This article is a pre-print of the article "E. Pacini, C. Mateos and C. GarcÃa Garino: "SI-based Scheduling of Scientific Experiments on Clouds".
IDAACS'2013. Vol. 2, pp. 699-704. Berlin, Germany. 2013. IEEE"
The published version is available at http://dx.doi.org/10.1109/IDAACS.2013.6663015

SI-based Scheduling of Scientific Experiments on Clouds

Elina Pacini[1], Cristian Mateos[2], Carlos García Garino[1,3]
[1] ITIC, Universidad Nacional de Cuyo, Mendoza, Argentina, epacini@itu.uncu.edu.ar
[2] ISISTAN-CONICET, Tandil, Buenos Aires, Argentina, cmateos@conicet.gov.ar
[3] Facultad de Ingeniería, Universidad Nacional de Cuyo, Mendoza, Argentina, cgarcia@itu.uncu.edu.ar

Abstract – **Scientists and engineers usually require huge amounts of computing power for performing their experiments. Precisely, Parameter Sweep Experiments (PSE) allow these kind of users to perform simulations by running the same scientific code with different input data, which results in many CPU-intensive jobs and thus computing environments such as Clouds must be used. We describe two Cloud schedulers based on two popular swarm intelligence (SI) techniques, namely ant colony optimization (ACO) and particle swarm optimization (PSO), to allocate virtual machines (VM) to physical Cloud resources. The main performance metrics to study are the number of serviced users by the Cloud –i.e., the number of Cloud users that the scheduler is able to successfully serve– and the total number of created VMs, in dynamic (non-batch) scheduling scenarios. Simulated experiments performed by using CloudSim and real PSE job data suggest that our schedulers, through a weighted metric, perform competitively with respect to the number of serviced users and achieve an effective assignment of VMs compared to a scheduler based on Genetic Algorithms.**

Keywords – **Parameter sweep experiments; Cloud computing; Scheduling; Ant colony optimization; Particle swarm optimization; Genetic algorithms**

I. Introduction

Scientists and engineers are nowadays faced to the need of computational power to satisfy the ever-increasing resource intensive nature of their experiments. PSEs, for example, consist of the repeated execution of the same application code with different input parameters resulting in different outputs. PSEs have the major advantage of generating lists of independent jobs [1], since the experiments are executed under multiple initial configurations many times. This makes these kinds of studies embarrassingly parallel from a computational perspective. Therefore, PSEs are suited for distributed/parallel computing.

A computing paradigm that is gaining momentum is Cloud Computing [2], which bases on the idea of providing an on demand computing infrastructure to end users. Typically, users exploit Clouds by instantiating one or more machine images to create virtual machines running a desired operating system on top of several physical machines. Since a Cloud can be considered as a pool of virtualized computing resources, it allows the dynamic scaling of applications by provisioning resources via virtualization. The various VMs are distributed among different physical resources or consolidated to the same machine in order to increase resource utilization. To perform this, scheduling the basic processing units on a Cloud is an important issue and it is necessary to develop efficient scheduling strategies to appropriately allocate the VMs to physical resources. In this context, "scheduling" refers to the way VMs are allocated to run on the available computing resources, since there are typically many more VMs running than physical resources. However, scheduling is an NP-complete problem and therefore it is not trivial from an algorithmic perspective.

Recently, swarm intelligence (SI) has received increasing attention for this problem [3]. The concept refers to the collective behavior that emerges from a swarm of social insects [4]. Social insect colonies collectively solve complex problems that are beyond the capabilities of each individual insect, and the cooperation among them is self-organized without any central supervision. Inspired by these capabilities, researchers have proposed algorithms for combinatorial optimization problems. Scheduling in Clouds is also a combinatorial optimization problem, and some schedulers in this line that exploit SI have been proposed [5], however they have been superficially evaluated.

