with $R = 5$ m. The 2D 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 [7].

First, we run the PSE experiments in a single machine 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 [8]. The machine on which the tests were carried out is an AMD Athlon(tm) 64 X2 Dual Core Processor 3600+, with 2 GBytes of RAM, 400 Gbytes of storage, and a bandwidth of 100 Mbps. By means of the generated job data, we instantiated CloudSim. The experimental scenario consists of a datacenter with 10 physical resources with similar characteristics as the real machine where SOGDE was performed. The characteristics are 4,008 MIPS (processing power), 4 GBytes (RAM), 400 GBytes (storage), 100 Mbps (bandwidth), and 4 CPUs. Then, each user connecting to the Cloud requests v VMs to execute their 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 details about the job data gathering and the CloudSim instantiation process, please see [14].

Moreover, we modeled two online Cloud scenarios in which new users connect to the Cloud every 90 and 120 seconds, respectively, and require the creation of 10 VMs each in which their PSEs –a set of 100 jobs– run (the base job set comprising 25 jobs obtained by varying the value of η was cloned to obtain larger sets). The number of users who connect varies as $u = 10, 20, ..., 100$, and since each user runs one PSE, the total number of jobs to execute is increased as $n = 100 * u$ each time. Each job, called *cloudlet* by CloudSim, had a length which varied between 244,527 and 469,011 Million Instructions (MI). Moreover, each job needs only one processing element (PE) or core to be executed. Then, we assumed a 1-1 job-VM execution model, i.e., jobs within a VM waiting queue are executed one at a time by competing for CPU time with other jobs from other VMs in the same hosts. In other words, a time-shared CPU scheduling policy was used, which ensures fairness. Lastly, the input file size and output file size had 93,082 and 2,202,010 bytes, respectively.

5.1 Performed comparisons

Our PSO algorithm was compared against another three schedulers. First, the *genetic algorithm (GA)* from [1], which is algorithmically related to our work and has been evaluated via CloudSim as well. The population structure represents 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 the Cloud, and the last field in this structure is the fitness field, which indicate the suitability of the hosts in each chromosome. Second, a *Random* allocation in which the VMs requested by the different users are assigned randomly to different physical resources. Although this algorithm does not provide an elaborated criterion to allocate the VMs to physical resources, it offers a popular baseline to evaluate how our scheduler performs. A third alternative scheduler in the form of an *Ideal* scheduler was used, which achieves the best possible allocation of VMs to physical resources in terms of the studied metrics. To allocate all the VMs, the scheduler uses a retry strategy until it is able to serve all users. The number of retries necessary to serve all users and create all requested VMs was 9. This scheduler is only for comparison purposes, however is not viable in practice because of the attempt overhead. In our experiments, the GA-specific parameters were set with the following values: chromosome size = 7 (6 genes for hosts and 1 gen for fitness), population size = 10 *and* number of iterations = 10 and the PSO-specific parameter neighbourhood size = 6.

The experiments measured the trade-off between the number of serviced users by the Cloud and the total number of created VMs among all users. The former increases every time the scheduler successfully allocates any of the requested VMs of a user, and the this latter is considered "serviced". Then, we derived a *weighted metric*, by which the results obtained from the different algorithms have been normalized and weighted with numerical weights. The normalized values for each metric and each user group U joining the Cloud are computed as $NormalValueU_{i=10,...,100} = 1 - \left(\frac{Max(valueU_i) - valueU_i}{Max(valueU_i) - Min(valueU_i)}\right)$, where *valueU* is the obtained value for each one of the basic metrics –serviced users and created VMs– and for each user group connected to the Cloud, *Max(valueU)* and *Min(valueU)* are the maximum and minimum values, respectively, for each basic metric among all the algorithms –PSO, GA, Random, Ideal– and for each user group connected to the Cloud. Moreover, the weighted metric is computed as: $WeightedMetricU_{i=10,...,100} = (wSU * NormalSU_i + wVMs * NormalVMsU_i)$ where *wSU* is the weight given to the number of serviced users by the Cloud (*NormalSU*) and *wVMs* weighs the total number of created VMs (*NormalVMs*). Based on these, and since throughput is often the primary limiting factor in many scientific and engineering efforts, and moreover, many scientists and engineers are interested in obtaining their results as soon as possible, it is important to give importance to both basic metrics. To perform this, we have assigned the pair of weights (wSU, wVMs) equal to (0.50, 0.50).

