Journal of Network and Systems Management
https://doi.org/10.1007/s10922-018-9481-0

A Bio-inspired Datacenter Selection Scheduler for Federated Clouds and Its Application to Frost Prediction

Elina Pacini[1,2] · **Lucas Iacono**[1,2] · **Cristian Mateos**[3] · **Carlos García Garino**[1]

Received: 14 August 2017 / Revised: 1 November 2018 / Accepted: 8 November 2018

Abstract

Frost is an agro-meteorological event which causes both damage in crops and important economic losses, therefore frost prediction applications (FPA) are very important to help farmers to mitigate possible damages. FPA involves the execution of many CPU-intensive jobs. This work focuses on efficiently running FPAs in paid federated Clouds, where custom virtual machines (VM) are launched in appropriate resources belonging to different providers. The goal of this work is to minimize both the makespan and monetary cost. We follow a federated Cloud model where scheduling is performed at three levels. First, at the *broker level*, a datacenter is selected taking into account certain criteria established by the user, such as lower costs or lower latencies. Second, at the *infrastructure level*, a specialized scheduler is responsible for mapping VMs to datacenter hosts. Finally, at the *VM level*, jobs are assigned for execution into the preallocated VMs. Our proposal mainly contributes to implementing bio-inspired strategies at two levels. Specifically, two broker-level schedulers based on Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO), which aim to select the datacenters taking into account the network latencies, monetary cost and the availability of computational resources in datacenters, are implemented. Then, VMs are allocated in the physical machines of that datacenter by another intra-datacenter scheduler also based on ACO and PSO. Performed experiments show that our bio-inspired scheduler succeed in reducing both the makespan and the monetary cost with average gains of around 50% compared to genetic algorithms.

Keywords Scientific computing · Frost prediction applications · Cloud computing · Scheduling · Ant colony optimization · Particle Swarm optimization · Genetic algorithms

✉ Elina Pacini
epacini@uncu.edu.ar

Extended author information available on the last page of the article

Published online: 22 November 2018

1 Introduction

Scientific computing is a field that applies computer science to solve typical scientific problems. Scientific computing is usually associated with large-scale computer modeling and simulation, and requires a sheer number of computing resources to quickly deliver results for ever-growing problem sizes. A concrete example of scientific applications are Precision Agriculture Applications, particularly Frost Prediction Applications (FPA). FPAs, which are composed of multiple numerical regressions [1] and machine learning techniques [2], are executed using data collected on-field by different instruments like thermometers, weather stations or Wireless Sensor Networks (WSNs) [3, 4]. These instruments generate considerable amounts of data, therefore somewhat powerful computational resources are required for storing and processing data. Specifically, frost prediction must be performed for each of the farms in which it is necessary to know whether a frost can be produced or not, i.e., that the FPA is executed with data collected from each one of the farms. Accordingly, to perform a frost prediction in an entire region (composed of a large number of farms of different sizes and with different number of sensor nodes), it is necessary to run the FPA on different machines in parallel and collecting the results. Therefore, from a computational perspective, running FPAs involves managing many independent jobs with a master-worker structure. Indeed, users relying on FPAs need a computing environment that delivers large amounts of computational power in order to obtain the predictions in the shortest possible time.

Cloud Computing [5, 6] brings a technological solution to the problem of frost prediction due to their reliability, availability and resources scalability. From a technical standpoint, Cloud permits the acquisition of fully-configured infrastructures through virtualization technologies [5], i.e., different types of Virtual Machine (VM) instances provide a wide spectrum of hardware and software configurations under a pay-per-use scheme. Usually, VM prices vary according to the acquired instance type and the pricing model of the Cloud provider. Concretely, a Cloud provides some measure of reliability and scalability regarding computing infrastructure both for data storing and FPA processing. Moreover, the economic costs of Cloud resources for scientific computing are lower compared with that of traditional in-house clusters [7]. However, since in single-datacenter Clouds resource availability might be limited, the option of obtaining extra resources from an arrangement of Cloud providers has appeared recently as an appealing solution [8, 9]. This ability to exploit resources from multiple Cloud providers is also called *federating Clouds* [10].

For executing resource intensive applications in general, and FPA in particular, when using federated Clouds it is necessary to properly manage physical resources, since they are part of geographically distributed datacenters. Therefore, for the efficient execution of jobs, scheduling should be performed at three abstraction levels [11]. Firstly, at the *broker* level, scheduling strategies are used for selecting datacenters taking into account issues such as network interconnections or monetary cost of allocating VMs on hosts that compose them. The broker generates an execution plan based on requirement criteria provided by the user and the offerings of the available Cloud providers. At this level a broker acts as an intermediary between the users and the Cloud providers. The broker utilizes broker strategies to route user requests to the most appropriate datacenter. Therefore, the optimal response time of a particular request and the efficient utilization of the datacenters are governed through datacenter selection policies [12]. Secondly, at the *infrastructure* level, VM scheduling algorithms are implemented to schedule the VM requests to the physical machines of a particular datacenter taking into account the requirement fulfilled with the requested resources (i.e. RAM, Memory, Bandwidth etc). Lastly, at the *VM* level, by using job scheduling techniques, jobs are assigned for execution into allocated virtual resources, which were allocated at the previous level. However, scheduling is an NP-Complete [13] problem and therefore it is not algorithmically trivial. Moreover, the fact that federated Cloud scheduling spans these three levels makes the problem even more challenging.

Bio-inspired strategies such as Swarm Intelligence (SI) metaheuristics have been suggested to solve combinatorial optimization problems—such as broker/VM/job scheduling—by simulating the collective behavior of social insects swarms [14]. Inspired by these capabilities, researchers have proposed algorithms or theories for combinatorial optimization problems, where the most popular SI-based strategies are Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). Moreover, job scheduling in federated Clouds is also a combinatorial optimization problem, and schedulers in this line that exploit SI have been proposed [15].

In this paper, we propose two broker schedulers based on ACO and PSO for the efficient execution of FPAs in federated Clouds. FPAs are applications of bag-of-task type, therefore, the execution of each FPA-job corresponds to a prediction in a given farm, and the greater the number of predictions to be made, the greater the number of jobs to be executed. The goal is to select the most suitable datacenter taking into account the network latencies, monetary cost and the availability of resources of each datacenter. The network latencies *among* datacenters negatively affect the response time—makespan—delivered to the user [11]. The less the network latency, the lower the influence on makespan. Moreover, when more VMs can be allocated in datacenters to which latency is lower, more physical resources can be exploited, and hence job execution time decreases. When a frost prediction is performed, reducing the makespan is very important because the earlier the defense methods start up in the presence of frost, the less the impact in the economic losses due to the loss of crops. Then, once our broker-level scheduler has selected a datacenter to execute jobs, VMs are allocated in the physical machines of that datacenter by another intra-datacenter schedulers based on ACO and PSO previously studied in [16]. To allocate the VMs into hosts, this scheduler must make a different number of "queries" (network messages) to hosts to determine their availability upon each VM allocation attempt. The number of queries to be performed by ACO and PSO and the latencies of datacenters also influence the makespan to the user. Besides, the lower the makespan to the user, the lower the monetary cost when the resources are used in a pay-per-hour basis. Finally, at the VM level, jobs are assigned to the preallocated VMs by using FIFO, as in [16]. Briefly, in this paper we put emphasis on evaluating how SI decisions taken at the broker and infrastructure levels influence both the makespan and the monetary cost.

Specifically, we formulate our problem as minimizing the *makespan*, i.e., the total execution time of all jobs, while also minimizing the *monetary cost* of a set of jobs. Our approach differs from those presented in literature since none has considered SI-based strategies at two levels (see next Section for a discussion on related efforts). Moreover, compared to previous works of our own, it is worth noting that:

- In [16] we have presented an ACO-based scheduler focused on the infrastructure level only. However, this scheduler operates at two out of the three levels mentioned above, i.e. it is designed for Clouds composed of a **single** datacenter and as such does not target **federated** Clouds.
- In [17, 18] we extended the previous scheduler to operate in federated Clouds and we included the PSO technique. However, SI was again applied at the infrastructure level and not at the broker level, where straightforward decision heuristics were used instead. It is important to mention, moreover, that in such works we implemented very simple strategies for selection of datacenters and the algorithms were formulated for a single objective problem. Concretely, the strategies took into account only latencies and the datacenters were selected via three policies based on Lowest-Latency-Time-First, First-Latency-Time-First, and Latency-Time-In-Round in order to reduce the response time. In this paper, we formulate the problem as a multiobjective one by including the monetary cost that had not been considered so far. Reformulating the problem as multiobjective makes it necessary to take optimal decisions in the presence of trade-offs between two or more conflicting objectives, and in the context of this work this means makespan and monetary cost. This makes the problem more computationally challenging and it is difficult to find optimal solutions.
- On the other hand, we have included a new real application (FPA unlike Parameter Sweep Experiments (PSE) as in previous works of our own), whose jobs are much more CPU-intensive and also have a greater deviation from each other with respect to their execution times. The algorithms developed in this work were developed taking into account the intrinsic characteristics of this type of application, providing a response to the users in the shortest possible time and with lower cost.

In summary, in this work we essentially extend [18] in order to also consider the monetary cost of paid Clouds and include a new broker-level scheduler based on ACO and PSO. Therefore, the objective of this paper is to illustrate an enhanced multiobjective broker policy that selects datacenters based on the network latency, monetary cost, and resources availability to ensure efficient and reliable execution of FPA-jobs over geographically distributed datacenters.

Experimenting in Cloud environments is subject to uncontrollable factors like network congestion and servers varying workloads. Measuring the performance of Internet based applications using real Cloud platforms is cumbersome [19, 20]. Therefore, simulation-based approaches have been adopted in order to avoid such issue under a stable and controllable environment. A popular toolkit for modeling and simulating Cloud Computing systems is CloudSim [21]. CloudSim is a broadly accepted testbed and particularly it actually represents the most common approach to experimentation in the area of meta-heuristics, such as SI, for resource scheduling in Grid/Cloud environments. This observation has been documented in a survey [15] written by some of the authors of this work.

In this paper, to set the basis for comparison and evaluate the overall performance, we used our already proposed SI-based schedulers at the broker and infrastructure level [16, 22], and FIFO at the VM level, in combination with an alternative scheduler—at the broker and infrastructure levels—based on Genetic Algorithms (GA) [23]. This alternative scheduler was also combined with a FIFO based policy at the VM level. Simulated experiments performed with job execution data extracted from a real-world FPA, suggest that the use of SI schedulers at two levels—broker and infrastructure levels—, deliver competitive performance in terms of makespan and monetary cost. Specifically, through the use of our bio-inspired scheduler we have obtained average gains of around a 50% with respect to GA.

The rest of the paper is organized as follows. Section 2 surveys and analyses relevant related works. Section 3 presents our proposal and the involved techniques at each level. Then, in Sect. 4 we present detailed experiments that show the viability of the approach via a real FPA. Finally, Sect. 5 concludes the paper and presents future extensions.

2 Related Work

Studying SI techniques, specially ACO and PSO [24–29], have been the focus of many research studies for solving combinatorial optimization problems in the last ten years. As shown in recent surveys and works [30–32], SI-based techniques have been increasingly applied to distributed job scheduling in Clouds in a variety of application domains. However, to the best of our knowledge, there are not efforts covering the three scheduling levels and where the authors also consider the use of SI at more than one level. The use of SI at more than one scheduling level is beneficial because it helps to narrow down the search space to be explored at each scheduling level.

Specifically, we address the scheduling of precision agriculture applications in federated Clouds in order to minimize both the makespan and the monetary cost of a set of jobs considering the influence of the network interconnections and latencies among heterogeneous datacenters. First, our approach differs from those presented in the literature since the existing works have not considered SI-based strategies at more than one scheduling level as we do in this paper. In previous works of our own [11, 18], we also proposed a scheduler for federated Clouds that exploits SI and the concept of job priorities for Parameter Sweep Experiments (PSE). However, it is important to mention that in such works, SI was only implemented at the infrastructure level. At the broker level, datacenters were selected according to their network latencies through three simple policies called Lowest-Latency-Time- First (LLTF), First-Latency-Time-First (FLTF), and Latency-Time-In-Round (LTIR). In addition, another major distinction is that in [11, 18] we considered dedicated datacenters to execute the application and minimizing monetary cost was not taken into account. However, since in public Clouds users must pay for the use of resources, and moreover, costs depend on the instance type and Cloud provider, it is also important to reduce the monetary costs associated when FPAs are executed in federated Clouds. In this work, we extend the above mentioned scheduler at the broker level in order to also consider the monetary cost and include ACO and PSO based strategies at the broker level. A Cloud broker provides an interoperability layer on top of the various Cloud provider interfaces.

Second, works found in the literature and also summarized in Table 1 are mainly focused on one Cloud level and do not consider advanced techniques for the three scheduling levels as we propose in this work. As can be seen in Table 1, among these works, we can first mention the two approaches proposed in [33, 34]. In [33] the authors have proposed a layered federated Cloud management architecture that incorporates the concepts of meta-brokering, Cloud brokers and automated, on-demand service deployment. The meta-brokering[1] component allows the system to interconnect the various Cloud brokers[2] available in the system. The broker component is responsible for managing the VMs instances hosted on a specific infrastructure as a service provider. In order to fast track the VMs instantiation, this architecture uses an automatic service deployment component that is capable of optimizing its delivery by decomposing and replicating it among the various cloud infrastructures. In this work, the authors proposed a federated Cloud solution that acts as an entry point to Cloud federations and deals with broker issues. However, [33] has not considered the use of SI-based strategies, and moreover, the authors have *exemplified* (not evaluated) the interaction of the various components of their proposed architecture through a low-level use case and not with a real application such as the FPA proposed in this paper. Then, in [34] the authors have presented another initial approach to the federated Clouds but considering a layered model based on the Infrastructure, Platform and Software as a Service models. In this work the benefits of decoupling the different layers have been discussed so that the execution of an application can be supported by diverse providers implementing different parts of the layer functionality. The authors have also introduced a motivational scenario to illustrate this layered model based on a Weather Research and Forecasting (WRF) application for domain experts which accepts high level parameters relating to user requirements such as cost or execution time. Besides, in this work it is shown how these requirements are either used in the negotiation process, where negotiation is constrained to well-defined sets of parameters among different providers, or transformed to new arguments to lower levels in the Cloud stack by using prediction models and inter-layer translation mechanisms. However, even though the authors have discussed different brokering strategies for providers to assign parts of the execution application to other partners while enforcing user policies, none of the strategies have been implemented yet. Besides, the authors in [34] have not obtained

[1] Analogous to the broker level in our paper.

