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

Elina Pacini
ITIC - UNCuyo University. Also CONICET
Mendoza, Mendoza, Argentina
epacini@itu.uncu.edu.ar

and

Cristian Mateos
ISISTAN-CONICET - UNICEN University. Also CONICET
Tandil, Buenos Aires, Argentina
cmateos@conicet.gov.ar

and

Carlos García Garino
ITIC and Facultad de Ingeniería - UNCuyo University
Mendoza, Mendoza, Argentina
cgarcia@itu.uncu.edu.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 hence parallel computing environments must be used. Within these, Infrastructure as a Service (IaaS) Clouds offer custom Virtual Machines (VM) that are launched in appropriate hosts available in a Cloud to handle such jobs. Then, correctly scheduling Cloud hosts is very important and thus efficient scheduling strategies to appropriately allocate VMs to physical resources must be developed. 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 job data from real scientific problems show that our scheduler achieves better performance than schedulers based on Random assignment and Genetic Algorithms. We also study the performance when supplying or not job information to the schedulers, namely a qualitative indication of job length.

Keywords: Parameter Sweep Experiments, Cloud Computing, IaaS, Job Scheduling, Particle Swarm Optimization, Genetic Algorithms

1 Introduction

Scientific computing applies computer science to solve typical scientific problems in disciplines such as Bioinformatics, Earth Sciences, High-Energy Physics, Molecular Science and Social Sciences. Parameter Sweep Experiments (PSEs) is a very popular way of conducting simulation-based experiments among scientists and engineers through which the same application code is run several times with different input parameters resulting in different outputs [1]. PSEs are experiments associated with large-scale computer modeling and simulation, and study how the variation of an individual model variable affect the outcome, for example when analyzing the influence of geometric imperfections in the response of a simple tensile test on steel bars subject to large deformations. By varying variable values, different

sub-studies are obtained, which need to be run on different machines to increase performance.

Cloud Computing [2] suits well in executing these problems, 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 [3]. 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. Under this model, in order to achieve the best performance, VMs have to fully utilize the physical resources [4, 5].

In addition, on an IaaS Cloud, resources are scheduled at two levels: Infrastructure-level, where one or more Cloud infrastructures are created and through a VM scheduler the VMs are allocated into physical machines, and VM-level in which through a job scheduling technique, jobs are assigned for execution into VMs. 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 jobs/VMs in appropriate machines –physical or virtual–. 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) [6] has received increasing attention lately among researchers, and refers to the collective behavior that emerges from social insects swarms to solve complex problems. Hence, researchers have proposed algorithms for combinatorial optimization problems [6]. Moreover, scheduling in Clouds is also a combinatorial optimization problem, and hence schedulers exploiting SI have been proposed.

Unlike previous work [7], where an ant colony optimization (ACO) was used for allocating VMs to physical resources and a priority-based policy for mapping jobs to VMs in batch scheduling scenarios, this paper proposes a particle swarm optimization (PSO) scheduler to allocate VMs to physical Cloud resources, and evaluates its performance in an online Cloud (non-batch) scenario as the one in [3], where multiple users connect to the Cloud at different times to execute their PSEs. Moreover, unlike the paper presented at HPCLatAm 2013, in this paper we study the effects of considering such priority-based policy in conjunction with the PSO scheduler. The motivation is that using the policy requires users to feed the scheduler with a qualitative indication of the length for jobs in a PSE, but performance may be improved with respect to not using the policy. Nevertheless, when the priority-based policy is not used, the user is not forced to supply job length information, which increases usability.

Experiments have been conduced to evaluate performance in respect to 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, thus increasing throughput. Moreover, when more VMs can be allocated, more physical resources can be taken advantage of, and hence PSE execution time decreases. On the other hand, in [7] we showed that our ACO scheduler effectively reduces flowtime and achieves very good makespan levels when the priority-based policy is used. Makespan is the maximum execution time of a set of jobs. Flowtime is the sum of job finish times minus job arrival times of a set of jobs.

