Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds

Maria Alejandra Rodriguez and Rajkumar Buyya

Abstract—Cloud computing is the latest distributed computing paradigm and it offers tremendous opportunities to solve large-scale scientific problems. However, it presents various challenges that need to be addressed in order to be efficiently utilized for workflow applications. Although the workflow scheduling problem has been widely studied, there are very few initiatives tailored for cloud environments. Furthermore, the existing works fail to either meet the user's quality of service (QoS) requirements or to incorporate some basic principles of cloud computing such as the elasticity and heterogeneity of the computing resources. This paper proposes a resource provisioning and scheduling strategy for scientific workflows on Infrastructure as a Service (IaaS) clouds. We present an algorithm based on the meta-heuristic optimization technique, particle swarm optimization (PSO), which aims to minimize the overall workflow execution cost while meeting deadline constraints. Our heuristic is evaluated using CloudSim and various well-known scientific workflows of different sizes. The results show that our approach performs better than the current state-of-the-art algorithms.

Index Terms—Cloud computing, resource provisioning, scheduling, scientific workflow

1 Introduction

Workflows have been frequently used to model large-scale scientific problems in areas such as bioinformatics, astronomy, and physics [1]. Such scientific workflows have ever-growing data and computing requirements and therefore demand a high-performance computing environment in order to be executed in a reasonable amount of time. These workflows are commonly modeled as a set of tasks interconnected via data or computing dependencies. The orchestration of these tasks onto distributed resources has been studied extensively over the years, focusing on environments like grids and clusters. However, with the emergence of new paradigms such as cloud computing, novel approaches that address the particular challenges and opportunities of these technologies need to be developed.

Over the years, distributed environments have evolved from shared community platforms to utility-based models; the latest of these being cloud computing. This technology enables the delivery of IT resources over the Internet [2], and follows a pay-as-you-go model where users are charged based on their consumption. There are various types of cloud providers [2], each of which has different product offerings. They are classified into a hierarchy of as-a-service terms: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). This paper focuses on IaaS clouds which offer the user a virtual pool of unlimited, heterogeneous resources that can be accessed on demand. Moreover, they offer the flexibility of elastically acquiring or releasing resources with varying configurations to best suit the requirements of an application. Even though this empowers the users and gives them more control over the resources, it also dictates the development of innovative scheduling techniques so that the distributed resources are efficiently utilized.

There are two main stages when planning the execution of a workflow in a cloud environment. The first one is the resource provisioning phase; during this stage, the computing resources that will be used to run the tasks are selected and provisioned. In the second stage, a schedule is generated and each task is mapped onto the best-suited resource. The selection of the resources and mapping of the tasks is done so that different user defined quality of service (QoS) requirements are met. Previous works in this area, especially those developed for Grids or Clusters, focused mostly on the scheduling phase. The reason behind this is that these environments provide a static pool of resources which are readily available to execute the tasks and whose configuration is known in advance. Since this is not the case in cloud environments, both problems need to be addressed and combined in order to produce an efficient execution plan.

Another characteristic of previous works developed for clusters and grids is their focus on meeting application deadlines or minimizing the makespan (total execution time) of the workflow while ignoring the cost of the utilized infrastructure. Whilst this is well suited for such environments, policies developed for clouds are obliged to consider the pay-per-use model of the infrastructure in order to avoid prohibitive and unnecessary costs.

Virtual machine (VM) performance is an additional challenge presented by cloud platforms. VMs provided by current cloud infrastructures do not exhibit a stable performance in terms of execution times. In fact, Schad et al. [21]

• *The authors are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia.*
E-mail: mariaars@student.unimelb.edu.au, rbuyya@unimelb.edu.au.