[2] Analogous to the infrastructure level in our paper.

Table 1 Summary of the analyzed approaches

Paper	Scheduling level	Algorithms	Objective	Cloud Environment	Constraint	Dedicated Datacenter
Our proposal	Broker, infrastructure, VM	ACO, PSO	Makespan, monetary cost	Federated cloud	Resources availability	Not dedicated
[18]	Broker, infrastructure, VM	LLTF, FLTF, LTIR ad-hoc policies	Makespan	Federated cloud	Without constraint	Dedicated
[11]	Broker, infrastructure, VM	LLTF, FLTF, LTIR ad-hoc policies	Makespan, flowtime	Federated cloud	Without constraint	Dedicated
[33]	Broker	N/A	N/A	Federated cloud	Without constraint	Dedicated
[34]	Broker	N/A	N/A	Federated cloud	Without constraint	Dedicated
[12]	Broker	VSBRP	Makespan, load balancing	Federated cloud	without constraint	Not dedicated
[35]	Broker, infrastructure	Integer programming based algorithm	Monetary cost, load balancing	Federated cloud	Cloud to deploy the VMs, load balancing, hardware configuration of individual VMs, budget	Dedicated
[36]	Broker	MO-GA, greedy heuristic	Energy consumption	Federated cloud	Without constraint	Not dedicated
[37]	Broker	Cost optimization policy/performance optimization policy	Monetary cost, makespan	Federated cloud	Budget, performance, instance types	Not dedicated
[23]	Broker, infrastructure	Dijkstra, GA	Monetary cost	Federated Cloud	Without constraint	Dedicated
[38]	Broker	Hybrid ACO-PSO	Monetary cost	Federated cloud	Without constraint	Dedicated
[39]	Broker	MOPSO	Response time of user request, economic profit of provider, energy consumption	IoT-oriented federated Cloud	Without constraint	Dedicated
[40]	Infrastructure	ACO	Energy consumption	Single datacenter	Without constraint	Dedicated

Table 1 (continued)

Paper	Scheduling level	Algorithms	Objective	Cloud Environment	Constraint	Dedicated Datacenter
[41]	Infrastructure	ACO	Total resource utilization, energy consumption	Single datacenter	Without constraint	Dedicated
[42]	Infrastructure	PSO	Makespan, energy consumption	Single datacenter	Without constraint	Dedicated
[43]	VM	Honey bee	Makespan, load balancing	Single datacenter	Without constraint	Dedicated
[44]	VM	ACO	Makespan, load balancing	Federated cloud	Without constraint	Not dedicated
[45]	VM	PSO	Economic profit of a provider	Hybrid cloud	Task deadlines	Dedicated
[46]	N/A	Hybrid PSO-FGA	Energy consumption	Federated cluster	Without constraint	Dedicated

results from the execution of the application with their proposed model as we do in this work, but they merely exemplify how the process works.

Then, a number of studies for Cloud broker-level strategies are discussed in [12, 35–37]. In particular, in [12] the authors proposed a heuristic-based technique called Variable Service Broker Routing Policy (VSBRP) that aims to achieve minimum makespan and considers the network latency, bandwidth and the size of the job. The proposed service broker policy has been proposed in order to reduce the overloading of the datacenters by redirecting the user requests to the next datacenter that yields better response and makespan. Moreover, in [35], the authors proposed a Cloud brokering approach that restricts the deployment of VMs across multiple heterogeneous datacenters according to some placement constraints (e.g., Clouds to deploy the VMs) defined by the user. Users can also steer the VM allocation by specifying maximum budget and minimum performance, as well as constraints with respect to load balancing, hardware configuration of individual VMs and budget. The implemented algorithms are based on integer programming formulations which enable price-performance placement trade-offs. Then, in [36, 37] the authors proposed different strategies at the broker level to optimize the scheduling of jobs across multiple providers. In the work presented in [36] the authors proposed a multi-objective genetic algorithm (MO-GA) for broker scheduling with the aim to optimize three objectives namely, energy consumption, CO2 emission, and the generated profit of a geographically distributed datacenters. On the other hand, in [37] the scheduler performs an optimal deployment of the jobs among datacenters optimizing a particular cost function based on different optimization criteria (e.g., monetary cost optimization or performance optimization) and different user constraints (e.g., budget, performance, instances types).

Some other works that deserve special attention are [23, 38, 39]. In [23], the authors used at the broker level the Dijkstra algorithm [47] to select the datacenter with the lowest monetary cost, and a GA for allocating VMs at the infrastructure level. Then, in [38], a broker federation strategy based on a hybrid ACO-PSO algorithm was proposed in order to solve the nonlinear integer programming problem and obtain the price benefits of reserved VMs. The broker is formed through a dynamic pricing method for geo-distributed datacenters. In this work the proposed strategy takes advantage of the price gap between on-demand and reserved VMs and the cost saving achieved is significant through the broker federation. Although in [23] the authors target both the broker and the infrastructure levels, the goal was to reduce the monetary costs without considering the completion time. Likewise, in [38] only the monetary was considered cost without paying attention to the completion time. For scientific applications in general, and FPAs in particular, the completion time is very important [22], since it allows users to accelerate result processing. The faster the frosts prediction results are known, the faster the defense methods can be triggered to prevent damages if frosts occur. On the other hand, in [39] a Cloud brokering as a multi-objective optimization problem was presented to find the appropriate connections between users and providers. This work was proposed for a Cloud brokering problem in an Internet of Things (IoT) Cloud system. For this, a multi-objective PSO (MOPSO) was proposed for maximizing the economic profit of the broker while minimizing the response time of requests from users and the energy consumption of service providers. However, although in these works [23, 38, 39] the authors have proposed the use of SI-based strategies in federated Clouds, in [38, 39], the

SI-based algorithms were implemented at the broker level without considering how the VMs are allocated at the infrastructure level, and without taking into account network issues, such as the inter-datacenter network latencies as we consider in our work. On the other hand, none of these works have considered the makespan which is a very important metric to consider when we work with scientific applications where the response time to the user is crucial. Finally, unlike these works, our work is subject to resource availability constraint in each data center, which is also important when working in non-dedicated Clouds.

With respect to works which address the scheduling problem at the infrastructure level—intra-datacenter—using SI-based strategies as we propose in this work, few efforts have been found [15]. Among them we can mention [40, 41]. In [40] the authors proposed a VM scheduler based on the ACO to perform the dynamical placement of VMs according to the current load on physical machines. The goal of this work was to minimize the energy consumption in a Cloud composed of a single datacenter. Then, in [41] the authors proposed a multi-objective ACO for the VM allocation problem. The goal of this work was to obtain a set of solutions that simultaneously maximize total resource utilization and minimize energy consumption. On the other hand, in the work [42] the authors proposed a PSO algorithm whose purpose is to map efficiently a set of VM instances in a set of physical machines while reducing both the energy consumption and makespan. This algorithm makes the best possible use of the power saving states of idle physical machines and instantaneous workload on the operational physical machines. However, although in these works SI-based algorithms at the infrastructure level have been used, the schedulers were proposed for Clouds composed of a single datacenter and not for federated Clouds. Besides, these works do not pay attention to monetary cost, and only in [42] the authors considered the makespan, achieving competitive performance as evidenced by experiments performed via CloudSim [21], which is also used in this paper.

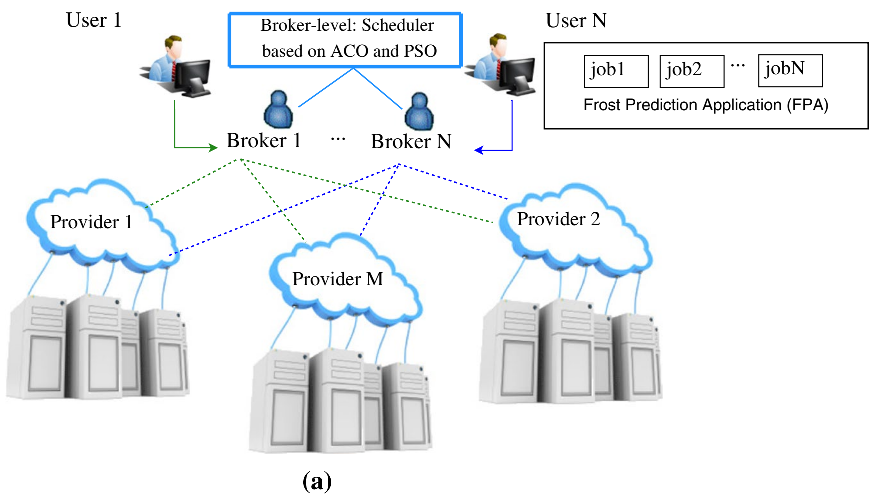
Finally, there are some works that apply SI at the VM level, among them we can mention [43–46]. In [43] a honey bee inspired load balancing algorithm was proposed. The goal of this algorithm is to achieve well-balanced load across VMs for maximizing the throughput. Throughput is the number of jobs that can be executed over a long period of time. The proposed algorithm also balances the priorities of jobs on the machines in such a way that the amount of waiting time of the jobs in the queue is minimal. Moreover, in [44] the authors proposed an ACO scheduler to perform efficient distribution of jobs by finding the best VMs to execute jobs. The aim of this work was minimizing the makespan and improve load balancing in the VMs. Moreover, in [45], a PSO based scheduling algorithm is proposed for hybrid Clouds. An hybrid Cloud is a Cloud environment which uses a mix of on-premises, private Cloud and third-party, public Cloud services with orchestration between the two platforms. Authors claim that by allowing workloads to move between private and public Clouds as computing needs and costs change, hybrid Clouds give a greater flexibility. In this work a private Cloud is able to outsource its jobs to other Clouds when its local VMs are not enough to satisfy users requirements. The scheduling problem in the proposed hybrid Cloud model has been formulated as a kind of deadline constrained job scheduling problem, in which each job has an strict deadline and the objective is to maximize the profit of a provider under the premise of guaranteeing each job deadline constraint. Another work is [46], where the authors have presented a multi-objective algorithm combining PSO and a GA with a Fuzzy crossover operator (MPSO-FGA) for solving the scheduling of parallel applications in Federated cluster environments with the aim of to minimize both the overall energy consumption and the makespan for a whole workload. The algorithm takes advantage of a weighted blacklist for effective representation of the computational resources availability, settled considering resources heterogeneity, communication resources contention and application requirements. It is important to mention that the work in [43] was proposed for Clouds composed of a single datacenter where the notion of broker level does not apply. Moreover, in [44–46], the authors focus on assigning jobs assuming the existence of pre-allocated VMs, i.e., the SI-based scheduling algorithms were applied at the VM level and not at the broker or the infrastructure level. It is important to mention that all these works [43–46] are complementary to the one proposed in this paper because so far we have not explored SI at the VM level.

It is worth noting that, from the related works found, most of the works which consider SI for federated Clouds have been proposed taking into account only one of the scheduling levels without considering metrics such as makespan and monetary cost, rendering difficult their applicability to execute FPAs in federated Cloud environments. For FPAs, the use of Clouds can be very beneficial due to the fact that frost fairly accurate frost predictions can be obtained from any geographical location 24 hours a day and 365 days a year. Such predictions are useful so that farmers are on alert to the event and can take precautions in their farms before a frost actually occurs. The next Section explains our bio-inspired approach in detail, which considers the three scheduling levels and besides, takes into account the issues of latencies, monetary cost and resources availability of datacenters.

3 Approach Overview

This paper focuses on providing users with a Cloud scheduler that supports the efficient execution of CPU-intensive applications and particularly Frost Prediction Applications (FPA). The main characteristic of FPAs is the need to obtain the results—predictions—in the shortest possible time in order to mitigate damage caused by frosts. Moreover, since resource usage incurs direct monetary costs, they should also be minimized. Therefore, the goal of our scheduler is to achieve a balance between the monetary costs and the makespan of a set of FPA-jobs when the jobs are executed into a federated Cloud. Makespan is the period of time between a user makes a request to the Cloud and he/she gets the answer (including the impact of the inter-datacenter latencies), i.e., the period of time in which a user requests a number of VMs to execute its FPAs, and the time in which all the FPA jobs finish their execution. Conceptually, an FPA is a set of $N = 1, 2, \dots, n$ independent jobs, where each job corresponds to a frost prediction in a farm within the same region. The jobs are executed on m Cloud machines. The makespan, of a job j in schedule S is denoted $C_j(S)$ and hence the makespan is $C_{max}(S) = max_j C_j(S)$. Achieving a low makespan is important since it means starting up defense methods as soon as possible and thus minimizing frost damage.