In this paper we extend our previous work presented in [8] by incorporating an analysis to evaluate the trade-off between performance –in terms of the number of serviced users and the number of created VMs– and viability. Viability refers to analyzing if the use of the priority-based policy at the VM-level is suitable and convenient in an online Cloud scenario. In this analysis, 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 balances the aforementioned metrics better. Two weights have been assigned to the individual metrics, i.e., a weight 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 [9], together with job data extracted from a real-world PSE and other scheduling algorithms for Clouds show that our PSO performs competitively with respect to the used metrics.

Section 2 gives preliminary concepts. Section 3 surveys related works. After that, Section 4 describes the PSO scheduler and the priority-based policy. Section 5 evaluates the approach. Section 6 concludes the paper.

2 Background

2.1 Cloud Computing basics

Clouds [2] are the current emerging trend in delivering IT services recently. Within a Cloud, applications, data and services are provided across dynamic and potentially geographically dispersed organizations. Clouds [2] refers to a set of technologies that offer computing services through a network, i.e., everything is provided as a service. A related important feature is the ability to scale up and down the computing infrastructure according to resource requirements.

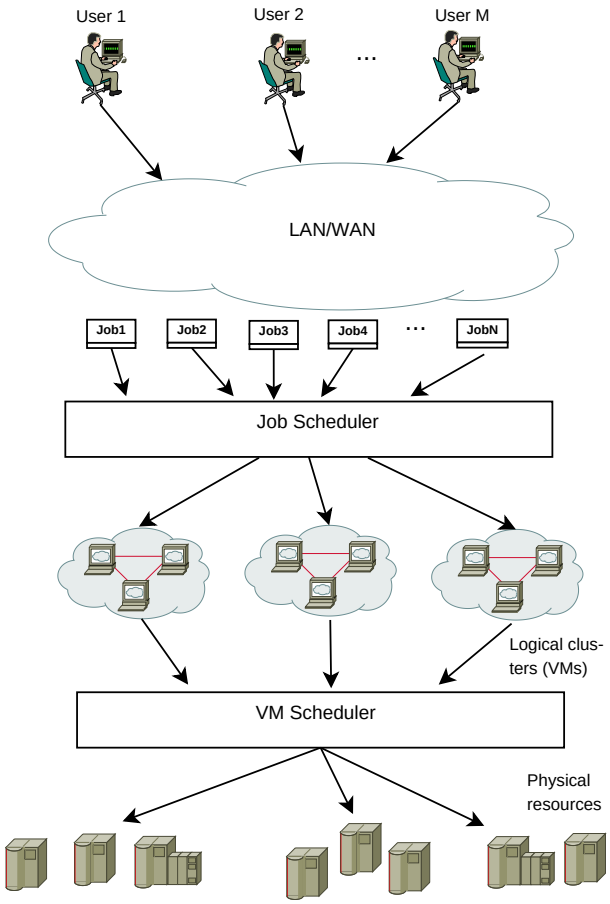

Figure 1: High-level view of a Cloud

Central to any Cloud is the concept of *virtualization*. By means of this support, users exploit Clouds by requesting them VMs that emulate running operating systems on top of several physical machines, which in turn run a host operating system. In addition, users can customize the execution environments or installed software in the VMs according to the needs of their experiments. In commercial Clouds, VMs are provided to users on a pay-per-use basis, i.e., based on cycles of CPU consumed, bytes transferred, etc. 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 [10].

While Clouds help scientific users to run complex applications, job and VM management is a key concern that must be addressed. Broadly, 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, unlike traditional scheduling (e.g., on clusters) where the executing units are mapped directly to physical resources at one level (middleware), 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 mapping 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.

However, job/VM scheduling is NP-complete, and therefore approximation heuristics are necessary.

2.2 Particle Swarm Optimization (PSO)

PSO [6] 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. Imagine 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.

A bee might find a place with a higher concentration of flowers than it had found previously. It would then be drawn to this new location as well as the location of the most flowers found by the whole swarm. Occasionally, one bee may fly over a place with more flowers than had been found by any bee in the swarm. The whole swarm would then be drawn toward that location in additional to their own individual discovery. In this way the bees explore the field by overflying locations of greatest concentration and then being pulled back toward them. Constantly, they are checking the territory they fly over against previously found locations of highest concentration hoping to find the absolute highest concentration of flowers. Eventually, the process leads all bees to the one single place F in the field with the highest concentration of flowers. Soon, all the bees swarm around this point. After that, bees will be unable to find any points of higher flower concentration, so they will be continually drawn back to F.