In this paper, we describe two schedulers, one based on ant colony optimization (ACO) and another exploiting particle swarm optimization (PSO), to allocate VMs to physical Cloud resources. Unlike previous work of our own [3], the aim of this paper is to experiment in a dynamic Cloud (non-batch) scenario in which multiple users connect to the Cloud at different times to execute their PSEs. The main performance metrics to study are the number of serviced users among all users that are connected to the Cloud, and the total number of VMs successfully handled by the scheduler. Simulated experiments performed with job data extracted from a real-world PSE [6] involving a viscoplastic problem suggest that the proposed SI schedulers deliver competitive performance with respect to the number of serviced users, i.e., the

This article is a pre-print of the article "E. Pacini, C. Mateos and C. GarcÃ-a Garino: "SI-based Scheduling of Scientific Experiments on Clouds". IDAACS’2013. Vol. pp. 699-704. Berlin, Germany. 2013. IEEE"
The published version is available at http://dx.doi.org/10.1109/IDAACS.2013.6663015

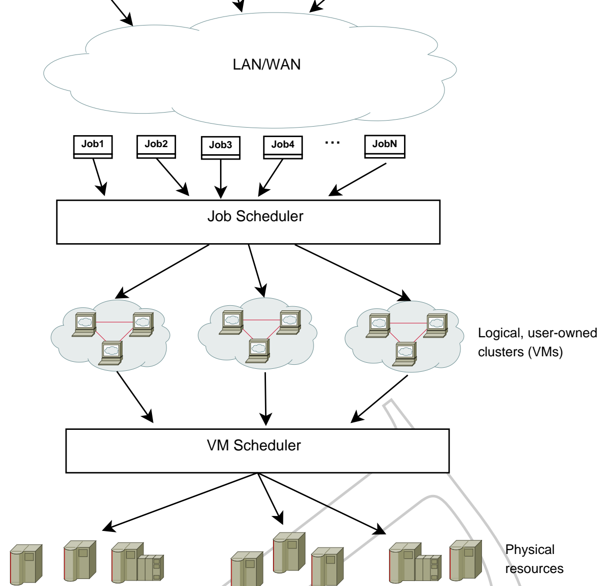

Figure 1. High-level view of a two levels Cloud scheduler

number of Cloud users that the scheduler is able to successfully serve, and the VMs that can be created and allocated. Experiments were performed by using the CloudSim simulator [7].

Sections II and III give background necessary to understand the concepts underpinning our schedulers. Section IV describe our schedulers. Section V presents the evaluation of the schedulers. Section VI concludes the paper and discuss future prospective extensions.

II. Cloud infrastructures: Overview

A Cloud [2] is basically a dynamic pool of virtualized computing resources. Virtualization refers to the capability of a software system of emulating various operating systems. Clouds allow the dynamic scaling of users applications by the provision of computing resources via VMs, which instantiate one *machine image*. VMs can be distributed among different physical machines or consolidated to the same machine to balance the load or increase the CPU utilization. In addition, users can customize the software installed in the VMs according to their experimental needs.

In contrast to traditional job scheduling (e.g., on clusters) wherein executing units are mapped directly to physical resources at one (middleware) level, on a virtualized environment resources need to be scheduled at two levels (see Figure 1). In the first level (bottom), one or more virtual “Cloud infrastructures” or logical clusters are created and through a VM scheduler the VMs are allocated into real hardware. In the second level (top), by using job scheduling techniques, jobs are assigned for execution into these virtual resources. Broadly, job scheduling is a mechanism that maps jobs to appropriate resources to execute, and the delivered efficiency directly affects the performance of the whole distributed environment. Furthermore, a Cloud can be dynamic, i.e., one or more scientific users can unpredictably connect to the Cloud via a network and require the creation of a number of VMs for executing their experiments (a set of jobs).

Although the use of Cloud infrastructures helps scientific users to run complex applications, job and VM management is a key concern that must be addressed. Particularly, in this work we focus on the problem of more efficiently solve the allocation of VMs to physical resources in a dynamic, multi-user Cloud. However, scheduling is in general NP-complete, and therefore approximation heuristics are necessary.