For running applications in federated Clouds, resources should be scheduled at three levels as shown in Fig. 1. A broker is created for each user that connects to


User 1
Broker-level: Scheduler based on ACO and PSO
User N
job1
job2
jobN
Frost Prediction Application (FPA)
Broker 1
Broker N
Provider 1
Provider 2
Provider M
(a)

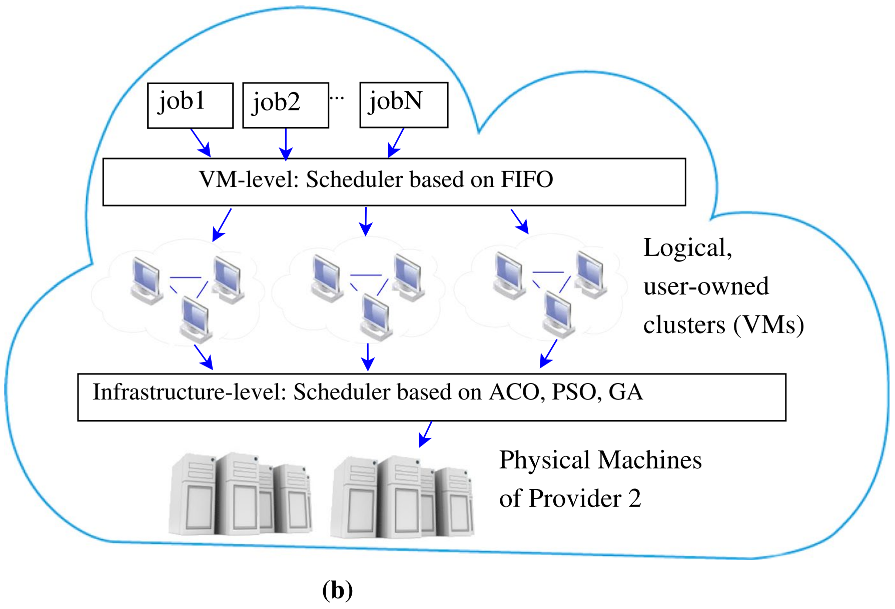

job1
job2
jobN
VM-level: Scheduler based on FIFO
Logical, user-owned clusters (VMs)
Infrastructure-level: Scheduler based on ACO, PSO, GA
Physical Machines of Provider 2
(b)

Fig. 1 Federated cloud: overview. **a** Federated cloud, **b** scheduling intra-datacenter

the Cloud. Each broker knows who are the providers that are part of the federation. The relation of each broker is colored with green and blue dotted lines. In addition, Fig. 1a illustrates how jobs sent by *User N* are executed in the datacenter of *Cloud Provider 2*. Then, Fig. 1b shows the intra-datacenter scheduling activities—inside *Cloud Provider 2*—, i.e., at the infrastructure level and the VM level.

The proposed scheduler proceeds as follows. Firstly, at the broker level, a datacenter D_{br_i} is selected via an SI-based scheduler (for the purposes of this paper, we consider ACO and PSO schedulers). In order to achieve a balance between the monetary cost and the makespan, at this level, the selected datacenter will be the one which provides the best balance between the monetary cost of the datacenter and the lowest communication latency to a broker. The less the network latency, the lower the influence on makespan to the user. For them, two weights have been assigned to the two individual metrics, i.e., a weight for the monetary cost (weightMC) and a weight for the communication latency (weightCL). Then, we assign the weights (weightMC, weightCL) = (0.5, 0.5) with the aim of balancing these two basic metrics. In addition, the selected datacenter must meet a certain percentage of available resources. In this paper, we consider non-dedicated datacenters, i.e., they might already have allocated VMs from other users or applications. At the infrastructure level (Fig. 1b), via another bio-inspired VM schedulers, user VMs are allocated in the physical resources (i.e., hosts) belonging to the selected datacenter at the broker level. Only, when there are no available hosts in the datacenter to allocate the VMs, a new datacenter $D_{br_j}(j \neq i)$ is selected at the broker level. It is important to mention, moreover, that this two-step selection serves the purpose of delimiting the elements to be explored in the search space by the bio-inspired algorithms (datacenters in the first step, physical hosts in the second one). In addition, solving the problem of the broker and infrastructure levels in a single step in federated Clouds would make the implementation of a bio-inspired algorithm very expensive due to the coordination messaging through the network and the influence of latencies. Finally, at the VM level, jobs are assigned to the preallocated VMs through a FIFO policy. Next subsections explain in more detail each the sub-schedulers employed at each level. Basic theory about SI-based strategies, which will help to understand the ideas implemented in the proposed approach, can be found in Appendix A.

3.1 Broker level Scheduler

Once a user makes a request to a broker, the scheduler at the broker level is executed to select the first datacenter to allocate the VMs, which are managed by the scheduler employed at the infrastructure level. Furthermore, the scheduler at this level can decide to deploy the VMs in a remote datacenter when there are insufficient physical resources in the datacenter where the VM creation was issued. At present, the policies studied at this level for the purposes of this paper are explained below.

3.1.1 Scheduler Based on ACO

Each time a user requests to a broker a number of VMs in which execute his/her FPA-jobs an *ant* is initialized for finding the most suitable datacenter (Algorithm 1). To this end, three parameters are initialized. A *step* parameter keeps track of the number of steps carried out by an ant, *maxStep* is equals to a predefined number of steps (i.e., the completion criterion of the ant work), and *vmPercentage* is a user-defined percentage value used by the ant to know whether a datacenter has enough resources to allocate at least these percentage of VMs. For them, each datacenter keeps track of the resources availability in terms of total available processing power, bandwidth , memory and storage availability. Resource availability is updated every time a VM is allocated/deallocated in/from a host in a datacenter and it is then used by the ant to estimate the percentage of VMs to allocate in a datacenter.

Algorithm 1 ACO-based datacenter selection algorithm

```
Procedure ACOBrokerScheduler(user, dcList, vmList)
Begin
   initializeLocalTable()
   initialize(maxSteps)
   suitableDatacenters=getSuitableDatacenters(dcList)
   ant=new Ant(user,suitableDatacenters)
   ant.initialize(step, vmPercentage)
   repeat
      ant.AntBroker(suitableDatacenters, vmList, maxSteps)
   until ant.isFinish()
   selectedDatacenter=dcList.get(ant.getDatacenter())
   VmScheduler(selectedDatacenter ,vmList)
   if(vmList.size()>0)
       ACOBrokerScheduler(ant.getUser(),dcList, vmList)
   end if
End
```

When an ant is created, a list of suitable datacenters in which the VMs can be allocated is built (*getSuitableDatacenters(dcList)*). A datacenter is suitable if it has hosts with processing power, storage capacity and memory greater than or equal to that of required by the VMs. The ant is randomly initialized in one of the obtained datacenters. A local table containing information both of the weighted metric, i.e., $weightedMetric = weightMC * MonetaryCost + weightCL * CommunicationLatency$, and an estimate of the percentage of VMs that can be allocated in each datacenter is created (*InitializeLocalTable()*) by the first ant which visits the datacenter. Monetary Cost is the hourly processing cost of a datacenter's resources.

Algorithm 2 ACO-specific logic

```
Procedure AntBroker(suitableDatacenters, vmList, maxSteps)
Begin
   dc=getInitialDatacenter(suitableDatacenters)
   While (step < maxSteps) do
      resourcesAvailability=getResourcesAvailability(dc)
      vmPercentageEstimation= calculateVMsPercentage(resourcesAvailability, vmList)
      weightedMetric=getWeightedMetric(dc)
      localTable.update(vmPercentageEstimation, weightedMetric)
      if (random() < searchRate) then
         nextDatacenter=randomlyChooseNextStep()
      else
         nextDatacenter=chooseNextStep()
      end if
      searchRate=searchRate-decreaseRate
      step=step+1
      moveTo(nextDatacenter)
   end while
End
```

In each iteration, the ant estimates the percentage of VMs that can be allocated in the datacenter which is visiting through the (*calculateVMsPercentage (resourcesAvailability, vmList)*) method and calculate the weighted metric by collecting the monetary cost and latency information of the datacenter through *getWeightedMetric(dc)*. This datacenter information collected by the ant is added to its private information table—*localTable*—, which is maintained in each datacenter through (*localTable.update(vmPercentageEstimation, weightedMetric)*). The percentage of VMs that can be allocated in a datacenter is:

$$vmPercentageEstimation = \frac{resourcesAvailability/hostProcessingPower}{vmListSize} * 100,$$

where *resourcesAvailability* is the total available processing power in a datacenter in MIPS --among all host which in addition have enough memory, bandwidth and storage-- , *hostProcessingPower* is the processing power in MIPS of its hosts, and *vmListSize* is the number of VMs not allocated by the ant in any datacenter so far.

The information table contains both the estimation percentage of VMs that can be allocated and the weighted metric of the datacenter the ant is visiting. Besides, the ant adds to the table the weighted metric of other datacenters, which were added to the table when the ant visited these datacenters. For them, the ant performs a predefined number of steps, i.e., *maxSteps*, looking for the datacenter that allows both allocating at least the predefined percentage of VMs (*vmPercentage* parameter) as well as achieving the best balance between monetary cost of their resources and communication latency intra-datacenter. At this level it is important to select datacenters with a lower latency because later, at the infrastructure level, latencies have a great influence when creating the VMs, and therefore, in the final makespan to the user.

Every time an ant moves from one datacenter to another it has two choices: moving to a random datacenter using a constant probability or *searchRate* through the *randomlyChooseNextStep()* method, or using the information table of the current datacenter (through *chooseNextStep()* method). The *searchRate* decreases with a *decreaseRate* factor as time passes, thus, the ant will be less dependent on random choice. Every time an ant visits a datacenter, it updates the datacenter information table with the information of other datacenters, but at the same time the ant collects the information already provided by the table of that datacenter, if any. The information table acts as a pheromone trail that an ant leaves while it is moving in order to choose better paths rather than wandering randomly in the federated Cloud. Entries of each information table are the datacenters that the ant has visited on their way to select the most suitable datacenter together with their weighted metric and percentage of VMs to allocate.

When the ant reads the table in a datacenter, it chooses the entry with the lowest weighted metric, which also meets the predefined percentage of VMs to allocate. If the weighted metric of the visited datacenter is smaller than any other datacenter in the table and in addition it meets the percentage of VMs defined, the ant chooses the datacenter with the smallest weighted metric. On the other hand, if the weighted metric of the visited datacenter is equal to any datacenter in the table, the ant selects the datacenter which has lower latency, that is, the ant prioritizes the makespan of the FPA. This process is repeated until *step* = *maxStep*s (finishing criterion). Finally, the ant invokes the infrastructure-level scheduler through *VmScheduler(selectedDatacenter,vmList)* method with the selected datacenter and the list of VMs to allocate. If the complete set of required VMs are not allocated in the selected datacenter, the *ACOBrokerScheduler* is executed again to select a new datacenter.

3.1.2 Scheduler Based on PSO

In this scheduler, a *particle* is initialized for finding the most suitable datacenter in which to allocate the VMs requested by an user for executing his/her FPA-jobs. Analogously to the example based on nature described in Appendix A, each particle is considered a bee and each datacenter represent locations in the field with different density of flowers. A particle is created each time a datacenter is requested to allocate the user VMs. When a particle is created, it is initialized in a random datacenter, i.e., in a random place in the field. The density of flowers of each datacenter is determined both the estimation percentage of VMs that can be allocated and the weighted metric (which is calculated as in ACO) of the datacenter which the particle is visiting. This definition helps to search in the search space and try to balance both the monetary costs and the makespan. The smaller the weighted metric on a datacenter, the better the flowers concentration. In addition, each time a particle visit a datacenter it check that the datacenter meets the percentage of VMs to allocate defined by the user. In the algorithm (see Algorithm 3), every time a datacenter is required, a particle is initialized in a random datacenter (*getInitialDatacenter()*). Each particle in the search space takes a position according to the weighted metric of the datacenter in which is initialized through the *getWM(datacenterId)* method. The weighted metric is calculated as well as ACO. The neighborhood of each particle is composed by the remaining datacenters 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 the *getNeighbors(datacenterId,neighborSize)* method. Each one of the neighbors—datacenters—that compose the neighborhood are selected randomly until the *neighborhoodSize* parameter defined by the user is reached.

Algorithm 3 PSO-based datacenter selection algorithm

```
Procedure PSOBrokerScheduler(dcList, vmList, neighborhoodSize)
Begin
  particle = new Particle(dcList)
  datacenterId = particle.getInitialDatacenter()
  resAvailability=getResAvailability(datacenterId)
  vmEstimation= calculateVMsPerc(resAvailability, vmList)
  currentWM = particle.getWM(datacenterId)
  neighbors = particle.getNeighbors(datacenterId, neighborSize)
  While (i < neighborhoodSize) do
    neighborId = neighbors.get(i)
    destResAvailability=getResAvailability(neighborId)
    destVmEstimation= calculateVMsPerc(destResAvailability, vmList)
    destWM = particle.getWM(neighborId)
    if(currentWM − destWM > velocity && destVmEstimation <= vmPercentage)
      velocity = currentWM − destWM
      currentWM = destWM
      destDatacenterId = neighbors.get(i)
    end if
    i=i+1
  end while
  selectedDatacenter=dcList.get(destDatacenterId)
  VmScheduler(selectedDatacenter,vmList)
  if(vmList.size()>0)
    PSOBrokerScheduler(dcList, vmList, neighborhoodSize)
  else
    vmScheduler.start()
  end if
End
```

In each iteration of the algorithm, the particle moves to the neighbors of its current datacenter in search of a datacenter with the lowest weighted metric, and which also meets the percentage of VMs to allocate defined by the user. The velocity of each particle is defined by the weighted metric difference between the datacenter to which the particle has been previously assigned with respect to its other neighboring datacenters. If any of the datacenters in the neighborhood has a lower weighted metric than the (randomly chosen) original datacenter, then the particle is moved to the neighbor datacenter with a greater velocity. Taking into account that the particles moved through datacenters of their neighborhood in search of a datacenter with the lowest weighted metric, the algorithm reaches a local optimum quickly. Thus, each particle makes a move from their associated datacenter to the neighbor which has the minimum weighted metric. If all its neighbors have a greater weighted metric than the associated datacenter itself, the particle is not moved from the current datacenter. Moreover, if the weighted metric of its neighbors is equal to the weighted metric of the particle associated datacenter, the particle selects the neighbor which has lower latency, that is, the particle prioritizes the makespan of the FPA, as in ACO. Finally, the particle calls the infrastructure-level scheduler with the selected datacenter and the list of VMs to allocate. If the total number of VMs are not allocated in the selected datacenter, the *PSOBrokerScheduler* is executed again to select a new datacenter.