Manuscript received 17 Dec. 2013; revised 18 Mar. 2014; accepted 23 Mar. 2014. Date of publication 1 Apr. 2014; date of current version 30 July 2014.
Recommended for acceptance by I. Bojanova, R.C.H. Hua, O. Rana, and M. Parashar.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2314655

report an overall CPU performance variability of 24 percent on Amazon's EC2 cloud. The shared nature of the infrastructure as well as virtualization and the heterogeneity of the underlying non-virtualized hardware are some of the reasons behind such variability. This may have a significant impact when scheduling workflows on clouds and may cause the application to miss its deadline. Many scheduling policies rely on the estimation of task runtimes on different VMs in order to make a mapping decision. This estimation is done based on the VMs computing capacity and if this capacity is always assumed to be optimal during the planning phase, the actual task execution will most probably take longer and the task will be delayed. This delay will also impact the task's children and the effect will continue to escalate until the workflow finishes executing.

Our work is based on the meta-heuristic optimization technique, particle swarm optimization (PSO). PSO was first introduced by Kennedy and Ebehart in [4] and is inspired on the social behavior of bird flocks. It is based on a swarm of particles moving through space and communicating with each other in order to determine an optimal search direction. PSO has better computational performance than other evolutionary algorithms [4] and fewer parameters to tune, which makes it easier to implement. Many problems in different areas have been successfully addressed by adapting PSO to specific domains; for instance this technique has been used to solve problems in areas such as reactive voltage control [5], pattern recognition [6] and data mining [7], among others.

In this paper, we develop a static cost-minimization, deadline-constrained heuristic for scheduling a scientific workflow application in a cloud environment. Our approach considers fundamental features of IaaS providers such as the dynamic provisioning and heterogeneity of unlimited computing resources as well as VM performance variation. To achieve this, both resource provisioning and scheduling are merged and modeled as an optimization problem. PSO is then used to solve such problem and produce a schedule defining not only the task to resource mapping but also the number and type of VMs that need to be leased, the time when they need to be leased and the time when they need to be released. Our contribution is therefore, an algorithm with higher accuracy in terms of meeting deadlines at lower costs that considers heterogeneous resources that can be dynamically acquired and released and are charged on a pay-per-use basis.

The rest of this paper is organized as follows. Section 2 presents the related work followed by the application and resource models as well as the problem definition in Section 3. Section 4 gives a brief introduction to PSO while Section 5 explains the proposed approach. Finally, Section 6 presents the evaluation of the algorithm followed by the conclusions and future work described in Section 7.

2 Related Work

Workflow scheduling on distributed systems has been widely studied over the years and is NP-hard by a reduction from the multiprocessor scheduling problem [7]. Therefore it is impossible to generate an optimal solution within polynomial time and algorithms focus on generating approximate or near-optimal solutions. Numerous algorithms that aim to find a schedule that meets the user's QoS requirements have been developed. A vast range of the proposed solutions target environments similar or equal to community grids. This means that minimizing the application's execution time is generally the scheduling objective, a limited pool of computing resources is assumed to be available and the execution cost is rarely a concern. For instance, Rahman et al. [9] propose a solution based on the workflow's dynamic critical paths, Chen and Zhang [10] elaborate an algorithm based on ant colony optimization that aims to meet different user QoS requirements and, finally, Yu and Buyya use Genetic Algorithms to implement a budget constrained scheduling of workflows on utility Grids [11].

The aforementioned solutions provide a valuable insight into the challenges and potential solutions for workflow scheduling. However, they are not optimal for utility-like environments such as IaaS clouds. There are various characteristics specific to cloud environments that need to be considered when developing a scheduling algorithm. For example, Mao and Humphrey propose a dynamic approach for scheduling workflow ensembles on clouds [12]. They acknowledge that there are various types of VMs with different prices and that they can be leased on demand, depending on the application's requirements. Furthermore, they tailor their approach so that the execution cost is minimized based on the cloud's pricing model, that is, VMs are paid by a fraction of time, which in most cases is one hour. They try to minimize the execution cost by applying a set of heuristics such as merging tasks into a single one, identifying the most cost-effective VM type for each task and consolidating instances. Although this is a valid approach capable of reducing the execution cost of workflows on clouds, the solution proposed only guarantees a reduction on the cost and not a near-optimal solution.