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 particle 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 Swarm Intelligence (SI) [11, 12, 13]. As shown in recent surveys [14, 15], 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 [16, 17]. Moreover, to the best of our knowledge, no effort aimed to address VM/job scheduling based on SI for online Clouds have been proposed and evaluated. By online we mean non-batch scenarios, i.e., where the user jobs to be executed in the Cloud is not available beforehand. In most related works, SI techniques are used to specifically solve job scheduling only, 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.

[18] proposes 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. Other relevant approaches are [19, 20]. In [19] the authors propose a PSO algorithm to solve the problem of load balancing in VMs. The goal of this work is to minimize the execution time of the jobs. In the work [20] an improved PSO is proposed to reduce job average execution time and increase the rate availability of resources. L. Jing et al. [21] have formulated a model for job-resource mapping and minimize the overall execution cost of jobs. The aim was to reduce the energy consumption and improve the profit. Finally, in the work [22] the authors propose a novel self-adaptive Particle Swarm Optimization scheduler to map efficiently a set of VM instances onto a set of Cloud physical resources and reduce energy consumption. Energy consumption has become a crucial problem [23], 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.

Other relevant approaches, but based on ACO, are explained in the works in [24, 25, 26]. In [24], 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 [25, 26] the authors have proposed ACO-based Cloud schedulers for mapping jobs and VMs. The goal was to minimize makespan and maximize load balancing, respectively. An interesting aspect of [25] 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, [25] used only 25 jobs and a Cloud comprising 5 machines.

Another work, in which authors propose schedulers for mapping VMs to physical resources, such as the SI-based technique proposed in this paper, is [27]. In this work, the authors propose a bio-inspired technique based on Genetic Algorithms (GA) for allocation and scheduling of VMs in federated Clouds. The proposed GA takes into account the capacity of the links connecting among different Cloud providers in order to avoid quality of service degradation for VMs allocated on partner Cloud providers.

Regarding to works where authors propose schedulers which consider job priorities, some relevant results are explained in [28, 29, 30, 31]. Job priority is an important issue in scheduling because several times jobs should be serviced earlier than other those jobs cannot stay for a long time in a system. For example, in our work we assign higher priority values to the jobs with the greatest length. However, all these efforts are aimed to address priority job scheduling in Grids and not in Clouds.

All in all, the presented approaches differ mainly from our proposal that none of the surveyed works address the scheduling problem at the two levels of a Cloud, and moreover, from the mentioned works which deal with Clouds, none of them 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. In this case job execution times between jobs can be very different. This is due to the fact that running the same code or solver against many input values usually yields very dissimilar execution times as well. This situation is undesirable since the user can not process/visualize the outputs of the whole PSE until all jobs finish. Thus, in principle, giving higher (or lower) priority to jobs that are supposed to take longer to finish may help in improving output processing. To address this problem some researchers have considered priority of jobs in the scheduling algorithms [32, 28].

Subsequently, the user's jobs are distributed and executed on the v VMs created for him. Since the total number of VMs required by all users and the total number of jobs to execute are usually greater than the number of Cloud resources, a strategy that achieves a good use of these resources is needed. This strategy is implemented at the two-levels of a Cloud, where by means of a PSO-based scheduler, the user VMs are allocated to physical resources, and then by means of a priority-based policy the user's jobs are allocated to the pre-allocated VMs. Below we describe the derived schedulers both at the Infrastructure-level and at the VM-level.