3.1.3 Alternative Scheduler Based on GA

This algorithm is implemented at this level for comparative purposes with our schedulers based on ACO and PSO. In the algorithm, proposed in [23], the population structure is represented as the set of datacenters. Each chromosome is an individual in the population that represents a subset of the searching space. Each gene (field in a chromosome) is a datacenter, and the last field in this structure is the fitness field, which indicates the suitability of the datacenters in each chromosome, i.e., the fitness field indicates the result of the fitness function and it is calculated as the inverse of the accumulated weighted metrics of all datacenters that compose the chromosome. The weighted metric in each is datacenter is calculated in the same way as ACO and PSO.

A chromosome with higher fitness indicates that its associated set of datacenters has the most suitable datacenters to be selected. Each chromosome keeps combinations of datacenters and its associated fitness. This fitness value is updated every time a datacenter is requested by a broker to indicate the suitability of the datacenters in each chromosome.

In each generation, a new population *P2* originated from the initial population *P* is formed by selecting chromosomes using a Roulette method, given a probability of selection proportional to the chromosome fitness. This *P2* population is recombined using a uniform crossover with the aim of exploring more possible datacenters with better fitness than the current selection. The evaluation step is done over the *P2* population to update the fitness field of this new recombined population. Chromosomes with low fitness in *P* are replaced by the better individuals in *P2*. Thus, the algorithm preserves the best individuals to increase the probability of a better selection. At the end of generations, two sorting steps are done: one local to provide a sorted list of datacenters in the chromosome with higher fitness, and a global sort, to provide a sorted list of individuals with better fitness. The selection will begin in the first datacenter of the first chromosome. If this datacenter is not suitable for allocating the percentage of VMs defined by the user, then the next datacenter in the chromosome with better fitness is selected.

3.2 Intra-Datacenter Scheduler

To implement the infrastructure level policy, we use the ACO and PSO algorithms we previously proposed in [11]. Below we describe these algorithms and an alternative scheduler based on GA that we use at this level for comparative purposes to our proposal.

3.2.1 Scheduler Based on ACO

The scheduler at the infrastructure level is performed to find those hosts in the selected datacenter at the broker level that have availability to allocate VMs. Here, each ant works independently and represents a VM "looking" for the best host to which it can be allocated, i.e., an ant is initialized for each VM allocated to the datacenter. A master table containing information on the load of each host is initialized. To do this, first, a list of all suitable hosts in which can be allocated the VM is obtained. In each iteration, the ant collects the load information of the host that it is visiting and adds this information to its private load history. The ant then updates a load information table of visited hosts, which is maintained in each host. This table contains information of the own load of an ant, as well as load information of other hosts, which were added to the table when other ants visited the host. Like in the ACO based broker algorithm, the load table of each host acts as a pheromone trail and it is useful to guide other ants to choose better paths. 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 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 H to another it has two choices: moving to a random host using a constant probability or *searchRate*, or using the load table information of H. Again, the search rate decreases with a *decreaseRate* factor as time passes. This process is repeated until the finishing criterion, i.e., performing a predefined number of steps (*maxAntSteps*), is met. Due to the fact that the datacenters have different numbers of hosts with each other, the *maxAntSteps* varies depending on the datacenter being explored. Specifically, the *maxAntSteps* parameter varies for each datacenter according to a user-defined percentage value. Finally, the ant delivers its VM to the current host and finishes its task. Besides, every time the ant allocate its associated VM, the total availability of the datacenter in which the VM is allocated is updated. In the same way, every time a VM finishes its task and is released, the total availability of the datacenter in which the VM is released is updated, thus increasing the overall availability of the datacenter.

Since each step by an ant involves moving through the intra-datacenter network to obtain information regarding the availability of the hosts from the selected datacenter, it incurs latencies. We have added a control to minimize the number of steps performed by an ant: every time an ant visits a host that has not allocated VMs yet, i.e., the host load is equal to zero, the ant allocates its associated VM to it directly without performing further steps. It is important to note also that although latencies are minimized at this level, they have much less impact on the makespan than those produced inter-datacenter. The smaller the number messages sent to the hosts through the network, the smaller the impact of the latencies in the makespan given to the user, and therefore, a lower monetary cost (because of the pay-per-hour basis of Clouds). This control to minimize the number of steps performed by an ant favors sending less number of messages regardless the network topology of the datacenter.

3.2.2 Scheduler Based on PSO

In order to find the hosts that have availability to allocate VMs this algorithm is started. Similarly to the PSO at the broker level, each particle works independently and represents a VM looking for the best host—in the previously selected datacenter—to which it can be allocated. Following the aforementioned analogy at the broker level, in this algorithm 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. The density of flowers of each host is determined by its load.

This definition helps to search in the load search space 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, for each VM requested by the user, a particle is initialized in a random host of the selected, 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. Load refers to the total CPU utilization within a host and is calculated as well as ACO. The neighborhood of each particle is composed by the remaining hosts in the datacenter excluding the one in which the particle is initialized, i.e., in the same way that at the broker level, the neighborhood represents other places in the field with different flower concentration. The neighborhood of that particle is obtained randomly. Moreover, like the *maxAntSteps* parameter of ACO, in this algorithm the size of the *neighborhoodHostsSize* parameter varies depending of the datacenter which is being explored according to a predefined percentage value.

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 v 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 is less loaded than the original host, then the particle is moved to the neighbor host with a greater velocity. 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 traveling through the intra-datacenter network, similarly to ACO, a control to minimize the number of moves that a particle performs have been added: every time a particle moves from the associated host to a neighbor host that has not allocated VMs yet, the particle allocates its associated VM to it immediately. Again, although latencies are minimized at this level, they have much less impact on the makespan than those produced inter-datacenter.

3.2.3 Alternative Scheduler Based on GA

Similarly to the GA at the broker level, the population structure is represented as a subset of physical resources that compose the selected datacenter at the broker level. Again, each chromosome is an individual in the population that represents a part of the search space. Each gene (field in a chromosome) is a host in the datacenter, and the last field in this structure is the fitness field, which indicates the suitability of the hosts, i.e., the result of the fitness function and it is calculated as the inverse of the accumulated load of all hosts composing the chromosome. The load in each is host is calculated taking into account the number of VMs that are executing in it. A chromosome with higher fitness indicates that its associated set of hosts has the most free cores to perform the current allocation. The chromosome selection mechanism as well as the steps of crossover and fitness evaluation are performed in the same way as the GA at the broker level.

3.3 VM Scheduler

Once the VMs have been allocated to hosts at the infrastructure level, the scheduler proceeds to assign the jobs to these VMs. The VMs were instantiated by the scheduler at the infrastructure level from the VM images supported (i.e., offered) by each datacenter which meet the requirements of memory, CPU, storage and bandwidth established by the user. The user is the one who indicates which are the characteristics of the VMs that needs to instantiate at the moment of requesting the VMs to the Cloud. Concretely, the VMs are instantiated by the scheduler at the infrastructure level from the VM images supported by each datacenter and that meets the requirements of memory, CPU, storage and bandwidth established by the user. The user is the one who indicates which are the characteristics of the VMs that needs to instantiate at the moment of requesting VMs to the federated Cloud.

At this level, the scheduling algorithm uses two lists, one containing the jobs that have been sent by the user, i.e., a FPA, and the other list contains all user VMs that are already allocated to a host and hence are ready to execute jobs. The algorithm iterates the list of all jobs and then, retrieves jobs by a FIFO policy. Each time a job is obtained from the list it is submitted to be executed in a VM in a round robin fashion. Internally, the algorithm maintains a queue for each VM that contains its list of jobs to be executed. The procedure is repeated until all jobs have been submitted for execution using the allocated VMs. To ensure fairness, 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.

4 Evaluation

To assess the effectiveness of our scheduler and constituting policies/techniques, we processed a frost prediction application (FPA) with data extracted from real sensors in the field. Broadly, the experimental methodology involved two steps. First, we executed a precision agriculture application in a single amazon instance by varying the number of sensor nodes to be processed, which allowed us to gather real job data, i.e., processing times and input/output file data sizes (see Sect. 4.1). By means of the generated job data, we instantiated the CloudSim simulation toolkit, which is explained in Sect. 4.2. Lastly, the obtained results regarding the performance of our proposal compared to some Cloud scheduling alternatives are reported in Sect. 4.3.

4.1 Frost Prediction Application

The study of frosts prediction is of special interest in many places around the world [48], and in particular in the Province of Mendoza, Argentina [49]. The reason is because frosts are one of the main causes of crops damage in the region and generates large economic losses in agricultural production,[3] mainly when they occur in spring season, affecting vineyards and fruit trees. In Mendoza, mainly adventive and radiation frosts occur, and while the first ones are predictable with traditional meteorological numerical models [1], the second ones are not.

Although frost happens every years, defense methods like heathers, sprinklers and wind turbines are used by farmers in order to minimize frost damage. Frost defense methods are activated by alarms generated by Frost Alarm Systems (FAS). FAS perform on-field data acquisition and data management. Moreover, FAS ensure production quality and guarantee crops traceability. The on-field data acquisition process can be performed using traditional instruments like thermometers, weather stations or Wireless Sensor Networks (WSNs) [3, 4]. Compared to traditional measurement instruments and weather stations, WSNs have the advantage that they can cover extensive areas with low cost devices called sensor nodes. This advantage is of special interest for studying frosts, due to the dependence of this phenomenon with terrain characteristics like presence of weeds, trees or adjacency to mountains. Sometimes it has been observed the occurrence of frost only in a few hectares of the farm (such as those at the base of mountains) and in other hectares of the same farm (such as those surrounded by trees) the event was not observed.

For the purposes of this paper, we based our research on an FPA based on the method presented by Snyder and Melo-Abreu [1]. In order to perform the frost prediction, the method takes temperature and humidity data—collected through a WSN and weather stations—and calculates and dew points in the days in which radiation frosts occurred. These days must belong to the month in which the prediction is performed (regardless of the year). In addition, it is necessary that temperature, humidity and dew points have been registered two hours after sunset in the prediction day. An example of the data collected format are provided in Appendix B.

Formally, the minimum temperature is calculated by the following multiple regression (MR) equation in 1:

$$T_p = s_T * T_o + s_D * D_0 + i\,, \tag{1}$$

where T_p is the minimum temperature to be predicted, T_o and D_0 are the temperature and dew point, respectively, registered the same day of the frost prediction two hours after sunset, i is the MR intercept value. Finally, s_T is the temperature slope and s_D the dew point slope. The values of s_T and i are calculated from the Eqs. 2 and 3, respectively.

$$s_T = \frac{\sum(T_{h0} - \bar{T}_{h0})(T_m - \bar{T}_m)}{\sum(T_{h0} - \bar{T}_{h0})^2}\,, \tag{2}$$

[3] Historical statistics can be found here (in Spanish): http://acovi.com.ar/observatorio/wp-content/uploads/2014/09/Ambiental5.xlsx.

$$i = \frac{\sum T_m - s_T \sum T_{h0}}{n}, \tag{3}$$

where T_{h0} are historical temperatures registered two hours after sunset in the same month in which the frost prediction is performed, T_m is the minimum temperature that happened in those days, and n is the number of historical data. Finally, $\bar{T}_{h0}$ and $\bar{T}_m$ are the average temperatures data. The dew point slope s_D is calculated by using the Eq. 4:

$$s_D = \frac{\sum(D_{h0} - \bar{D}_{h0})(R - \bar{R})}{\sum(D_{h0} - \bar{D}_{h0})^2}, \tag{4}$$

where D_{h0} are historical dew points two hour after sunset in the same month in which the frost prediction is performed and R the residuals. The parameters $\bar{D}_{h0}$ and $\bar{R}$ are the average values of D_{h0} and R, respectively. Finally, the residual is calculated with the expression: $R = T_m - s_T * T_o + i$.

The FPA was coded in Java. An MySQL database was used for storing both the WSN and weather stations data, and the obtained results after running the FPA. Moreover, the integration of WSN data with Cloud infrastructures was performed through a WSN-Cloud integration platform called Sensor Cirrus.[4] Sensor Cirrus [50] manages the WSN data using Cloud services and includes the development of the FPA for data processing. Figure 2 illustrates a scheme of the FPA [51].

As can be seen in step (1) of Fig. 2, a database containing weather stations and sensors nodes data is generated. Next, in step (2), the FPA gets from the database the needed data to execute the frost prediction (T_o, D_o, etc.). Finally, in step (3) the FPA is executed in the Cloud resulting in the minimum temperature that will occur in the night (4).

4.2 CloudSim Instantiation

In this work, data have been collected through WSNs and weather stations instrumented in the field, in order to simulate a frost prediction in a region of the Province of Mendoza, Argentina. For this purpose, a scenario in which 40 farms are instrumented has been modeled and simulated, each one of them with a different number of sensor nodes that vary between 10 and 1000 sensor nodes depending on the farm size (see Table 2). Then, in order to perform a prediction for each one of the farms, historical data (temperature, humidity and dew points) of 50 days in which there have been frosts are considered. The historical data was obtained from the National Oceanic and Atmospheric Administration[5] database

[4] https://sensorcirrus.com/.