Another recent work on workflow ensemble developed for clouds is presented by Malawski et al. [13]. They propose various dynamic and static algorithms which aim to maximize the amount of work completed, which they define as the number of executed workflows, while meeting QoS constraints such as deadline and budget. Their solutions acknowledge different delays present when dealing with VMs leased from IaaS cloud providers such as instance acquisition and termination delays. Furthermore, their approach is robust in the sense that the task's estimated execution time may vary based on a uniform distribution and they use a cost safety margin to avoid generating a schedule that goes over budget. Their work, however, considers only a single type of VM, ignoring the heterogeneous nature of IaaS clouds.

While the algorithms presented by Mao and Humphrey [12] and Malawski et al. [13] are designed to work with workflow ensembles, they are still relevant to the work done in this paper since they were developed specifically for cloud platforms and as so include heuristics that try to embed the platform's model. More in line with our work is the solution presented by Abrishami et al. [14] which presents a static algorithm for scheduling a single workflow instance on an IaaS cloud. Their algorithm is based on the workflow's partial critical paths and it considers cloud features such as VM heterogeneity, pay-as-you-go and time

interval pricing model. They try to minimize the execution cost based on the heuristic of scheduling all tasks in a partial critical path on a single machine which can finish the tasks before their latest finish time (which is calculated based on the application's deadline and the fastest available instance). However, they do not have a global optimization technique in place capable of producing a near-optimal solution; instead, they use a task level optimization and hence fail to utilize the whole workflow structure and characteristics to generate a better solution.

Other authors have used PSO to solve the workflow scheduling problem. Pandey et al. [15] propose a PSO based algorithm to minimize the execution cost of a single workflow while balancing the task load on the available resources. While the cost minimization objective is highly desired in clouds, the load balancing one makes more sense in a non-elastic environment such as a cluster or a grid. The execution time of the workflow is not considered in the scheduling objectives and therefore this value can be considerably high as a result of the cost minimization policy. The authors do not consider the elasticity of the cloud and assume a fixed set of VMs is available beforehand. For this reason, the solution presented is similar to those used for grids where the schedule generated is a mapping between tasks and resources instead of a more comprehensive schedule indicating the number and type of resources that need to be leased, when they should be acquired and released, and in which order the tasks should be executed on them.

Wu et al. [16] also use PSO to produce a near-optimal schedule. Their work focuses on minimizing either cost or time while meeting constraints such as deadline and budget. Despite the fact that their heuristic is able to handle heterogeneous resources, just as Pandey et al. [15], it assumes an initial set of VMs is available beforehand and hence lacks in utilizing the elasticity of IaaS clouds.

Finally, Byun et al. [23] develop an algorithm that estimates the optimal number of resources that need to be leased so that the execution cost of a workflow is minimized. Their algorithm also generates a task to resource mapping and is designed to run online. The schedule and resources are updated every charge time interval (i.e., every hour) based on the current status of the running VMs and tasks. Their approach takes advantage of the elasticity of the cloud resources but fails to consider the heterogeneous nature of the computing resources by assuming there is only one type of VM available.

3 Problem Formulation

3.1 Application and Resource Models

A workflow application $W = (T, E)$ is modeled as a directed acyclic graph (DAG) where $T = \{t_1, t_2, \ldots, t_n\}$ is the set of tasks and E is the set of directed edges. An edge e_{ij} of the form (t_i, t_j) exists if there is a data dependency between t_i and t_j, case in which t_i is said to be the parent task of t_j and t_j is said to be the child task of t_j. Based on this definition, a child task cannot be executed until all of its parent tasks are completed. A sample workflow is shown in Fig. 1. In addition, each workflow W has a deadline δ_W associated to it. A deadline is defined as a time limit for the execution of the workflow.

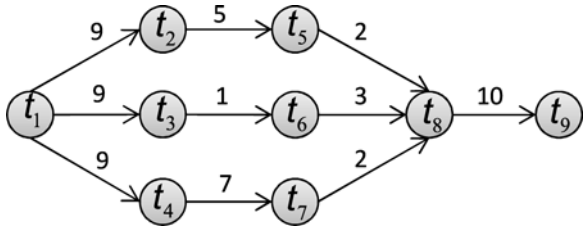

Fig. 1. Sample workflow. Each node represents a task and the arcs show the data transfer times between nodes.