Table 1 shows the obtained results, which arise from averaging 20 times the execution of each algorithm (little deviations were obtained). Both when users connect to the Cloud every 90 and 120 seconds, the schedulers can not always serve all connected users because a scheduler can not create the requested VMs by the users. The creation of some VMs fails since at the moment a user issues the creation, all physical resources are already fully busy with VMs belonging to other users. Depending on the algorithm,

Table 1: Results: weighted metric

Users connected to the Cloud	Gap = 90				Gap = 120			
	PSO	GA	Random	Ideal	PSO	GA	Random	Ideal
10	0.15	0.02	0.12	1	0.44	0.17	0.31	1
20	0.17	0.04	0.16	1	0.33	0.15	0.23	1
30	0.22	0.07	0.20	1	0.29	0.17	0.22	1
40	0.20	0.1	0.16	1	0.28	0.17	0.20	1
50	0.20	0.1	0.14	1	0.25	0.17	0.18	1
60	0.18	0.11	0.13	1	0.23	0.18	0.16	1
70	0.19	0.11	0.12	1	0.24	0.18	0.15	1
80	0.19	0.11	0.12	1	0.22	0.18	0.14	1
90	0.18	0.12	0.12	1	0.22	0.18	0.14	1
100	0.17	0.12	0.11	1	0.22	0.19	0.13	1

some schedulers are able to find to some extent a host with free resources to which at least one VM per user is allocated. When the scheduler is not able to create at least one of the requested VMs by a user, then the user is considered "not served".

Among all approaches, excluding Ideal which is taken as reference of an ideal performance, GA is the one that serves the least users but creates the most VMs. This is because the population size is 10, and each chromosome contains 6 different hosts, so after 10 iterations GA has more chances to get the host with better fitness, and can thus allocate more VMs to the first users who connect to the Cloud. Random serves more users than PSO and GA but with less VMs. While Random serves many users, it may not be fair with the response times for users, because the algorithm assigns the VMs to physical resources randomly, and many of the creations of the VMs requested by users might fail. There are situations where for a single user Random is able to create only one VM where all jobs of the user are executed. This situation means that the user must wait too long to complete their jobs. Finally, PSO achieves to serve a greater number of users than GA and create a greater number of VMs than Random. As shown, the proposed PSO algorithm delivers the best balance with respect to the number of serviced users and the total number of created VMs, with gains $\%GainPSO = 100 - \frac{(WeightedMetric(GA,Random)*100)}{(WeightedMetric(PSO))}$ of 29.41% over GA and 35.29% over Random in the 100-user scenario and when users connect every 90 seconds. Moreover, when users connect every 120 the gains of PSO over GA and Random is 13.63% and 40.90%, respectively, in that scenario.

Next, we evaluate the number of network messages sent by each one of the studied schedulers. To achieve allocate the VMs into hosts, each scheduler must make a different number of "queries" to hosts to determine their availability upon each VM allocation attempt. These queries are performed through messages sent to hosts over the network to obtain information regarding their availability. Fig. 2 illustrates the number of network messages sent to hosts by each algorithm to allocate the VMs and when users are connected to the the Cloud every 90 seconds (left subfigure) and 120 seconds (right subfigure). The Ideal scheduler needs to send messages to hosts every time a VM