4.1 PSO-based scheduler at the Infrastructure-level

To implement the VM allocation part of the scheduler, the Grid scheduler proposed in [33] has been adapted to Clouds. In the adapted PSO algorithm, each particle works independently and represents a VM looking for the best host 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 1), every time a user requires a VM, a particle is initialized in a random host (`getInitialHost()`, line number 6), i.e., in a random place of flower in the field. 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 in line number 11. Load refers to the total CPU utilization within a host and is calculated as 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 each particle is composed by the remaining hosts excluding the one in which the particle is initialized, i.e., the neighborhood represents other places in the field with different flower concentration. The neighborhood of that particle is obtained through `getNeighbors(hostId,neighborSize)` in line number 8. Each one of the neighbors –hosts– that compose the neighborhood are selected randomly as it is the technique that delivered the best results in the performed experiments. Moreover, the size of the particle neighborhood is a parameter defined by the user. In our experiments, this parameter is directly associated to the number of hosts that constitute the Cloud.

Algorithm 1 PSO Infrastructure-level Cloud scheduler: Core logic

```
1  Procedure PSOallocationPolicy(vm, hostList)
2  Begin
3    i=0, networkMessages=0
4    velocity=-1
5    particle = new Particle(vm, hostList)
6    initialHostId = particle.getInitialHost()
7    currentPositionLoad = particle.calculateTotalLoad(initialHostId)
8    neighbours = particle.getNeighbours(initialHostId , neighbourSize)
9    While (i < neighbours.size()) do
10         neighbourId = neighbours.get(i)
11     destPositionLoad = particle.calculateTotalLoad(neighbourId)
12         networkMessages=networkMessages+1
13       if(destPositionLoad == 0)
14         currentPositionLoad = destPositionLoad
15         destHostId = neighbours.get(i)
16         i=neighbours.size()
17       end if
18       if(currentPositionLoad - destPositionLoad > velocity)
19         velocity = currentPositionLoad - destPositionLoad
20         currentPositionLoad = destPositionLoad
21         destHostId = neighbours.get(i)
22       end if
23   i=i+1
24   end while
25   allocatedHost=hostList.get(destHostId)
26   if(!allocatedHost.allocateVM(vm)
27       PSOallocationPolicy(vm, hostList)
28 End
```

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 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.

Since each move a particle performs involves moving through the network, we have added a control to minimize the number of moves that a particle performs: 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 number of messages sent over the network by a particle from the associated host to their neighbors hosts to obtain information regarding their availability –load– is accumulated in the `networkMessages` variable `--line number 12--` every time a particle moves.

In our algorithm, there are some issues related to the classical PSO. In the former, 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 environment in which the users are requesting VMs and sending jobs all the time, the use of the past experience of each particle is not useful, therefore, zero value is assigned to $c1$ in order to omit the effect of the past history of the particle. Also again, because of the dynamic nature of the problem, 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 –host– is the better to move, a value of one have been used for $c2$. Finally, the parameter $r2$ is equal to one to weight the velocity of the global best solution found among all the neighborhood, i.e., the velocity to the least loaded host. 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 moves, i.e., the position of a particle is the load value of its associated host and it changes while the particle is moved to its neighbors.

4.2 Priority-based scheduler at the VM-level

Before submitting VMs and jobs to the Cloud, each one of the jobs of a PSE can be tagged with a *qualitative* priority represented as an integer value. Priorities are assigned in terms of the relative estimated execution time required by each PSE job. These priorities values are provided by the disciplinary user, who because of his experience, has arguably extensive knowledge about the problem to be solved from a modeling perspective, and therefore can estimate the individual time requirements of each job with respect to the rest of the jobs of his/her PSE. In other words, the user can identify which are the individual experiments within a PSE that might require more time to obtain a solution.

The estimated execution time of each experiment depends on the particular problem at hand and chosen PSE variable values. Once the user has identified the experiments that may require more time to be executed, a simple tagging strategy is applied to assign a "category" (number) to each job. These categories represent the priority degree that will have a job with respect to the others in the same PSE, e.g., high priority, medium priority or low priority, but other categories could be defined. Despite jobs can demand different times to execute and the user can identify those that take longer, the number of categories should be limited for usability reasons.

At the time of executing the priority-based policy (see Algorithm 2), i.e., once the VMs have been created and allocated in physical machines, the scheduler takes into account the priority value of each job (if given) to determine the order in which jobs will be executed. The scheduler processes the jobs with higher priority (or heavier) first and then the remaining jobs.