The IaaS cloud provider offers a range of VM types. A VM type VM_i is defined in terms of its processing capacity P_{VM_i} and cost per unit of time C_{VM_i}. We target workflow applications such as those presented by Juve et al. [1]. Based on the profiling results obtained in their work for memory consumption and the VM types offered by Amazon EC2, we assume that VMs have sufficient memory to execute the workflow tasks.

We assume that for every VM type, the processing capacity in terms of floating point operations per second (FLOPS) is available either from the provider or can be estimated [17]. This information is used in our algorithm to calculate the execution time of a task on a given VM. Performance variation is modeled by adjusting the processing capacity of each leased VM and introducing a performance degradation percentage deg_{VM_i}.

The unit of time τ in which the pay-per-use model is based is specified by the provider; any partial utilization of the leased VM is charged as if the full time period was consumed. For instance, for $\tau = 60$ minutes, if a VM is used for 61 minutes, the user will pay for two periods of 60 minutes, that is, 120 minutes. Also, we assume that there is no limitation on the number of VMs that can be leased from the provider.

The execution time $ET_{t_i}^{VM_j}$ of task t_i in a VM of type VM_j and is estimated using the size I_{t_i} of the task in terms of floating point operations (FLOP). This is depicted in Equation (1). Additionally, $TT_{e_{ij}}$ is defined as the time it takes to transfer data between a parent task t_i and its child t_j and is calculated as depicted in Equation (2). To calculate $TT_{e_{ij}}$ we assume that the size of the output data $d_{t_i}^{out}$ produced by task t_i is known in advance and that the entire workflow runs on a single data center or region. This means that all the leased instances are located on the same region and that the bandwidth β between each VM is roughly the same. Notice that the transfer time between two tasks being executed on the same VM is 0. Finally, the total processing time $PT_{t_i}^{VM_j}$ of a task in a VM is computed as shown in Equation (3), where k is the number of edges in which t_i is a parent task and s_k is 0 whenever t_i and t_j run on the same VM or 1 otherwise:

$$ET_{t_i}^{VM_j} = I_{t_i}/(P_{VM_j} * (1 - deg_{vm_j})), \tag{1}$$

$$TT_{e_{ij}} = d_{t_i}^{out}/\beta, \tag{2}$$

$$PT_{t_i}^{VM_j} = ET_{t_i}^{VM_j} + \left(\sum_{1}^{k} TT_{e_{ij}} * s_k\right). \tag{3}$$

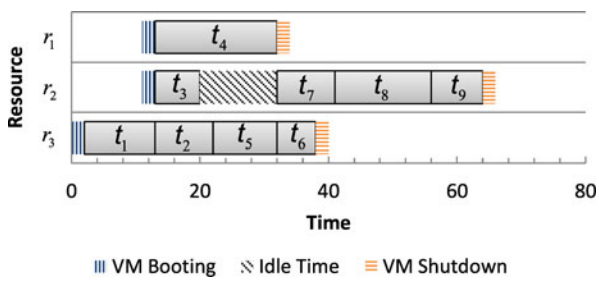

Fig. 2. Example of a schedule generated for the workflow shown in Fig. 1. Each task is mapped onto one of the three available resources. Parent tasks are executed before their children ensuring the data dependencies are met.

Many IaaS providers do not charge for data transfers if they are made within the same data center; hence, we do not consider this fee when calculating the workflow's execution cost. Nevertheless, we do consider the fact that a VM needs to remain active until all the output data of the running task is transferred to the VMs running the child tasks. Moreover, when a VM is leased, it requires an initial boot time in order for it to be properly initialized and be made available to the user; this time is not negligible and needs to be considered in the schedule generation as it could have a considerable impact on the same. We acknowledge such delay present in most cloud providers [21] when generating our schedule and calculating the overall cost for the execution of the workflow.