III. Swarm Intelligence

Since artificial life techniques have been effective in combinational optimization problems, they result good alternatives to achieve the goals proposed in this work. SI represents a set of artificial life techniques that has received increasing attention for solving this type of problems [5]. The advantage of SI derives from their ability to explore solutions in large search spaces in a very efficient way. All in all, using SI techniques remains an interesting approach to cope in practice with the NP-completeness of job scheduling problems.

Particularly, the ACO algorithm [8] arise from the way real ants behave in nature, i.e., from the observation of ant colonies when they search the shortest paths to reach a food source from their nest. In nature, real ants move randomly from one place to another to search for food, and upon finding food and returning to their nest each ant leaves a *pheromone* that lures other working ants to the same course. When more and more ants choose the same path, the pheromone trail is reinforced and even more ants will further choose it. This positive feedback eventually leaves all the ants following few paths. Over time the shortest paths will be intensified by the pheromone faster since the ants will both reach the food source and travel back to their nest at a faster rate.

In practice, to optimize job scheduling problems, the ACO algorithm has the advantage of being most of the time easily visualized via graphical representations. For example, graph designs are used to identify the problem, including jobs and executing physical resources or machines (nodes) and scheduling decisions (arcs). Here, each job can be carried out by an ant. Ants cooperatively search for machines with available computing resources and deliver the jobs to these machines. In our work, moreover, each ant carries one VM.

On the other hand, the PSO algorithm [9] is an attempt to mimic the behavior of natural processes from animals such as birds and insects such as bees. The PSO algorithm

This article is a pre-print of the article "E. Pacini, C. Mateos and C. GarcÃ-a Garino: "SI-based Scheduling of Scientific Experiments on Clouds". IDAACS'2013. Vol. 2, pp. 699-704. Berlin, Germany. 2013. IEEE"
The published version is available at http://dx.doi.org/10.1109/IDAACS.2013.6663015

can be employed to tackle down optimization problems by modeling the social behavior of these animals or insects. In the algorithm the general term "particle" is used to represent birds, bees or any other individuals who exhibit social and group behavior.

An example based on nature to illustrate the algorithm is as follows: a group of bees flies over the countryside looking for flowers. Their goal is to find as many flowers as possible. At the beginning, bees do not have knowledge of the field and fly to random locations with random velocities looking for flowers. Each bee has the capability of remembering 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.

When used to model job scheduling, a PSO algorithm instantiation consists of particles, each maintaining one potential solution to the entire problem. The design of the representation of a particle is different for each type of problem. An example of how to represent particles when modeling the job scheduling problem is placing the position of a particle in a search space containing n dimensions. Each dimension i is the job to schedule, and the corresponding value (natural number) represents the machine in which a job can be allocated. Furthermore, the global best position will indicate the best possible schedule. Again, in our work, particles represent VMs.

IV. Proposed schedulers: Brief description

Conceptually, the scheduling problem in this paper can be formulated as follows. A number of users are connected to the Cloud at different times to execute their PSEs. To perform this, each user requests to the Cloud the creation of v VMs. A PSE is formally defined as a set of $N = 1, 2, ..., n$ independent jobs, where each job corresponds to a particular value for a variable of the model being studied by the PSE. The jobs are distributed and executed on the v VMs created by the corresponding user. Since the total number of VMs required by all users is greater than the number of Cloud physical resources (i.e., hosts), a strategy that achieves a good use of these physical resources must be implemented. 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, AntZ and ParticleZ, the algorithms proposed in [10] to solve the problem of load balancing in Grid environments have been adapted to be used in Clouds.