Algorithm 2 The SubmitJobsToVMsByPriority procedure

```
1  Procedure SubmitJobsToVMsByPriority(jobList)
2  Begin
3    vmIndex=0
4    while (jobList.size() > 0)
5      job=jobList.getJobSortedByPriority()
6      vm=getVMsList(vmIndex)
7      vm.scheduleJobToVM(job)
8      vmIndex=Mod(vmIndex+1,getVMsList().size())
9      jobList.remove(job)
10   end while
11 End
```

To carry out the assignment of jobs to VMs, this sub-algorithm uses two lists, one containing the jobs that have been sent by the user, and the other list contains all user VMs that are already allocated to a physical resource and hence are ready to execute jobs. The procedure iterates the list of all jobs –`jobsList` in line number 4– and then, through `getJobSortedByPriority()` method –line number 5– 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 the `jobList` it is submitted to be executed in a VM in a round robin fashion. The VM where the job is executed is obtained through the method `getVmsList(vmIndex)` –line number 6–. Internally, the algorithm maintains a queue for each VM that contains a list of jobs that have been assigned to be executed. The procedure is repeated until all jobs have been submitted for execution.

5 Evaluation

We have processed a real study case for solving a well-known benchmark problem [34]. 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 [34].

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 [35]. 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 [7].

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 [27], 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 [36, 37] 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 neighborhood size = 6.

In the experiments we measure on one hand, 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 user is considered "serviced". On the other hand, we analyze the trade-off between performance –in terms of serviced users and created VMs– and usability when using the priority-based policy at the VM-level in an online Cloud scenario compared to not considering priorities (using FIFO policy).

Then, based on these two performance metrics, we derived a *bi-objective weighted metric*, by which the results obtained from the different algorithms have been normalized and weighted with numerical values. 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).

We report the results when executing PSEs submitted by multiple users in the simulated Cloud using our two-level scheduler, which arise from averaging 20 times the execution of each algorithm. Previously, to select the appropriate number of executions for reporting the results, experiments were performed with different numbers of executions: 15, 20, 25 and 30. Although the higher the number of executions, the more accurate the obtained results, deviations in

Table 1: Weighted metric when using the priority-based policy at the VM-level

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	0.10	0.16	1	0.28	0.17	0.20	1
50	0.20	0.10	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

the order of 1/100 were obtained in the results when the number of executions increased between 15 and 30. Then, 20 executions allowed us to have valid results.

Table 1 shows the weighted metric for the different algorithms. The first column illustrates the number of users trying to connect to the Cloud, and moreover, each row represents a different scenario. Specifically, the first row represents the situation where up to 10 users are connected but not all can be serviced. In the second row up to 20 users are connected, whereas in the third row up to 30 users connect, and so on. Then, the following columns shows the results of the weighted metric for each one of the algorithms and for each connection gap.

As can be seen, 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, 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. It is important to note that, while Random serves many users, it is in general not fair with the response times for users, producing a very large flowtime [7]. The reason behind this is that the Random algorithm assigns the VMs to physical resources randomly, and many of the creations of the VMs requested by users might fail. There are situations where for a single user Random is able to create only one VM where all jobs of the user are executed. This situation means that the user must wait too long to complete their jobs and thus loses the benefit of using a Cloud. 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 are 13.63% and 40.90%, respectively, in that scenario.

Next, we study whether it is convenient or not to use the priority-based policy for mapping user jobs in online Cloud scenarios. This analysis is performed because although the disciplinary user is the one who know within which range of parameters values the jobs taking more time to run are, the manual tagging of all jobs becomes a tedious and difficult task. Therefore it is important to determine whether this task performed by the disciplinary user has impact on the performance obtained by the different schedulers.