[5] http://www.noaa.gov/.

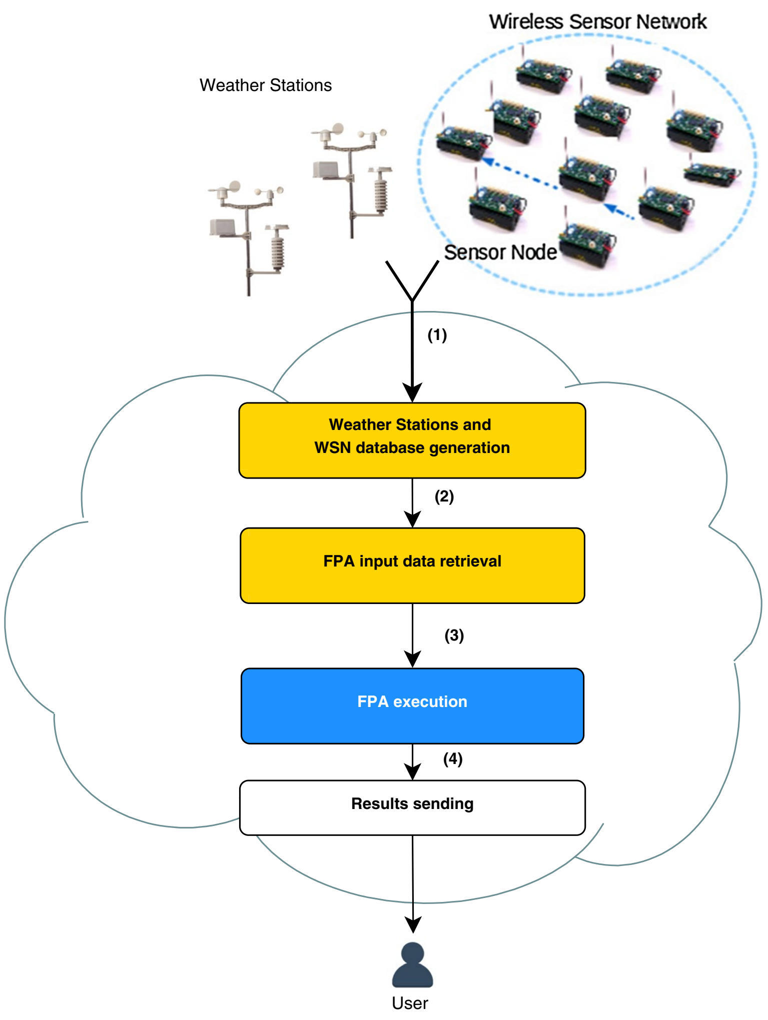


Fig. 2 Frost prediction application: overview

and collected through meteorological stations (see Appendix B). These 50 days must correspond the same month in which the prediction is performed (regardless the year), i.e., if the prediction is performed in July, historical data of 50 days with frost in July must be collected.

After gathering real data through the WSNs deployed in each one of the farms, we employed a single *m1.large* instance from Amazon EC2 to run the FPA with data from each one of the 40 farms. The reason why this instance type was chosen to set VMs in CloudSim is because in [50] was the instance which achieved the lowest makespan for executing different number of sensor nodes. Specifically, in [50] the experiments were performed in a real Cloud in order to find out which type of Amazon EC2 instance have better performance for

Table 2 Real FPA-jobs execution times (in one VM) and lengths

FPA-job Id	Number of sensor nodes in a farm	Execution time (s)	Length (MI)	FPA-job Id	Number of sensor nodes in a farm	Execution times (s)	Length (MI)
1	10	44	221,324	21	100	463	2,313,542
2	10	47	233,880	22	100	430	2,152,085
3	10	55	273,556	23	100	659	3,294,133
4	10	64	321,629	24	100	617	3,084,405
5	20	104	519,922	25	200	851	4,254,854
6	20	135	676,868	26	200	1190	5,952,555
7	20	155	774,263	27	200	1054	5,269,906
8	20	146	729,481	28	200	858	4,289,475
9	30	119	596,530	29	400	1341	6,704,712
10	30	124	619,631	30	400	3192	15,962,055
11	30	138	690,664	31	400	1620	8,098,335
12	30	109	543,668	32	400	1498	7,490,760
13	60	276	1,379,523	33	800	2924	14,621,554
14	60	240	1,198,350	34	800	3094	15,468,715
15	60	226	1,129,402	35	800	3158	15,792,208
16	60	231	1,155,044	36	800	2642	13,212,840
17	80	261	1,306,605	37	1000	4763	23,818,010
18	80	329	1,646,019	38	1000	5516	27,579,553
19	80	281	1,405,579	39	1000	5050	25,251,201
20	80	268	1,338,041	40	1000	2979	14,894,086

executing FPAs. The metrics measured in such work were the execution time and economic cost. The experiments consisted of executing the FPA with data from different number of sensors nodes in different types of Amazon EC2 instances (t1.micro, m1.small, m1.large, m1.xlarge and c3.xlarge), i.e., 40 FPA-jobs were executed in the different types of instances with data from10 to 1000 sensors nodes depending of the farm size. Once the FPA was executed, the shortest execution times and economic costs were obtained when executing the FPA in the m1.large instance.

The execution of 40 FPA-jobs (see Table 2) resulted in 40 input files (sensor nodes data) and 40 output files (with frost prediction information). The test was solved using Sensor Cirrus [50]. Once the execution times were obtained, we approximated for each field frost prediction—or job—the number of executed CPU instructions by the following formula $NI_i = mipsCPU * T_i$, where NI_i is the number of million instructions (MI) to be executed by, or associated to, a job i, $mipsCPU$ is the processing power of the CPU of our real computer measured in MIPS, and T_i is the time that took to run the job i on the VM. We have used such measure (MIPS) because in CloudSim the processing power both of the physical machines and the

Table 3 Cloud datacenters (DC) characteristics

DC	# Hosts	Latency (s)	Monetary cost (hourly)	Hosts characteristics			
				Proc. power (MIPS)	RAM (GB)	Storage	Cores
D1	30	0.80	$1,11	7200	32	500 GB	8
D2	50	1.75	$1,42	9900	32	1 TB	6
D3	20	0.32	$0,19	8036	16	500 GB	8
D4	30	2.00	$1,22	7500	16	1 TB	8
D5	50	0.25	$1,32	7200	32	500 GB	8
D6	10	1.50	$0,35	4008	8	1 GB	4
D7	20	0.29	$0,17	5618	12	500 GB	6
D8	20	2.20	$0,73	5200	8	500 GB	4
D9	50	0.50	$0,13	6600	12	500 GB	8
D10	50	1.20	$1,57	7527	16	500 GB	8

VMs must be configured in MIPS. As a consequence, the same must be done to configure jobs, which have associated a length expressed in MI. Subsequently, once the simulation is performed, the makespan of jobs is reported in seconds. Next is an example of how to calculate the number of instructions of a job that took 463 seconds to execute. The VM (EC2 instance) where the experiment was executed had a processing power of 5000.21 MIPS. Then, the approximated number of instructions for the job was 2,313,542 MI (Million Instructions). Resulting jobs execution times for the 40 farms are shown in Table 2. Note that in a real-world scenario the overall execution time of a job n times in the same machine will not be exactly the same the n times. The reason is because the jobs executions times depend on the load state of the underlying resources at the moment they are executed. For this reason, in order to perform more realistic experiments, the real jobs execution times showed in Table 2 were randomly modified with a margin of variability between −0.20 and 0.20% [52].

After gathering real job data, we instantiated the CloudSim toolkit [21], which is heavily used within the community to evaluate Cloud solutions. The experimental scenario consists of a federated Cloud composed of 10 heterogeneous datacenters. The network topology is defined using BRITE [53]. BRITE is a topology generation tool that provides a topology file used by CloudSim to define the different Cloud nodes that compose a commonly-found federation (i.e., datacenters, brokers) and the network connections among them. Each datacenter is composed of a different number of hosts which are not all dedicated, i.e., some of them are busy executing pre-existing jobs in other VMs. In our scenario, each datacenter has already allocated a random number of VMs, which involve a percentage of busy hosts between 30 and 80%, i.e., of the total datacenter availability. The characteristics of the the datacenters and the machines that compose them are shown in Table 3. All hosts have an internal bandwidth of 1,000 Mbps. Moreover, a user requests 100 VMs to execute its

FPA-jobs. Each VM has the same characteristics as a *m1.large* instance of Amazon EC2 (5000 MIPS, 7.5 GB RAM, 100 GB image size and 2 CPU).

The number of instructions to be executed by each job (Length in Table 2) varies between 221,324 MI and 25,579,553 MI. Moreover, the experiments have input file and output file sizes of 1.6 MB and 2.03 MB, respectively.

In this work, we evaluated the performance of executing the frost prediction in an entire region as we increased the number of FPA-jobs—described in Table 2—to be performed. Specifically, for each region-specific frost prediction, we evaluated the performance of their associated FPA-jobs in the simulated Cloud as we increased the number of FPA-jobs to be executed, i.e., $40 * i$ jobs with $i = 25, 50, \ldots, 250$. This is, the base job set comprising 40 farms (showed in Table 2) were cloned to obtain larger sets, i.e., regions composed of a greater number of farms. Therefore, the base FPA-jobs set to perform varies between 1000 and 10,000.

4.3 Performed Experiments

Next we report the results when executing the FPA in an entire region, which means the prediction of the FPA-jobs in the simulated federated Cloud. Execution is handled using our three-level scheduler and two GA-based alternative schedulers both for selecting datacenters and assigning VMs to hosts. Due to their high CPU requirements, the jobs that are waiting to be executed in a VM are executed one at a time by competing for the CPU time with other jobs from other VMs that are allocated in the same physical machine. In other words, a time-shared CPU scheduling policy was used at the datacenter level, which ensures fairness. Particularly, we study/combine the two SI-based policies for selecting datacenters at the broker level discussed in Sects. 3.1.1 and 3.1.2, and the policies for mapping VMs to hosts described in Sects. 3.2.1 and 3.2.2 while comparing them against GA [23] (see Sects. 3.1.3 and 3.2.3).

In our experiments both at the broker and infrastructure levels, the specific-parameter of each algorithm (e.g., *neighborhoodSize* and *neighborhoodHostSize* in PSO, *maxSteps* and *maxAntSteps* in ACO and *chromosomeSize* in GA), has been configured so as to explore up to 60% of the number of datacenters—neighborhoodSize and maxSteps—and the number of hosts of each datacenter—neighborhoodHostSize and maxAntSteps—, i.e., the value of the specific-parameter is equals to 6 when the number of datacenters is equal to 10 and is equal to 12 when the number the host at the infrastructure level is equal to 20. Furthermore, in the ACO algorithms we have set the mutation rate and decay rate parameters with values equal to 0.6 and 0.1, respectively, and the GA population size equals 100. In [23] the authors have set the chromosome size equal to the number of hosts, but in this paper we have reduced this number in order to reduce network consumption and being fair to GA with respect to ACO and PSO. Finally, the *vmPercentage* parameter has been set to 60%, i.e., each selected datacenter by an ant should be available to allocate at least 60% of the requested VMs by the user. For simplicity, from now on, we will refer to policies at the broker level as BPSO, BACO, BGA in order to differentiate them from schedulers at the infrastructure level. In all cases, the competing policies both

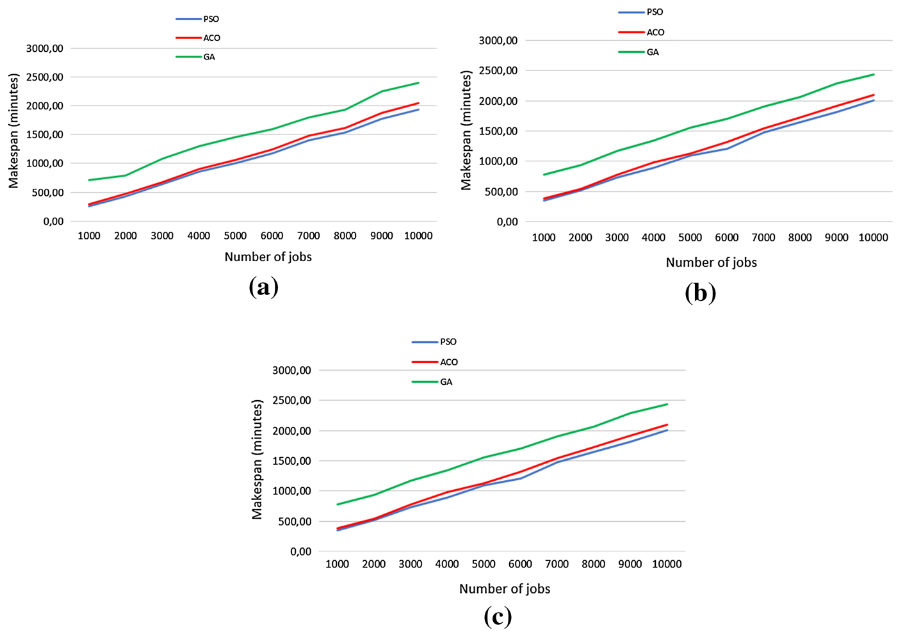


Fig. 3 Makespan as the number of FPA-jobs increases. **a** BPSO-based broker, **b** BACO-based broker, **c** BGA-based broker

at the broker level and the infrastructure level were also used in conjunction with the VM-level FIFO-based policy for handling jobs within VMs.