Algorithm 1 ACO-based Cloud scheduler: Core logic

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

In our adapted ACO algorithm, each ant works independently and represents a VM "looking" for the best host to which it can be allocated. When a VM is requested by a user, an ant is initialized. A master table containing information on the load of each host is initialized. Subsequently, if an ant associated to the VM that is executing the algorithm already exists, the ant is obtained from a pool of ants. If the VM does not exist in the ant pool, then a new ant is created. To do this, first, a list of all suitable hosts to which the VM can be allocated is obtained. Each working ant and its associated VM are added to the ant pool and the ACO-specific mechanism starts to operate (see Algorithm 1). In each iteration, the ant collects the load information of the host that is visiting and adds this information to its private load history. The ant then updates a load information table of visited hosts (`localLoadTable.update()`), which is maintained in each host. This table contains information of other hosts, which were added to the table when other ants visited the host. The load is calculated on each host taking into account the CPU utilization made by all the VMs that are executing on each host, i.e., $load = numberOfExecutingVMs/numberOfPEsInHost$, where *numberOfExecutingVMs* is the number of VMs that are executing in the host, and *numberOfPEsInHost* is the total number of processing elements (cores) in the host. This metric is useful for an ant to choose the least loaded host to allocate its VM.

When an ant moves from one host to another it has two choices: moving to a random host using a constant probability or *mutation rate*, or using the load table information of the current host (`chooseNextStep()`). The mutation rate decreases with a *decay rate* factor as

This article is a pre-print of the article "E. Pacini, C. Mateos and C. GarcÃ-a Garino: "SI-based Scheduling of Scientific Experiments on Clouds". IDAACS'2013. Vol. 2, pp. 699-704. Berlin, Germany. 2013. IEEE
The published version is available at http://dx.doi.org/10.1109/IDAACS.2013.6663015

time passes, thus, the ant will be more dependent on load information than to random choice. This process is repeated until the finishing criterion is met. The completion criterion is equal to a predefined number of steps (*maxSteps*). Finally, the ant delivers its VM to the current host and finishes its task.

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

Algorithm 2 PSO-based Cloud scheduler: Core logic

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

Moreover, in our adapted PSO algorithm, all hosts belonging to a Cloud are considered as a flock or a swarm, and each host in the Cloud is a particle in this flock. Following the analogy from the classical PSO, the position of each host in the flock can be determined by its load. This definition helps to search in the load search space and try to minimize the load. In our algorithm (see Algorithm 2), every time a user requires a VM, this latter 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 initial host, and *hostTotalMips* is the (hardware-given) total amount of processing power in the host. The neighborhood of that particle is also obtained through `getNeighbors(hostId,neighborSize)`. Each one of the neighbors that compose the neighborhood are selected randomly as it is the technique that delivers the best results according to our preliminary assessment. The size of the particle neighborhood is a parameter defined by the user.

In each iteration of the algorithm, the particle moves through its neighbors in search of a hosts with a lower load. 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.

Lastly, once the VMs have been allocated to physical resources by an SI technique (ACO or PSO), the scheduler proceeds to assign the jobs to these VMs. This sub-algorithm uses two lists, one containing the jobs that have been sent by the user, i.e., a PSE, and the other list contains all user VMs that are already allocated to a physical resource and hence are ready to execute jobs. The algorithm iterates the list of all jobs, and then retrieves jobs through a FIFO policy. Each time a job is obtained from the list it is submitted to be executed in a VM in a round robin fashion. Internally, the algorithm maintains a queue for each VM that contains its list of jobs to be executed. The procedure is repeated until all jobs have been submitted for execution using the allocated VMs.

V. Evaluation

To assess the effectiveness of our schedulers in a non-batch Cloud environment where multiple users dynamically connect to execute their PSEs, we have processed a real study case for solving a well-known benchmark problem discussed in [6]. The problem involves studying a plane strain plate with a central circular hole. The dimensions of the plate were 18 x 10 m, with $R = 5$ m. The 2D finite element mesh used had 1,152 elements. Unlike previous studies of our own [11], in which a geometry parameter –imperfection– was chosen to generate the PSE jobs, in this case a material parameter –viscosity– 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

This article is a pre-print of the article "E. Pacini, C. Mateos and C. GarcÃ-a Garino: "SI-based Scheduling of Scientific Experiments on Clouds". IDAACS'2013. Vol. 2, pp. 699-704. Berlin, Germany. 2013. IEEE"
The published version is available at http://dx.doi.org/10.1109/IDAACS.2013.6663015

$1 \cdot 10^8$ Mpa. Useful and introductory details on viscoplastic theory and numerical implementation can be found in [6].