3.2 Problem Definition

Resource provisioning and scheduling heuristics may have different objectives; this work focuses on finding a schedule to execute a workflow on IaaS computing resources such that the total execution cost is minimized and the deadline is met. We define a schedule $S = (R, M, TEC, TET)$ in terms of a set of resources, a task to resource mapping, the total execution cost and the total execution time. Fig. 2 shows a sample schedule generated for the workflow depicted in Fig. 1. $R = \{r_1, r_2, \ldots, r_n\}$ is the set of VMs that need to be leased; each resource r_i has a VM type VM_{r_i} associated to it as well as an estimated lease start time LST_{r_i} and lease end time LET_{r_i}. M represents a mapping and is comprised of tuples of the form $m_{t_i}^{r_j} = (t_i, r_j, ST_{t_i}, ET_{t_i})$, one for each workflow task. A mapping tuple $m_{t_i}^{r_j}$ is interpreted as follows: task t_i is scheduled to run on resource r_j and is expected to start executing a time ST_{t_i} and complete by time ET_{t_i}. Equations (4) and (5) show how the total execution cost TEC and total execution time TET are calculated:

$$TEC = \sum_{i=1}^{|R|} C_{VM_{r_i}} * \left\lfloor \frac{(LET_{r_i} - LST_{r_i})}{\tau} \right\rfloor, \tag{4}$$

$$TET = max\{ET_{t_i} : t_i \in T\}. \tag{5}$$

Based on the previous definitions, the problem can be formally defined as follows: find a schedule S with minimum TEC and for which the value of TET does not exceed the workflow's deadline. This is depicted in Equation (6).

$$\begin{aligned} &Minimize\ TEC \\ &subject\ to\ TET \leq \delta_W. \end{aligned} \tag{6}$$

4 Particle Swarm Optimization

Particle swarm optimization is an evolutionary computational technique based on the behavior of animal flocks. It was developed by Eberhart and Kennedy [4] in 1995 and has been widely researched and utilized ever since. The algorithm is a stochastic optimization technique in which the most basic concept is that of particle. A particle represents an individual (i.e., fish or bird) that has the ability to move through the defined problem space and represents a candidate solution to the optimization problem. At a given point in time, the movement of particles is defined by their velocity, which is represented as a vector and therefore has magnitude and direction. This velocity is determined by the best position in which the particle has been so far and the best position in which any of the particles has been so far. Based on this, it is imperative to be able to measure how good (or bad) a particle's position is; this is achieved by using a fitness function that measures the quality of the particle's position and varies from problem to problem, depending on the context and requirements.

Each particle is represented by its position and velocity. Particles keep track of their best position *pbest* and the global best position *gbest*; values that are determined based on the fitness function. The algorithm will then at each step, change the velocity of each particle towards the *pbest* and *gbest* locations. How much the particle moves towards these values is weighted by a random term, with different random numbers generated for acceleration towards *pbest* and gbest locations [18]. The algorithm will continue to iterate until a stopping criterion is met; this is generally a specified maximum number of iterations or a predefined fitness value considered to be good enough. The pseudo code for the algorithm is shown in Algorithm 1. In each iteration, the position and velocity of a particle are updated based in Equations (7) and (8) respectively:

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t), \tag{7}$$

$$\begin{aligned} \vec{v}_i(t+1) = \omega \cdot \vec{v}_i(t) + c_1 r_1(\vec{x}_i^*(t) - \vec{x}_i(t)) \\ + c_2 r_2(\vec{x}^*(t) - \vec{x}_i(t)), \end{aligned} \tag{8}$$

where:

$$\begin{aligned} &\omega = inertia, \\ &c_i = acceleration\ coefficient,\ i = 1, 2 \\ &r_i = random\ number, i = 1, 2\ and\ r_i \in [0, 1] \\ &\vec{x}_i^* = best\ position\ of\ particle\ i \\ &\vec{x}^* = position\ of\ the\ best\ particle\ in\ the\ population \\ &\vec{x}_i = current\ position\ of\ particle\ i \end{aligned}$$

The velocity equation contains various parameters that affect the performance of the algorithm; moreover, some of them have a significant impact on the convergence of the algorithm. One of these parameters is ω, which is known as the inertia factor or weight and is crucial for the algorithm's convergence. This weight determines how much previous velocities will impact the current velocity and defines a tradeoff between the local cognitive component and global social experience of the particles. On one hand, a large

inertia weight will make the velocity increase and therefore will favor global exploration. On the other hand, a smaller value would make the particles decelerate and hence favor local exploration. For this reason, a ω value that balances global and local search implies fewer iterations in order for the algorithm to converge.