Figure 3 illustrates the makespan results when executing the FPA described in Sect. 4.1. Moreover, each one of the subfigures in Fig. 3 compare the makespan for each one of the policies implemented at the broker level (BPSO, BACO, BGA) and all the considered scheduling algorithms at the infrastructure level (PSO, ACO, GA), respectively. Below we show how each one of the scheduling levels (broker, infrastructure and VM) have influenced the performance metrics.

As can be seen in Fig. 3a our BPSO algorithm is the one that delivers the lowest makespan to the user with respect to BACO and BGA. The lowest makespan is obtained when BPSO is combined with PSO at the infrastructure level. The lowest makespan is equal to 202.02 and 1456.96 minutes when the number of jobs to be executed is 1000 and 10,000, respectively. In the second place is BPSO combined with ACO, whose makespans are equal to 216.07 and 1535.33 minutes when the number of jobs is increased from 1000 to 10,000, and in the third place is BACO combined with PSO (see Fig. 3b), whose makespans vary between 262.76 and 1935.57 minutes when the number of jobs are 1000 and 10,000, respectively. This happens because most VMs are allocated in datacenters with lower latencies, and therefore they have less influence in the completion time—makespan—when PSO, ACO and GA—at the infrastructure level—send network messages to the hosts to inquire about their availability. Besides, since all the broker schedulers consider both network latencies and the percentage of VMs that can be allocated in a

Table 4 Average gains of using the combination BPSO-PSO w.r.t. BACO and BGA (0-100%)

	BACO			BGA		
	PSO (%)	ACO (%)	GA (%)	PSO (%)	ACO (%)	GA (%)
BPSO-PSO	24.67	25.79	48.18	31.82	33.48	51.88

datacenter, they avoid to explore datacenters with lower latency but which can allocate few VMs. It is desirable to avoid exploring datacenters with limited availability of resources because such searches involve making use of network resources and therefore, a greater number of latencies influence the overall execution time of the application.

Table 4 shows the average gain of using the combination of BPSO-PSO—the policies through which the lowest makespan was obtained—regarding BACO and BGA combined with PSO, ACO and GA—the competing meta-heuristic algorithm considered in this work at the infrastructure level—. The makespan average gains are calculated as in equation 5, where *FPAbaseSet* is equals to 40 jobs—40 instrumented farms—and $i = 25, 50, \dots, 250$; and makespan (BrokerS-InfrastructureS) is the combination of the schedulers at the broker level with the schedulers at the infrastructure level (e.g., BACO-PSO, BACO-ACO, etc.).

$$makespanGain = \left[\sum_{j=FPAbaseSet*i} \frac{(makespan_j(BrokerS - InfrastructureS) - makespan_j(BPSO - PSO))}{(makespan_j(BrokerS - InfrastructureS))} \right] \Big/ 10 \tag{5}$$

As can be seen, the best average gains of BPSO-PSO—48.18% and 51.88%—are obtained regarding BACO-GA and BGA-GA, respectively. It is important to note, however, that the use of BPSO-PSO yield also important gains w.r.t. BACO-PSO, BACO-ACO, BGA-PSO and BGA-ACO, whose gains around 25% and 33%.

Secondly, among all the algorithms implemented at the infrastructure level and regardless the policy used at the broker level, Fig. 3a–c show that the described PSO and ACO performed rather well compared to GA regarding the makespan, being PSO the algorithm that achieves the best performance. At this level, each algorithm sends a different number of messages to the hosts to query about their availability and allocate the VMs. With respect to PSO and ACO, they make less use of network resources than GA, being PSO the one which sends less network messages. The number of messages to send both by PSO and ACO depends of the *neighborhoodHostSize* and the *maxAntSteps* parameters, respectively, i.e., the maximum number of moves that a particle/ant performs to allocate its associated VM, which is equal to the 60% of a datacenter size. It is important to note that, when PSO and ACO find an idle host, they allocate the current VM and immediately stop hosts exploration. This action reduces the total number of messages sent through the network, and therefore, the total latencies which influence the overall makespan. Finally, GA is the algorithm that produces the greatest makespan in all cases. Since GA contains

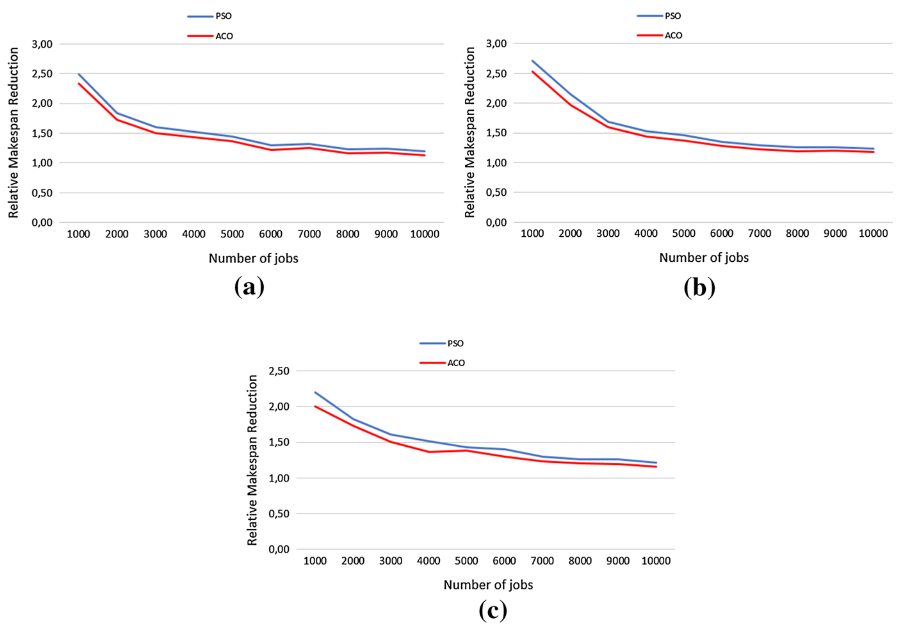


Fig. 4 Relative makespan reduction regarding GA. **a** BPSO-based broker, **b** BACO-based broker, **c** BGA-based broker

a population size of 100 and the chromosome sizes are of 60% of a datacenter size, to calculate the fitness value, the algorithm sends one message for each host of the chromosome to know its availability and obtain the chromosome containing the best fitness value. The number of messages sent by GA depends on both the number of host within each chromosome and the population size.

The reason why, regardless of the policy used at the broker level, PSO provides the shortest makespan, it is because this algorithm does not repeat the visited hosts in each allocation of a VM. As we explained in Sect. 3.2.2, each particle visits each one of the hosts in its neighborhood, which are different from each other, looking for the host with the lowest load. This increases the chances of PSO of finding a host with load equals to zero, thereby reducing the total number of moves to perform. Moreover, in the ACO algorithm there is the possibility that an ant visits some hosts more than once, thereby reducing the total number of different visited hosts. This is because ACO uses a random function in the early steps to choose the host to which it performs the movement. This random function may force the ant to repeat visiting a host when moving from one host to another.

Complementary, Fig. 4 summarizes the relative makespan reduction regarding the worst competitor—GA—as the number of jobs is increased and for all schedulers at the broker level. Besides, due to the fact that the PSO and ACO makespan results are very close, Table 5 shows the obtained gains of using PSO and ACO at the infrastructure level with respect to GA as the number of jobs increase. As can be seen, the greatest gains are obtained when BPSO is used, being PSO the scheduler which achieves the best gains. Concretely, the gain of BPSO-PSO with respect to

Table 5 Makespan Gains of using PSO and ACO w.r.t. GA (0–100%)

Number of FPA-jobs	BPSO		BACO		BGA	
	Gains PSO w.r.t. GA	Gains ACO w.r.t. GA	Gains PSO w.r.t. GA	Gains ACO w.r.t. GA	Gains PSO w.r.t. GA	Gains ACO w.r.t. GA
1000	63.13	59.93	57.91	56.14	54.63	50.00
2000	48.62	47.06	46.47	41.90	45.21	40.22
3000	41.54	39.20	38.44	35.23	37.93	33.53
4000	36.47	34.64	33.06	31.90	31.95	29.44
5000	32.38	31.31	30.67	28.79	29.92	26.65
6000	28.81	27.98	26.56	22.97	25.68	20.71
7000	24.40	23.98	22.39	19.83	21.82	18.82
8000	22.20	21.16	19.36	16.91	18.58	14.80
9000	21.83	18.46	20.23	16.56	19.78	14.02
10,000	19.42	14.94	18.11	13.60	16.50	11.72

BPSO-GA yielded as a result 63.13% when the number of jobs is equal to 1,000. In the second place is BPSO-ACO, whose gains with respect to BPSO-GA yield as a result 59.93%. Note that the larger the number of jobs to be executed, the lower the impact of the latencies in the makespan, and therefore, the lower gains. The reason is because the latencies are set at the moment of creating the virtual infrastructure.

Due to the fact that our aim is to reduce both the makespan and monetary cost, in Fig. 5 it can be seen that important monetary cost reductions are also obtained when we use first, the BPSO-PSO and second, the BPSO-ACO schedulers. The reason is because, as we show above, the selection of datacenters with lower latencies produce important improvements in the makespan, and therefore, important reduction in monetary costs. Due to Cloud VMs are leased per execution hour, the monetary costs are closely related to the application's makespan. Particularly, Figs. 4b and 5a, c illustrate the monetary costs by each one of the schedulers implemented at the broker level. As shown in all the subfigures, regardless of the policy used at the broker level, PSO is the algorithm that produces the lower monetary cost to the user with respect to ACO and GA. Particularly, the lowest monetary cost is obtained when BPSO is combined with PSO as well as the makespan. The lower the makespan the lower the monetary cost for the user. Similarly to the makespan, the average gains obtained by BPSO-PSO regarding BGA-GA and BACO-GA are 47.40% and 50.48%, respectively. Moreover, the use of BPSO-PSO yield also gains w.r.t. BACO-PSO, BACO-ACO, BGA-PSO and BGA-ACO, with gains between 24% and 36%. This average gains means that our proposed scheduler achieves a greater effectiveness and system reactivity when executing the FPA, i.e., it allows to obtain the frost predictions in a lower time and at a lower cost. Obtaining the frost predictions in a low time is important due to the fact that it allows the farmers to be alerted as soon as possible to a possible frost phenomenon. Reducing costs is important because it would also allow farmers to instrument their farms with a greater number of sensors

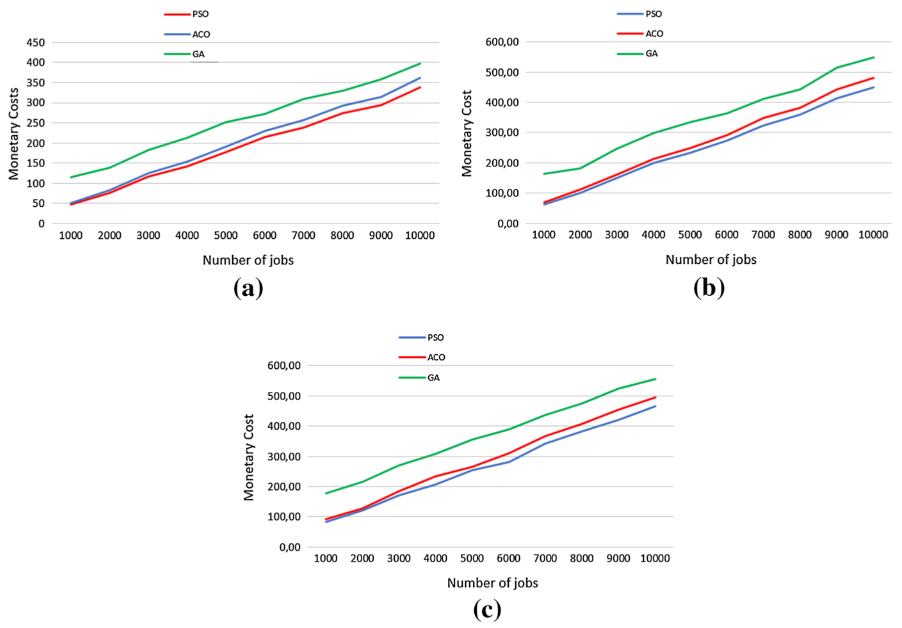


Fig. 5 Monetary cost as the number of FPA-jobs increases. **a** PSO-based broker, **b** ACO-based broker, **c** GA-based broker

nodes. A greater number the sensors nodes would allow either to achieve a more accurate prediction or to extend the surface of the field to be installed.

5 Conclusions

Federated Clouds [10] potentially provide plenty of resources to users, specially when the number of VMs required by a user exceeds the maximum capacity that can be provided by a single provider or datacenter. Then, broker/VM/job scheduling plays a fundamental role since it is basically NP-complete [13], and thus many variants based on approximation techniques have been proposed. In our view, in federated Clouds, scheduling should be performed at three levels (broker, infrastructure and VM) [11], making the problem even more challenging compared to other distributed environments.

SI-inspired algorithms have received increasing attention in the Cloud research community for dealing with a large number of optimization problems, such as Cloud scheduling [15, 30]. SI refers to the collective behavior that emerges from a swarm of social insects. Social insect colonies collectively solve complex problems through intelligent emergent behavior. Historically, researchers have proposed algorithms exploiting this idea for solving a variety of combinatorial optimization problems. Moreover, scheduling in Clouds is a combinatorial optimization problem, and many schedulers based on SI, particularly ACO and PSO, have been proposed. Basically, researchers have introduced changes to the traditional bio-inspired techniques to achieve different Cloud scheduling goals [15]. However, to the best of our knowledge, existing efforts do not address in general *federated Clouds* where the different providers/datacenters geographically distributed are selected through these strategies.