The experimental methodology involved two steps. First, we executed 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 [12]. Then, by means of the generated job data, we instantiated the CloudSim simulation toolkit. The experimental scenario consists of a datacenter with 10 physical resources with similar characteristics as the real machine where the SOGDE runs were performed. The characteristics are 4,008 MIPS (processing power), 4 GBytes (RAM), 400 GBytes (storage), 100 Mbps (bandwidth), and 4 cores. 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 more details about the real job data gathering and the CloudSim instantiation process the reader is referred to [3].

To evaluate the performance in the simulated Cloud we have modeled a dynamic Cloud scenario in which new users connect to the Cloud every 120 seconds, and require the creation of 10 VMs each in which their PSEs –a set of 100 jobs– run. The number of users who connect to the Cloud varies as $u = 10, 20, ..., 100$, and since each user executes one PSE –a set of $10 * 25$ jobs–, the total number of jobs to execute is increased as $n = 100 * u$ each time.

Moreover, our proposed SI algorithms are compared to another alternative scheduler, which is closely related to our work from an algorithmic standpoint. Particularly, we used the genetic algorithm (GA) proposed in [13], which has been previously evaluated via CloudSim as well. In this algorithm, the population structure is represented as the set of physical resources that compose a datacenter and each chromosome is an individual in the population that represents a part of the searching space. Each gene (field in a chromosome) is a physical resource in the Cloud, and the last field in this structure is the *fitness* field, which indicate the suitability of the hosts in each chromosome. In addition, a second 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 an exponential back-off retry strategy until it is able to serve all users. The number of retries enough to serve all users and create all requested VMs was 20. This scheduler has been implemented in this way to obtain the ideal values to which all its competitors should be compared against, however it is clearly impactical in real life situations.

In our experiments, the GA-specific parameters were set to the following values: chromosome size = 8, population size = 10 and number of iterations = 10. Moreover, for each one of our SI schedulers, we have supplied the ACO-specific parameters with values within the range of values studied in [10]: mutation rate = 0.6, decay rate = 0.1 and maximum steps = 8, and the PSO-specific parameter neighbourhood size = 6.

Experiments have been performed with the aim of measuring the trade-off between the number of serviced users by the Cloud and the total number of created VMs among all users. The basis for these metrics is that the more the number of serviced users, the higher the end-user throughput, and the greater the number of created VMs, the greater the parallelism and therefore the lower the flowtime [3]. The number of serviced users increases every time the scheduler successfully allocates any of the requested VMs. A user is considered "serviced" if the scheduler can create at least one VM for its required jobs. Based on these two metrics, we derived a *weighted metric*, by which the results obtained from the different algorithms have been normalized and weighted with numerical weights. The normalized values for each metric and each user group U connected to the Cloud are computed as $NormalValueU_{i=10,...,100} = 1 - \left(\frac{Max(valueU_i) - valueU_i}{Max(valueU_i) - Min(valueU_i)}\right)$, where *valueU* represents the obtained value for each one of the basic metrics –serviced users and created VMs– and for each user group connected to the Cloud, *Max(valueU)* and *Min(valueU)* are the maximum and minimum values, respectively, for each basic metric among all the algorithms –ACO, GA, PSO, 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 importance 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 both metrics are important and they are to be balanced, we have assigned the pair of weights (wSU, wVMs) equal to (0.50, 0.50). The higher the value of the weighted metric, the better the metric balance of an algorithm with respect its competitors.

Results in Table I show that the proposed ACO and PSO algorithms deliver the best balance with respect to the number of serviced users, and the total number of created VMs, for example with gains of $\frac{(0.11-0.07)}{0.11} = 0.57\%$ of ACO and PSO over GA in the most challenging scenario. Moreover, the weighted metric values illustrate that the performance between ACO and PSO is very close. In the experiments, both ACO and PSO achieve to serve a greater number of users than GA, but GA is the scheduler that creates the greatest number of VMs, excluding the ideal scheduler which is a factitious algorithm to show the best values to reach.