ALGORITHM 1
PARTICLE SWARM OPTIMIZATION

1. Set the dimension of the particles to d
2. Initialize the population of particles with random positions and velocities
3. For each particle, calculate its fitness value
   - **3.1** Compare the particle's fitness value with the particle's $pbest$. If the current value is better than $pbest$ then set $pbest$ to the current value and location
   - **3.2** Compare the particle's fitness value with the global best $gbest$. If the particle's current value is better than $gbest$ then set $gbest$ to the current value and location
   - **3.3** Update the position and velocity of the particle according to equations 7 and 8
4. Repeat from step 3 until the stopping criterion is met.

Conversely, c_1 and c_2 do not have a critical effect in the convergence of PSO. However, tuning them properly may lead to a faster convergence and may prevent the algorithm to get caught in local minima. Parameter c_1 is referred to as the cognitive parameter as the value $c_1 r_1$ in Equation (8) defines how much the previous best position matters. On the other hand, c_2 is referred to as the social parameter as $c_2 r_2$ in Equation (8) determines the behavior of the particle relative to other neighbors.

There are other parameters that are not part of the velocity definition and are used as input to the algorithm. The first one is the number of particles; a larger value generally increases the likelihood of finding the global optimum. This number varies depending on the complexity of the optimization problem but a typical range is between 20 and 40 particles. Other two parameters are the dimension of the particles and the range in which they are allowed to move, these values are solely determined by the nature of the problem being solved and how it is modeled to fit into PSO. Finally, the maximum velocity defines the maximum change a particle can have in one iteration and can also be a parameter to the algorithm; however, this value is usually set to be as big as the half of the position range of the particle.

5 PROPOSED APPROACH

5.1 PSO Modeling

There are two key steps when modeling a PSO problem. The first one is defining how the problem will be encoded, that is, defining how the solution will be represented. The second one is defining how the "goodness" of a particle will be measured, that is, defining the fitness function.

To define the encoding of the problem, we need to establish the meaning and dimension of a particle. For the scheduling scenario presented here, a particle represents a workflow and its tasks; thus, the dimension of the particle is equal to the number of tasks in the workflow. The dimension of a particle will determine the coordinate system used to define its position in space. For instance, the position of a two-dimensional particle is specified by two coordinates,

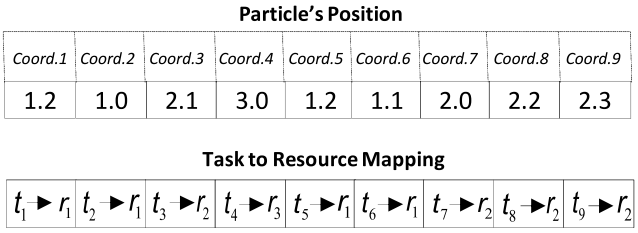

Fig. 3. Example of the encoding of a particle's position. There are nine tasks to schedule so the particle is nine-dimensional and its position has nine coordinate values. The coordinate index (coordinate 1 through coordinate 9) maps onto a task (t_1 through t_9). The value of each coordinate is a real number between 0 and the number of available resources, in this case 3. When rounded with the floor function, this values maps onto a resource (r_1 through r_3).

the position of a three-dimensional one is specified by three coordinates and so on. As an example, the particle depicted in Fig. 3 represents a workflow with nine tasks; the particle is a nine-dimensional one and its position is defined by nine coordinates, coordinates 1 through 9.