In Table 2 we show the results of the weighted metric when job information is not provided. As can be seen, the weighted metric values are very close to the values obtained when using the priority-based policy in most cases. However, there are some observations that can be highlighted. Firstly, the PSO algorithm still offers the best balance in terms of serviced users and created VMs with respect to its competitors. This suggest that the strategy implemented at the Cloud Infrastructure-level has a significant impact on the results. Secondly, the use of the priority-based policy for allocating jobs in the VMs seems to be more suitable when the load is moderate, i.e., when fewer users try to connect

Table 2: Weighted metric when not using the priority-based policy at the VM-level

Users connected to the Cloud	Gap = 90				Gap = 120			
	PSO	GA	Random	Ideal	PSO	GA	Random	Ideal
10	0.12	0.04	0.09	1	0.37	0.14	0.30	1
20	0.16	0.04	0.13	1	0.31	0.14	0.22	1
30	0.21	0.08	0.18	1	0.28	0.11	0.23	1
40	0.20	0.09	0.17	1	0.27	0.17	0.21	1
50	0.20	0.11	0.14	1	0.25	0.16	0.16	1
60	0.19	0.11	0.14	1	0.23	0.17	0.16	1
70	0.19	0.11	0.11	1	0.23	0.18	0.16	1
80	0.19	0.11	0.13	1	0.22	0.18	0.15	1
90	0.18	0.11	0.11	1	0.22	0.18	0.14	1
100	0.17	0.11	0.10	1	0.21	0.18	0.14	1

to the Cloud. For instance, when users connect to the Cloud every 90 seconds, starting from 30 users, the weighted metric when using the priority-based policy does not improve the results when priorities are not used. On the other hand, when users connect to the Cloud every 120 seconds, the weighted metric when using the priority-based policy is equals to that of not using priorities, starting from 50 users. This means that the higher the gap connection time the more acute the difference in favor of using job priorities, assuming moderate load.

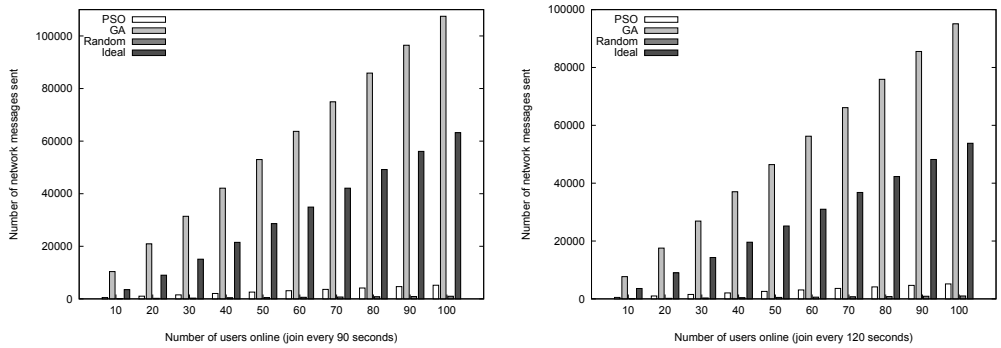

Figure 2: Results as the number of users increases: Number of network messages

Finally, we evaluate the number of network messages sent by each one of the studied schedulers, irrespective of the VM-level scheduling policy used. 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 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 a back-off strategy to retry allocating failing VMs 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 [7], Random gets the worse performance in terms of makespan and flowtime in batch scenarios. These results are encouraging because they indicate 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

Scientists and engineers are nowadays faced to the need of computational power to satisfy the increasing resource intensive nature of their simulations. For example, 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 and thus many variants based on approximation techniques have been proposed. Besides, in Clouds, scheduling is performed at two levels (job and VM), making the problem even more challenging compared to other distributed environments.

Recently, 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.

In this paper we have described a two-level Cloud scheduler based on SI, particularly PSO, that operates under the IaaS model and pays special attention to the balance both throughput and response time in an online Cloud. Moreover, we have explored whether the use of a priority-based policy at the VM-level is suitable for this application scenario. Simulated experiments performed with CloudSim and real PSE job data show on one hand that our PSO scheduler provides a good balance to these metrics. Besides, the use of the priority-based policy is more convenient when fewer users are connected to the Cloud because it has little effect when more users (40-50 and beyond) connect to the Cloud. 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.