Therefore, in this work we have presented a three level Cloud scheduler based on SI for the efficient execution, in terms of the makespan and monetary cost, of FPAs on federated Clouds. The novelty of this scheduler is the inclusion of SI at broker level. Concretely, the scheduler includes at the broker level two policies—BPSO and BACO—that consider network information, monetary costs and resources availability of datacenters. Then, at the infrastructure level, the policies at the broker level are also combined with two bio-inspired strategies—based on ACO and PSO—for the efficient allocation of VMs in the hosts of a selected datacenter. Finally, at the VM level, through a FIFO policy, jobs are assigned to the allocated VMs. Simulated experiments performed with CloudSim and real FPA job data suggest that our bio-inspired three-level scheduler provide a better balance between makespan and monetary cost to the user regarding GA. Particularly, when PSO, ACO and GA are combined with BPSO, the makespan and monetary cost are the lowest w.r.t. BACO and BGA being BPSO-PSO the lowest among them. Particularly, when PSO, ACO and GA are combined with BPSO, the makespan and monetary cost are the lowest w.r.t. BACO and BGA being BPSO-PSO the lowest among them. Average gains range from approximately 10% and 32%.

We are extending this work in several directions. We will explore the ideas exposed in this paper in the context of other bio-inspired techniques such as Artificial Bee Colony (ABC) [54, 55], which is also extensively used to solve combinatorial optimization problems. Another issue to consider is enhance the scheduler with dynamic optimization capabilities, enabling the dynamic reallocation (migration) of VMs from one host to another. The migration of VMs might allow to meet a specific optimization criteria such as reduce the number of hosts in use for minimizing energy consumption or balance the workload of all resources to avoid resources saturation and performance slowdown.

An aspect that deserves special attention is to incorporate other types of scientific experiments. Some examples of applications that could benefit from being executed in Clouds are scientific workflows. Scientific workflows applications [52, 56, 57] are common in areas such as bioinformatics, earthquake science, and astronomy, and its main feature is based on the jobs are executed according to their dependencies, and besides, the jobs have the characteristic of being not only CPU intensive but also data intensive. Data intensive computing [58] is a type of parallel computing application which typically processes terabytes or petabytes of data and it is often referred to as Big Data. This application types devote most of their processing time to I/O and manipulation/movement of data. Due to the fact that these applications require the transfer of large volumes of data, it is important to develop new scheduling strategies at the broker level that not only consider the latency of datacenters in which jobs will be executed, but also their bandwidths. This will provide excellent research opportunities for new schedulers based on SI-based optimization techniques.

In addition, in order to improve the performance of the proposed scheduler it is also important to extend the use of bio-inspired algorithms to the VM level. In the FPA used in this work, the execution of a job corresponds to the frost prediction in a farm, and moreover, each farm is instrumented with different number of sensors nodes (depending on the farm size). The greater the number of sensor nodes a farm has, the greater amount of data to be processed to predict the frost and, therefore, the greater the execution time and monetary cost to execute each job. Implementing bio-inspired strategies that have information about the jobs sizes would allow a better search of a suitable VM to execute each job, and as a consequence, it could greatly help improve the overall performance of the FPA. At this level we plan to explore both ACO and PSO as well as other SI-based algorithms such as ABC or Artificial Fish Swarm Algorithm (AFSA) [59].

Finally, we plan to explore other types of network topologies approaching real-life Cloud scenarios, such as fat-tree and leaf-spine. However, for this type of topologies to be included it is necessary to completely redesign the scheduling algorithms so that they are able to route the different nodes. Moreover, for evaluation purposes, we could explore FTCloudSim [60], an extension to CloudSim that in principle supports fat-tree datacenter topologies.

Acknowledgements We acknowledge the financial support provided by ANPCyT through grants PICT-2012-2731, PICT-2014-1430 and PICT-2015-1435, and UNCuyo project project 06/B308. We want to thank the anonymous reviewers for their valuable comments and suggestions that helped to improve the quality of this paper.

Appendix A: Swarm Intelligence Techniques

SI [61] bases on studying collective behaviors that emerge from interactions between individuals and the environment which they live in to solve optimization problems. Examples of systems in which SI inspires are ants colonies, fish schools, birds flocks, and herds of land animals, where the whole group of individuals perform a desired task (i.e., feeding), which might not be made individually. For example, an ant is relatively unintelligent, but when it is part of a colony, some behaviors emerge from the interactions between ants, such as searching for food. Moreover, an individual fish makes dynamic decisions to swim in one direction or another but only up to a certain point. If there are several fishes, when the first fish swims near a food source, the other fishes will listen to the first fish instead of following other instincts.

According to M. Dorigo and M. Birattari [62], in an SI system (a) there are many individuals, (b) the individuals are relatively homogeneous, i.e., they are either all identical or they belong to a few typologies, (c) the interactions among the individuals are based on simple behavioral rules that exploit only local information that the individuals exchange directly or via the environment, and (d) the overall behavior of the system results from the interactions of individuals with each other and with their environment, i.e., the group behavior self-organizes.

The following subsections briefly describe the most popular SI techniques—i.e., ACO and PSO—that are widely used in job scheduling problems such as the one

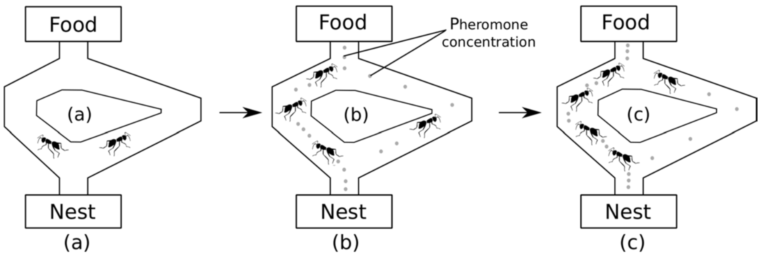


Fig. 6 Adaptive behavior of ants

addressed in this paper. This appendix has been partially extracted from a paper of our own [15].

Ant Colony Optimization

The ACO algorithm [63] arises from the way real ants behave in nature. An interesting aspect of this behavior is how ants manage to locate short paths to reach a food source from their nest. The ACO algorithm can solve computational problems since the algorithm has the ability to reduce paths and precisely to find the shortest paths. In nature, ants move randomly from one place to another to search for food. On the return to its nest each ant leaves an hormone—called pheromone—that lures other working ants to the same course. When more and more ants choose the same path, the pheromone trail is intensified and even more ants will further choose it. Over time the shortest paths will be intensified by the pheromone faster. That is because the ants will both reach the food source and travel back to their nest at a faster rate. Furthermore, if over time ants do not follow a certain path, its pheromone trail evaporates. From an algorithmic point of view, the pheromone evaporation process is useful for preventing the convergence to a local optimum solution.

Figure 6 shows two possible nest-food source paths. Figure 6a shows that ants will move randomly at the beginning and choose one of the two paths. The ants that follow the faster path will naturally get to the objective before other ants, and in doing so the former group of ants will leave a pheromone trail. Moreover, the ants that perform the round-trip faster, strengthen more quickly the quantity of pheromone in the shorter path (see Fig. 6b). The ants that reach the food source through the slower path will find attractive to return to the nest using the faster path. Eventually, most ants will choose the left path as shown in Fig. 6c.

ACO employs artificial pheromone trails that play the role of information that is dynamically updated by ants to reflect their accumulated experience in contributing to solve an entire problem. In practice, to optimize job scheduling problems, the ACO algorithm is mapped to graphical representations, usually graphs. A graph for example may include jobs and executing physical resources or machines (nodes) and scheduling decisions (arcs). Each job can be carried out by an ant to search for machines with available computing resources.

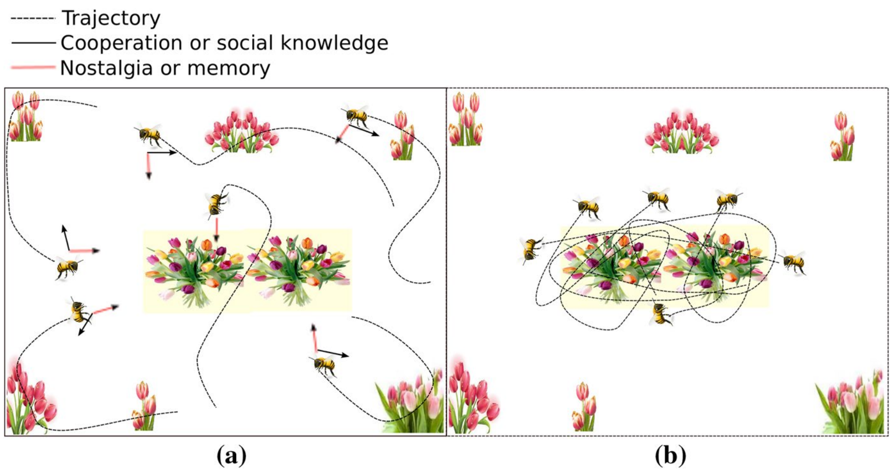


Fig. 7 PSO modeling using as example the movement of a bees swarm on a flowers field. **a** Bees are attracted to areas of highest concentration of flowers found by each individual, **b** bees are attracted to the area with the highest flowers concentration of the whole swarm

Particle Swarm Optimization

PSO [64] mimics the behavior of natural processes from animals such as birds and insects such as bees. In PSO the general term "particle" is used to represent birds, bees or any other individuals who exhibit social and group behavior. Suppose 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 more 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.

A bee may fly over a place with more flowers than it had been found by any bee in the swarm. The whole swarm would then be drawn toward that location by considering the bees own individual discovery. In this way the bees explore the field by overflying locations with the greatest flower 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. After that, bees will be unable to find any points of higher flower concentration, so they will be always drawn back to F. This social behavior is shown in Fig. 7.

When used to model job scheduling, a PSO algorithm instantiation consists of particles, each maintaining one potential solution to the entire scheduling problem. The design of the representation of a particle is different for each type of problem.

Table 6 Weather station data used for the FPA

Sensors Data							
Year	Month	Day	Hour	Minute	Cloud coverage	Temperature	Humidity
2003	3	5	6	0	OVC	24	57
2003	3	5	7	0	OVC	22	59
2003	3	5	8	0	BKN	22	59
2003	3	5	9	0	BKN	23	51
2003	3	5	10	0	BKN	22	52
2003	3	5	11	0	BKN	22	57
2003	3	5	12	0	BKN	23	57
2003	3	5	13	0	BKN	23	57
2003	3	5	14	0	BKN	23	57
2003	3	5	15	0	BKN	25	57
2003	3	5	16	0	BKN	25	56
2003	3	5	17	0	BKN	24	56
2003	3	5	18	0	BKN	22	56
2003	3	5	19	0	SCT	21	61
2003	3	5	20	0	SCT	22	59

An example of how to represent particles when modeling the job scheduling problem is placing the position of a particle in an *n*-dimensional search space. Each dimension *i* is the job to schedule, and the corresponding value represents the machine in which a job can be allocated. Furthermore, the global best position will indicate the best possible schedule.

Appendix B: Data Information from Weather Stations and Sensor Nodes

Table 6 illustrates the data structure used for running the FPA. This data were extracted from a raw dataset provided by the National Oceanic and Atmospheric Administration[6] database. The dataset contains 10 years of atmospheric data (from 2003 to 2013) that correspond to the the weather station of the Plumerillo Airport, Mendoza. Atmospheric data are expressed in METeorological Aerodrome Report (METAR) format. METAR format is commonly used by pilots in pre-flight weather briefing and by meteorologists for weather forecasting.

In the Table 6, the first five columns show the acquisition time data including year, month, day, hour and minute, respectively. Column six indicates cloud coverage (in METAR notation) and columns seven and eight indicate the temperature and humidity levels, respectively.

[6] http://www.noaa.gov/.

Table 7 METAR Cloud coverage notation

Abbreviation	Description
SCK	"No cloud/Sky clear"
CLR	"No clouds below 12,000 ft (3,700 m)"
NSC	"No significant cloud", i.e., none below 5,000 ft (1,500 m)
FEW	"Few" = 1–2 oktas
SCT	"Scattered" = 3–4 oktas
BKN	"Broken" = 5–7 oktas
OVC	"Overcast" = 8 oktas, i.e., full cloud coverage
VV	Clouds cannot be seen because of fog or heavy precipitation, so vertical visibility is given instead

Table 7 details METAR cloud coverage notation which is reported by the number of "oktas" (eighths) of the sky that is occupied by clouds. First column indicates the METAR abbreviation and the second column represents the abbreviation description. For example, in the first row of Table 6, OVC indicates that on 5/3/2003 at 06:00 am, the sky was overcast, i.e., with full cloud coverage. Regarding okta, this is a unit of measurement to describe the amount of cloud coverage at any given location, e.g., five oktas are equal to five octaves of the sky covered with clouds.