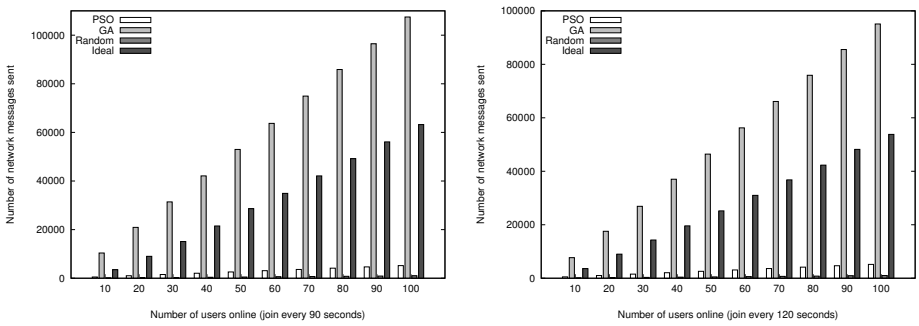

Fig. 2: Results as the number of users increases: Number of network messages

is allocated to know the hosts states and to decide where to allocate the VM. Moreover, as mentioned earlier, Ideal always performs a number of creation retries until all users are served, which makes the number of messages sent even higher. It is important to note, however, that the Ideal algorithm implementation was executed using the back-off strategy with a number of retries equal to 9 to obtain the ideal values to reach. The number of network messages sent to hosts rose from 3,500 to 63,200 and from 3,600 to 53,800 when the number of users connected to the Cloud went from 10 to 100, and users join to Cloud every 90 seconds and 120 seconds, respectively.

Since GA contains a population size of 10 and chromosome sizes of 7 (6 genes for hosts), to calculate the fitness function, the algorithm sends one message for each host of the chromosome to know its availability and obtain the chromosome containing the best fitness value. This is, the VM is allocated to a host belonging to the chromosome with the best fitness value. The number of messages to send is equal to the number of host within each chromosome multiplied by the population size. As Figure 2 shows, GA heavily uses network resources, and the number of network messages sent varied from 10,372.3 to 107,477.1 and from 7,683 to 95,088.8 when the number of connected users was increased from 10 to 100 and the gaps equal to 90 and 120 seconds, respectively.

Random sends one network message to a random host for each attempt of VM creation, making the lowest network resource usage. The number of network messages rose from 100 to 1,000 when the number of connected users to the Cloud went from 10 to 100 and both gaps. Finally, our PSO algorithm, however, makes less use of the network resources than GA and the Ideal scheduler. Due to the fact that we configure the neighborhood size to 6, PSO sends a maximum of *6* messages per VM allocation. Moreover, when PSO finds an unloaded host, it allocates the current VM and does not make any further move. This reduces the total number of network messages sent. The number of network messages sent by PSO to hosts rose from 492.7 to 5,186.7 and from 493.2 to 5,182.5 when the number of users connected to the Cloud went from 10 to 100 and the gaps equal to 90 and 120 seconds, respectively. One point in favor is that, unlike Ideal and GA, PSO sent messages in the order of 100-1000 as Random did.

To conclude, although Random sends few network messages, and the use of the network is important in distributed environments, in most Clouds network interconnections are fast. Moreover, as we shown previously, Random is a very inefficient algorithm in terms of performance because it creates few VMs, and moreover, as we described in our previous work [14], Random gets the worse performance in terms of makespan and

flowtime in batch scenarios. These results are encouraging because they suggest that PSO is close to obtaining the best possible solution balancing all the employed evaluation metrics and making a reasonable use of the network resources.

6 Conclusions

PSEs is a type of simulation that involves running a large number of independent jobs and requires a lot of computing power. These jobs must be efficiently processed in the different computing resources of a distributed environment such as the ones provided by Cloud. Consequently, job scheduling in this context indeed plays a fundamental role. SI-inspired algorithms have received increasing attention in the research community, and refers to the collective behavior that emerges from a swarm of social insects. Through studying social insect colonies, researchers have proposed algorithms for combinational optimal problems. Moreover, job scheduling in Clouds is also a combinational optimal problem, and some SI-inspired schedulers have been proposed.

Existing related efforts do not address in general *online* environments where multiple users connect to scientific Clouds to execute their experiments. To the best of our knowledge, no effort aimed at balancing the number of serviced users in a Cloud and the total number of created VMs by the scheduler exists. We have proposed a Cloud VM scheduler based on PSO that considers these metrics. Simulated experiments performed with CloudSim and real PSE job data suggest that our PSO scheduler provides a good balance to these metrics. As alternative schedulers, we used Genetic Algorithms, Random and an Ideal assignment. We have also evaluated the number of network messages sent to the host by each one of the studied schedulers to allocate the VMs.