In the future, we plan to materialize our scheduler on top of a real Cloud platform, e.g., OpenNebula (`http://opennebula.org/`). Second, we will consider other Cloud scenarios, e.g., federated Clouds with heterogeneous physical resources belong to different Cloud providers devoted to create an uniform Cloud resource interface to users. Moreover, an interesting research line in federated domains is the study of interconnection capacities –network links– among the domains. Indeed, in [27] a GA-based solution for the problem has been proposed, and therefore we aim at studying the usefulness of PSO in this context.

Finally, since our work is focused on the IaaS model where custom VMs are launched to be executed in the hosts available in a datacenter, energy consumption is another important issue. When complex policies such as PSO or GA are used, the balance between throughput and response time is better, but CPU usage, access to memory and transfer through the network are higher compared to that of more simpler policies, e.g., a Random policy. Therefore, to execute many jobs or create a large number of VMs, the accumulated resource usage overhead may be significant, resulting in higher demands for energy, which should be assessed.

Acknowledgments

We thank the HPCLatAm and the Special Issue reviewers for their helpful suggestions to improve the paper. The first author acknowledges her Ph.D. scholarship granted by the CONICET and formerly by UNCuyo and ANPCYT through PRH project. This research is supported by the ANPCyT (PAE-PICT 2007 program, projects 2311 and 2312), and UNCuyo (grant 06/B253).

References

[1] C. Youn and T. Kaiser, "Management of a parameter sweep for scientific applications on cluster environments," *Concurrency & Computation: Practice & Experience*, vol. 22, pp. 2381–2400, 2010.

[2] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[3] E. Pacini, C. Mateos, and C. García Garino, "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.

[4] H. Salimi, M. Najafzadeh, and M. Sharifi, "Advantages, Challenges and Optimizations of Virtual Machine Scheduling in Cloud Computing Environments," *International Journal of Computer Theory and Engineering*, vol. 4, no. 2, pp. 189–193, 2012.

[5] 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*, vol. 28, no. 2, pp. 358 – 367, 2012.

[6] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[7] 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*, vol. 56, pp. 38–50, 2013.

[8] E. Pacini, C. Mateos, and C. García Garino, "Dynamic scheduling based on particle swarm optimization for cloud-based scientific experiments," in *VI Latin American Symposium on High Performance Computing (HPCLatAm 2013)*. Mendoza, Argentina: ITIC, ICB, UNCuyo, July 2013, pp. 1–12.

[9] 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*, vol. 41, no. 1, pp. 23–50, 2011.

[10] L. Wang, M. Kunze, J. Tao, and G. von Laszewski, "Towards building a Cloud for scientific applications," *Advances in Engineering Software*, vol. 42, no. 9, pp. 714–722, 2011.

[11] 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 Technolog*, vol. 4, no. 9, pp. 1181–1197, 2012.

[12] M. Pedemonte, S. Nesmachnow, and H. Cancela, "A survey on parallel ant colony optimization," *Applied Soft Computing*, vol. 11, no. 8, pp. 5181–5197, 2011.

[13] R. Poli, "Analysis of the publications on the applications of particle swarm optimisation," *Journal of Artificial Evolution and Applications*, no. 4, pp. 1–10, 2008.

[14] M. Tinghuai, Y. Qiaoqiao, L. Wenjie, G. Donghai, and L. Sungyoung, "Grid task scheduling: Algorithm review," *IETE Technical Review*, vol. 28, no. 2, pp. 158–167, 2011.

[15] F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future Generation Computer Systems*, vol. 26, no. 4, pp. 608–621, 2010.

[16] E. Pacini, C. Mateos, and C. García Garino, "Schedulers based on ant colony optimization for parameter sweep experiments in distributed environments," in *Handbook of Research on Computational Intelligence for Engineering, Science and Business*, Dr. Siddhartha Bhattacharyya, Dr. Paramartha Dutta, Ed. IGI Global, 2012, vol. I, ch. 16, pp. 410–447.

[17] E. Pacini, C. Mateos, and C. García Garino, "Distributed job scheduling based on Swarm Intelligence: A survey," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 252 – 269, 2014.

[18] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in Cloud Computing environments," in *International Conference on Advanced Information Networking and Applications*. IEEE Computer Society, 2010, pp. 400–407.