References

1. Snyder, R.L., de Melo-Abreu, J.P.: Frost Protection: Fundamentals, Practice and Economics, Volume 1 of Environment and Natural Resources Series. Food and Agriculture Organization of the United Nations (FAO), Rome (2005)
2. Bishop, C.: Pattern Recognition and Machine Learning, Volume 20 of Information Science and Statistics. Springer, Berlin (2006)
3. Oliveira, L., Rodrigues, J.: Wireless sensor networks: a survey on environmental monitoring. J. Commun. **6**(2), 143–151 (2011)
4. Rehman, A., Abbasi, A.Z., Islam, N., Shaikh, Z.A.: A review of wireless sensors and networks' applications in agriculture. Comput. Stand. Interfaces **36**(2), 263–270 (2014)
5. Buyya, R., Yeo, C., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. **25**(6), 599–616 (2009)
6. Mauch, V., Kunze, M., Hillenbrand, M.: High performance cloud computing. Future Gener. Comput. Syst. **29**(6), 1408–1416 (2013) **(Including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems)**
7. Zhai, Y., Liu, M., Zhai, J., Ma, X., Chen, W.: Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running mpi applications. In: State of the Practice Reports, vol. 11, pp. 1–11. ACM (2011)
8. Coutinho, R.C., Drummond, L.M., Frota, Y., de Oliveira, D.: Optimizing virtual machine allocation for parallel scientific workflows in federated clouds. Future Gener. Comput. Syst. **46**, 51–68 (2014)
9. Petri, I., Beach, T., Mengsong, Z., Montes, J.D., Rana, O., Parashar, M.: Exploring models and mechanisms for exchanging resources in a federated cloud. In: IEEE International Conference on Cloud Engineering (IC2E), pp. 215–224. IEEE (2014)
10. Celesti, A., Fazio, M., Villari, M., Puliafito, A.: Virtual machine provisioning through satellite communications in federated cloud environments. Future Gener. Comput. Syst. **28**(1), 85–93 (2012)

11. Pacini, E., Mateos, C., García Garino, C., Careglio, C., Mirasso, A.: A bio-inspired scheduler for minimizing makespan and flowtime of computational mechanics applications on federated clouds. J. Intell. Fuzzy Syst. **31**(3), 1731–1743 (2016)
12. Manasrah, A.M., Smadi, T., ALmomani, A.: A variable service broker routing policy for data center selection in cloud analyst. J. King Saud Univ. Comput. Inf. Sci. **29**(3), 365–377 (2017)
13. Woeginger, G.: Exact algorithms for NP-Hard problems: a survey. In: Junger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization—Eureka, You Shrink!, volume 2570 of Lecture Notes in Computer Science, pp. 185–207. Springer (2003)
14. Kennedy, J.: Swarm Intelligence. In: Zomaya, Albert Y. (ed.) Handbook of Nature-Inspired and Innovative Computing, pp. 187–219. Springer, New York (2006)
15. Pacini, E., Mateos, C., García Garino, C.: Distributed job scheduling based on Swarm Intelligence: a survey. Comput. Electr. Eng. **40**(1), 252–269 (2014). 40th-year commemorative issue
16. Pacini, E., Mateos, C., García Garino, C.: Balancing throughput and response time in online scientific clouds via ant colony optimization. Adv. Eng. Softw. **84**, 31–47 (2015)
17. Pacini, E., Mateos, C., García Garino, C.: SI-based scheduling of parameter sweep experiments on federated clouds. In: Hernandez, G., et. al. (eds.) First HPCLATAM—CLCAR Joint Conference Latin American High Performance Computing Conference (CARLA), volume 845 of High Performance Computing. Communications in Computer and Information Science, pp. 28–42. Springer (2014)
18. Pacini, E., Mateos, C., García Garino, C.: A three-level scheduler to execute scientific experiments on federated clouds. IEEE Latin Am. Trans. **13**(10), 3359–3369 (2015)
19. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
20. Iosup, A., Ostermann, S., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Trans. Parallel Distrib. Syst. **22**(6), 931–945 (2011)
21. Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
22. Pacini, E., Mateos, C., García Garino, C.: Multi-objective Swarm Intelligence schedulers for online scientific clouds. Special Issue on Cloud Computing. Computing, pp. 1–28 (2014)
23. Agostinho, L., Feliciano, G., Olivi, L., Cardozo, E., Guimaraes, E.: A Bio-inspired approach to provisioning of virtual resources in federated Clouds. In: Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), DASC 11, pp. 598–604, Washington, DC, USA, 12–14 December 2011. IEEE Computer Society (2011)
24. Chandra Mohan, B., Baskaran, R.: A survey: ant colony optimization based recent research and implementation on several engineering domain. Expert Syst. Appl. **39**(4), 4618–4627 (2012)
25. Mahdiyeh, E., Hussain, S., Mohammad, K., Azah, M.: A survey of the state of the art in Particle Swarm Optimization. Res. J. Appl. Sci. Eng. Technol. **4**(9), 1181–1197 (2012)
26. Pedemonte, M., Nesmachnow, S., Cancela, H.: A survey on parallel Ant Colony Optimization. Appl. Soft Comput. **11**(8), 5181–5197 (2011)
27. Poli, R.: Analysis of the publications on the applications of Particle Swarm Optimisation. J. Artif. Evol. Appl. **2008**(4), 1–10 (2008)
28. Tavares Neto, R.F., Godinho Filho, M.: Literature review regarding Ant Colony Optimization applied to scheduling problems: guidelines for implementation and directions for future research. Eng. Appl. Artif. Intell. **26**(1), 150–161 (2013)
29. Vlachos, A.: Ant colony system algorithm solving a thermal generator maintenance scheduling problem. J. Intell. Fuzzy Syst. **24**(4), 713–723 (2013)
30. Masdari, M., Salehi, F., Jalali, M., Bidaki, M.: A survey of PSO-based scheduling algorithms in cloud computing. J. Netw. Syst. Manag. **25**(1), 122–158 (2017)
31. Singha, U., Jain, S.: An analysis of swarm intelligence based load balancing algorithms in a cloud computing environment. Int. J. Hybrid Inf. Technol **8**(1), 249–256 (2015)
32. Zhan, Z.H., Liu, X.F., Gong, Y.J., Zhang, J., Chung, H.S.H., Li, Y.: Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Comput. Surv. **47**(4), 63:1–63:33 (2015)

33. Marosi, A.C., Kecskemeti, G., Kertesz, A., Kacsuk, P.: Fcm: anarchitecture for integrating iaas cloud systems. In: Cloud computing 2011: the second international conference on cloud computing, GRIDs, and virtualization, pp. 7–12. IARIA (2011)
34. Villegas, D., Bobroff, N., Rodero, I., Delgado, J., Liu, Y., Devarakonda, A., Fong, L., Sadjadi, S.M., Parashar, M.: Cloud federation in a layered service model. J. Comput. Syst. Sci. **78**(5), 1330–1344 (2012). JCSS Special Issue: Cloud Computing 2011
35. Tordsson, J., Montero, R.S., Moreno Vozmediano, Rl, Llorente, I.M.: Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Gener. Comput. Syst. **28**(2), 358–367 (2012)
36. Kessaci, Y., Melab, N., Talbi, E.-G.: A pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation. Cluster Comput. **16**(3), 451–468 (2013)
37. Lucas-Simarro, J.L., Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.: Scheduling strategies for optimal service deployment across multiple clouds. Future Gener. Comput. Syst. **29**(6), 1431–1441 (2013) **(Including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems)**
38. Song, Y., Peng, J., Liu, K., Jiang, F., Liu, W., Huang, Z.: A hybrid particle swarm ant colony based resource reservation for geo-distributed cloud service. In: 2016 IEEE Global Communications Conference (GLOBECOM), pp 1–6. IEEE (2016)
39. Kumrai, T., Ota, K., Dong, M., Kishigami, J., Sung, D.K.: Multiobjective optimization in cloud brokering systems for connected internet of things. IEEE Internet Things J. **4**(2), 404–413 (2016)
40. Feller, E., Rilling, L., Morin, C.: Energy-Aware Ant Colony based workload placement in clouds. In: 12th International Conference on Grid Computing, number 8 in Grid '11, pp. 26–33. IEEE Computer Society (2011)
41. Gao, Y., Guan, H., Qi, Z., Hou, Y., Liu, L.: A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J. Comput. Syst. Sci. **79**(8), 1230–1242 (2013)
42. Jeyarani, R., Nagaveni, N., Vasanth Ram, R.: Design and implementation of adaptive power-aware virtual machine provisioner (APA-VMP) using swarm intelligence. Future Gener. Comput. Syst. **28**(5), 811–821 (2012)
43. Dhinesh Babu, L.D., Venkata Krishna, P.: Honey bee behavior inspired load balancing of tasks in cloud computing environments. Appl. Soft Comput. **13**(5), 2292–2303 (2013)
44. de Oliveira, G.S., Ribeiro, E., Ferreira, D.A., Araújo, A.P.,Holanda, M., Walter, M.E.: ACOsched: a scheduling algorithm in afederated Cloud infrastructure for bioinformatics applications. In: International Conference on Bioinformatics and Biomedicine, pp. 8–14. IEEE (2013)
45. Zhang, G., Zuo, X.: Deadline constrained task scheduling based on standard-pso in a hybrid cloud. In: Tan, Y., Shi, Y., Mo, H. (eds.) Advances in Swarm Intelligence: 4th International Conference, ICSI 2013, pp. 200–209, Harbin, China. Springer, Berlin (2013)
46. Gabaldon, E., Vila, S., Guirado, F., Lerida, J.L., Planes, J.: Energy efficient scheduling on heterogeneous federated clusters using a fuzzy multi-objective meta-heuristic. In: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1–6 (2017)
47. Sedeño Noda, A., Raith, A.: A dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem. Comput. Oper. Res. **57**, 83–94 (2015)
48. Breque, F., Nemer, M.: Frosting modeling on a cold flat plate: comparison of the different assumptions and impacts on frost growth predictions. Int. J. Refrig. **69**, 340–360 (2016)
49. Brun Laguna, K., Diedrichs, A.L., Chaar, J.E., Dujovne, D., Taffernaberry, J.C., Mercado, G., Watteyne, T.: A demo of the peach iot-based frost event prediction system for precision agriculture. In: 2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), pp. 1–3. IEEE (2016)
50. Iacono, L., Vázquez-Poletti, J.L., García Garino, C., Llorente, I.M.: A Model to Calculate Amazon EC2 Instance Performance in Frost Prediction Applications. In: Hernández, G., et al. (eds.) High Performance Computing: First HPCLATAM—CLCAR Latin American Joint Conference, CARLA 2014, pp. 68–82. Springer, Berlin (2014)
51. Iacono, L., Vázquez-Poletti, J.L., García Garino, C., Llorente, I.M.: A performance models for frost prediction on public cloud infrastructures. Comput. Inf. **37**(4), 815–837 (2018)
52. Monge, D.A., Pacini, E., Mateos, C., García Garino, C.: Meta-heuristic based autoscaling of cloud-based parameter sweep experiments with unreliable virtual machines instances. Comput. Electr. Eng. **69**, 364–377 (2018)

53. Jung, J.K., Jung, S.M., Kim, T.K., Chung, T.M.: A study on the cloud simulation with a network topology generator. World Acad. Sci. Eng. Technol. **6**(11), 303–306 (2012)
54. Madivi, R., Kamath, S.S.: An hybrid bio-inspired task scheduling algorithm in cloud environment. In Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT), pp. 1–7. IEEE (2014)
55. Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artif. Intell. Rev. (2012)
56. Ghafarian, T., Javadi, B.: Cloud-aware data intensive workflow scheduling on volunteer computing systems. Future Gener. Comput. Syst. **51**, 87–97 (2015)
57. Zhao, Y., Li, Y., Raicu, L., Lu, S., Tian, W., Liu, H.: Enabling scalable scientific workflow management in the cloud. Future Gener. Comput. Syst. **46**, 3–16 (2015)
58. Philip Chen, C.L., Zhang, C.Y.: Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf. Sci. **275**, 314–347 (2014)
59. Neshat, M., Sepidnam, G., Sargolzaei, M., Toosi, A.N.: Artificial fish swarm algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications. Artif. Intell. Rev. **42**(4), 965–997 (2014)
60. Zhou, A., Wang, S., Yang, C., Sun, L., Sun, Q., Yang, F.: Ftcloudsim: support for cloud service reliability enhancement simulation. Int. J. Web Grid Serv. **11**(4), 347–361 (2015)
61. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, Oxford (1999)
62. Dorigo, M., Birattari, M.: Swarm intelligence. Scholarpedia **2**(9), 1462 (2007)
63. Dorigo, M., Stützle, T.: The ant colony optimization metaheuristic: algorithms, applications, and advances. In: Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science, chapter 9, pp. 250–285. Springer (2003)
64. Kennedy, J., Eberhart, R.: Swarm Intelligence. Morgan Kaufmann Publishers Inc., San Francisco (2001)

Elina Pacini received her Ph.D. degree in Computer Science from the UNICEN, in 2014, and a B.Sc. in Information Systems Engineering from the UTN-FRM, Argentina, in 2005. She is Adjunct Professor at the UNCuyo and member of the ITIC and the CONICET. She is interested in job and VM scheduling, Cloud Computing and scientific applications.

Lucas Iacono received his Ph.D. degree in Engineering from the Mendoza University, in 2015, and a B.Sc. in Electronic and Electrical Engineering from the same institution in 2007. He is a full professor at the UNCuyo and member of the ITIC. His research interests include Mobile Robotics, Internet of Things and Cloud Computing.

Cristian Mateos received his Ph.D. degree in Computer Science from the UNICEN, in 2008, and his M.Sc. in Systems Engineering in 2005. He is a full time Adjunct Professor at the UNICEN and member of the ISISTAN and the CONICET. He is interested in parallel/distributed programming, Grid middlewares, Service-oriented Computing and Mobile Computing.

Carlos García Garino received his Ph.D. degree from Universidad Politécnica de Cataluña, Spain, in 1993, and the Civil Engineering degree from UBA, Argentina, in 1978. He is a full Professor at the UNCuyo, and director of the ITIC. He is interested in Computer Networks, Distributed Computing and Computational Mechanics.

Affiliations

Elina Pacini[1,2] · **Lucas Iacono**[1,2] · **Cristian Mateos**[3] · **Carlos García Garino**[1]

Lucas Iacono
liacono@uncu.edu.ar

Cristian Mateos
cmateos@conicet.gov.ar

Carlos García Garino
cgarcia@itu.uncu.edu.ar

1 ITIC and Facultad de Ingeniería, UNCuyo University, Mendoza, Argentina

2 CONICET, Buenos Aires, Argentina

3 ISISTAN-CONICET, UNICEN University, Tandil, Buenos Aires, Argentina