This article is a pre-print of the article "E. Pacini, C. Mateos and C. GarcÃ-a Garino: "SI-based Scheduling of Scientific Experiments on Clouds". IDAACS'2013. Vol. 2, pp. 699-704. Berlin, Germany. 2013. IEEE"
The published version is available at http://dx.doi.org/10.1109/IDAACS.2013.6663015

Table I. WEIGHTED METRIC RESULTS

Users connected to the Cloud	ACO	GA	PSO	Ideal
10	0.24	0.03	0.29	1
20	0.23	0.04	0.23	1
30	0.19	0.07	0.19	1
40	0.17	0.07	0.16	1
50	0.15	0.07	0.14	1
60	0.13	0.08	0.13	1
70	0.13	0.07	0.12	1
80	0.12	0.07	0.12	1
90	0.12	0.08	0.11	1
100	0.11	0.07	0.11	1

As suggested, a greater throughput in terms of serviced users means more fairness in resource sharing, and a greater number of created VMs involves greater parallelism, and therefore, a greater rate of jobs per unit time can be executed. The results obtained so far are encouraging because they indicate that using SI techniques helps in better dealing with the throughput-response time trade-off in scientific Clouds.

VI. CONCLUSIONS AND FUTURE WORKS

PSEs is a type of simulation popular among scientists that involves running a large number of independent jobs, each representing a different set of variable values in a model. Moreover, these jobs must be efficiently processed in the different computing resources of a parallel environment such as the ones offered by a Cloud. Then, job scheduling plays a fundamental role.

Recently, SI-inspired algorithms have received increasing attention in the Cloud research community for dealing with VM and job scheduling. SI refers to the collective behaviour that emerges from a swarm of social insects. Social insect colonies collectively solve complex problems through intelligent emergent behavior.

Existing efforts in this line do not address in general dynamic (or *online*) environments where multiple users connect to scientific Clouds to execute their PSEs and to the best of our knowledge, no effort aimed at balancing the number of serviced users in a Cloud and the number of created VMs exists. Indeed, the greater the number of serviced users, the better the throughput, and the more the created VMs, the higher the achieved parallelism. More parallelism means executing jobs faster, and hence a more agile human processing of PSE job results. More serviced users means a more fair assignment of Cloud computing resources. Simulated experiments performed with the well-established CloudSim toolkit and real PSE job data show that our ACO and PSO schedulers provide a good balance to the trade-off between the number of serviced users and the total number of created VMs. As alternative schedulers, we used Genetic Algorithms and an ideal assignment.

In the near future, we plan to materialize the resulting schedulers on top of a real Cloud platform, such as Emotive Cloud (http://www.emotivecloud.net/) or OpenNebula (http://opennebula.org/), which are designed for extensibility. Second, we will consider other Cloud scenarios, for example, with heterogeneous machines. We will also evaluate how the variation of the parameters inherent to each SI algorithm (e.g., the ones that control ant or particle behaviour) affect their performance. Finally, we will also evaluate network consumption.

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] 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.

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

[5] 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.), vol. I, ch. 16, pp. 410–447, IGI Global, 2012. ISBN13: 9781466625181.

[6] 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, July 2013. ISSN: 0377-0427.

[7] 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.

[8] M. Dorigo, *Optimization, Learning and Natural Algorithms*. Phdthesis, Politecnico di Milano, Italy, Milano, Italy, 1992.

[9] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE Computer Society, 1995.

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

[11] C. Careglio, D. Monge, E. Pacini, C. Mateos, A. Mirasso, and C. García Garino, "Sensibilidad de resultados del ensayo de tracción simple frente a diferentes tamaños y tipos de imperfecciones," in *Mecánica Computacional* (M. G. E. Dvorkin and M. Storti, eds.), vol. XXIX, pp. 4181–4197, AMCA, 2010. ISSN 1666-6070.

[12] 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.

[13] 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)*, DASC 11, (Washington, DC, USA), pp. 598–604, IEEE Computer Socienty, 12-14 December 2011. ISBN: 978-0-7695-4612-4.