[19] Z. Liu and X. Wang, "A pso-based algorithm for load balancing in virtual machines of cloud computing environment," in *Advances in Swarm Intelligence*, ser. Lecture Notes in Computer Science, Y. T. et al., Ed. Springer Berlin Heidelberg, 2012, vol. 7331, pp. 142–147.

[20] S. Zhan and H. Huo, "Improved PSO-based Task Scheduling Algorithm in Cloud Computing," *Journal of Information & Computational Science*, vol. 9, no. 13, pp. 3821–3829, 2012.

[21] J. Liu, X. Guo Luo, and X. M. F. Zhang, "Job Scheduling Algorithm for Cloud Computing Based on Particle Swarm Optimization," *Advanced Materials Research*, vol. 662, pp. 957–960, 2013.

[22] 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*, vol. 28, no. 5, pp. 811–821, 2012.

[23] Y. Liu and H. Zhu, "A survey of the research on power management techniques for high-performance systems," *Software Practice & Experience*, vol. 40, no. 11, pp. 943–964, October 2010.

[24] E. Feller, L. Rilling, and C. Morin, "Energy-Aware Ant Colony Based Workload Placement in Clouds," in *12th International Conference on Grid Computing*, ser. Grid '11, no. 8. IEEE Computer Society, 2011, pp. 26–33.

[25] S. Banerjee, I. Mukherjee, and P. Mahanti, "Cloud Computing initiative using modified ant colony framework," in *World Academy of Science, Engineering and Technology*. WASET, 2009, pp. 221–224.

[26] Z. Zehua and Z. Xuejie, "A load balancing mechanism based on ant colony and complex network theory in open Cloud Computing federation," in *2nd International Conference on Industrial Mechatronics and Automation*. IEEE Computer Society, 2010, pp. 240–243.

[27] L. Agostinho, G. Feliciano, L. Olivi, E. Cardozo, and E. Guimaraes, "A Bio-inspired Approach to Provisioning of Virtual Resources in Federated Clouds," in *Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, ser. DASC 11. IEEE Computer Society, 12-14 December 2011, pp. 598–604.

[28] M. Zafril, A. Kamalrulnizam, A. Hanan, S. Shahir, and W. Nurulsafawati, "Performance comparison of priority rule scheduling algorithms using different inter arrival time jobs in grid environment," *International Journal of Grid and Distributed Computing*, vol. 4, no. 3, pp. 61–70, 2011.

[29] K. Gkoutioudi and H. Karatza, "Multi-criteria job scheduling in grid using an accelerated genetic algorithm," *Journal of Grid Computing*, vol. 10, no. 2, pp. 311–323, 2012.

[30] S. Bansal, B. Kothari, and C. Hota, "Dynamic task-scheduling in grid computing using prioritized round robin algorithm," *International Journal of Computer Science Issues*, vol. 8, no. 2, pp. 472–477, 2011.

[31] P. Mousumi, S. Debabrata, and S. Goutam, "Dynamic job scheduling in cloud computing based on horizontal load balancing," *International Journal of Computer Technology and Applications*, vol. 2, no. 5, pp. 1552–1556, 2011.

[32] S. Ghanbari and M. Othman, "A priority based job scheduling algorithm in cloud computing," *Procedia Engineering*, vol. 50, pp. 778–785, 2012.

[33] S. Ludwig and A. Moallem, "Swarm intelligence approaches for grid load balancing," *Journal of Grid Computing*, vol. 9, no. 3, pp. 279–301, 2011.

[34] 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*, vol. 246, pp. 174–184, Jul. 2013.

[35] 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*, vol. 42, no. 13, pp. 1187–1197, 2006.

[36] J. Changhee and N. Shroff, "Performance of random access scheduling schemes in multi-hop wireless networks," in *Fortieth Asilomar Conference on Signals, Systems and Computers*, ser. ACSSC '06, 2006, pp. 1937–1941.

[37] S. Seiden, “Barely random algorithms for multiprocessor scheduling,” *Journal of Scheduling*, vol. 6, no. 3, pp. 309–334, 2003.