We plan to materialize our scheduler on top of a real Cloud platform, e.g., OpenNebula (`http://opennebula.org/`). We will also consider other Cloud scenarios, for example, with heterogeneous machines. We will also evaluate how the variation of the algorithm parameters (e.g., neighborhood size in PSO, chromosome size, population size/number of iterations in GA) influence the performance and network consumption.

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 Society (12-14 December 2011)
2. Banerjee, S., Mukherjee, I., Mahanti, P.: Cloud Computing initiative using modified ant colony framework. In: World Academy of Science, Engineering and Technology. pp. 221–224. WASET (2009)
3. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press (1999)
4. Buyya, R., Yeo, C., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6), 599–616 (2009)
5. 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 (2011)

6. Feller, E., Rilling, L., Morin, C.: Energy-Aware Ant Colony Based Workload Placement in Clouds. In: 12th International Conference on Grid Computing. pp. 26–33. No. 8 in Grid '11, IEEE Computer Society (2011)
7. 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)
8. 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)
9. Jeyarani, R., Nagaveni, N., Vasanth Ram, R.: Design and implementation of adaptive power-aware virtual machine provisioner (APA-VMP) using swarm intelligence. Future Generation Computer Systems 28(5), 811–821 (2012)
10. Liu, Y., Zhu, H.: A survey of the research on power management techniques for high-performance systems. Software Practice & Experience 40(11), 943–964 (October 2010)
11. Liu, Z., Wang, X.: A pso-based algorithm for load balancing in virtual machines of cloud computing environment. In: et al., Y.T. (ed.) Advances in Swarm Intelligence, Lecture Notes in Computer Science, vol. 7331, pp. 142–147. Springer Berlin Heidelberg (2012)
12. Ludwig, S., Moallem, A.: Swarm intelligence approaches for grid load balancing. Journal of Grid Computing 9(3), 279–301 (2011)
13. Mahdiyeh, E., Hussain, S., Mohammad, K., Azah, M.: A survey of the state of the art in particle swarm optimization. Research journal of Applied Sciences, Engineering and Technolog 4(9), 1181–1197 (2012)
14. 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)
15. Pacini, E., Mateos, C., García Garino, C.: Schedulers based on ant colony optimization for parameter sweep experiments in distributed environments. In: Dr. Siddhartha Bhattacharyya, Dr. Paramartha Dutta (ed.) Handbook of Research on Computational Intelligence for Engineering, Science and Business, vol. I, chap. 16, pp. 410–447. IGI Global (2012)
16. Pacini, E., Mateos, C., García Garino, C.: Dynamic scheduling of scientific experiments on clouds using ant colony optimization. In: Third International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering. Civil-Comp Press (2013), paper 33
17. Pandey, S., Wu, L., Guru, S., Buyya, R.: A particle swarm optimization-based heuristic for scheduling workflow applications in Cloud Computing environments. In: International Conference on Advanced Information Networking and Applications. pp. 400–407. IEEE Computer Society (2010)
18. Pedemonte, M., Nesmachnow, S., Cancela, H.: A survey on parallel ant colony optimization. Applied Soft Computing 11(8), 5181–5197 (2011)
19. Poli, R.: Analysis of the publications on the applications of particle swarm optimisation. Journal of Artificial Evolution and Applications (4), 1–10 (2008)
20. Tinghuai, M., Qiaoqiao, Y., Wenjie, L., Donghai, G., Sungyoung, L.: Grid task scheduling: Algorithm review. IETE Technical Review 28(2), 158–167 (2011)
21. Wang, L., Kunze, M., Tao, J., von Laszewski, G.: Towards building a Cloud for scientific applications. Advances in Engineering Software 42(9), 714–722 (2011)
22. Xhafa, F., Abraham, A.: Computational models and heuristic methods for Grid scheduling problems. Future Generation Computer Systems 26(4), 608–621 (2010)
23. Zehua, Z., Xuejie, Z.: A load balancing mechanism based on ant colony and complex network theory in open Cloud Computing federation. In: 2nd International Conference on Industrial Mechatronics and Automation. pp. 240–243. IEEE Computer Society (2010)
24. Zhan, S., Huo, H.: Improved PSO-based Task Scheduling Algorithm in Cloud Computing. Journal of Information & Computational Science 9(13), 3821–3829 (2012)