The range in which the particle is allowed to move is determined in this case by the number of resources available to run the tasks. As a result, the value of a coordinate can range from 0 to the number of VMs in the initial resource pool. Based on this, the integer part of the value of each coordinate in a particle's position corresponds to a resource index and represents the compute resource assigned to the task defined by that particular coordinate. In this way, the particle's position encodes a mapping of task to resources. Following the example given in Fig. 3; there are three resources in the resource pool so each coordinate will have a value between 0 and 3. Coordinate 1 corresponds to task 1 and its value of 1.2 means that this task was assigned to resource 1. Coordinate 2 corresponds to task 2 and its value of 1.0 indicates that task 2 was assigned to resource 1. The same logic applies to the rest of the coordinates and their values.

Since the fitness function is used to determine how good a potential solution is, it needs to reflect the objectives of the scheduling problem. Based on this, the fitness function will be minimized and its value will be the total execution cost TEC associated to the schedule S derived from the particle's position. How this schedule is generated is explained later in this section.

Because of the elasticity and dynamicity of the resource acquisition model offered by IaaS providers, there is no initial set of available resources we can use as an input to the algorithm. Instead, we have the illusion of an unlimited pool of heterogeneous VMs that can be acquired and released at any point in time. Consequently, a strategy to define an initial pool of resources that the algorithm can use to explore different solutions and achieve the scheduling objective needs to be put in place.

Such strategy needs to reflect the heterogeneity of the VMs and give PSO enough options so that a suitable particle (i.e., solution) is produced. If this initial resource pool is limited, then so will be the resources that can be used to schedule the tasks. If it is very large, then the number of possible schedules becomes very large and so does the search space explored by PSO, making it difficult for the algorithm to converge and find a suitable solution.

A possible approach would be to project the illusion of unlimited resources into the algorithm by simulating a pool of VMs, one of each type for each task. Notice that at this stage, the algorithm is evaluating various solutions and therefore no VMs need to be actually leased; a simple representation of them is sufficient for the algorithm to work at this point. This strategy though, may result in a very large VM pool and hence a very large search space.

Instead, to reduce the size of the search space, we propose the following scheme. Let P be the set containing the maximum number of tasks that can run in parallel for a given workflow; then the initial resource pool R_{intial} that PSO will use to find a near-optimal schedule will be comprised of one VM of each type for each task in P. Our algorithm will then select the appropriate number and type of VMs to lease from this resource pool. In this way, we reflect the heterogeneity of the computing resources and reduce the size of the search space while still allowing the algorithm to execute all the tasks that can run in parallel to do so. The size of R_{intial} would then be equal to $|P| * n$ (where n is the number of available VM types) and thus, it is possible for PSO to select more than $|P|$ resources if required (unless $n = 1$).

As for the problem constraints, PSO was not designed to solve constrained optimization problems. To address this, we use a version of PSO that incorporates the constraint-handling strategy proposed by Deb et al. [20]. In such strategy, whenever two solutions are being compared, the following rules are used to select the better one. If both of the solutions are feasible, then the solution with better fitness is selected. If on the other hand, one solution is feasible and the other one is not, then the feasible one is selected. Finally, if both solutions are infeasible, the one with the smaller overall constraint violation is selected. The latter scenario implies that a measure of how much a solution violates a constraint needs to be in place. Our problem specifies a single constraint, meeting the application's deadline. Therefore, we define the overall constraint violation value of a solution to be the difference between the solution's makespan and the workflow's deadline. In this way, a solution whose makespan is closer to the deadline will be favored over a solution whose makespan is further away.

5.2 Schedule Generation

The pseudo-code to convert a particle's position into a schedule is shown in Algorithm 2. Initially, the set of resources to lease R and the set of task to resource mappings M are empty and the total execution cost $TEC$and time TET are set to zero. After this, the algorithm estimates the execution time of each workflow task on every resource $r_i \in R_{initial}$. This is expressed as a matrix in which the rows represent the tasks, the columns represent the resources and the entry $ExeTime[i, j]$ represent the time it takes to run task t_i on resource r_j. This time is calculated using Equation (1). The next step is the calculation of the data transfer time matrix. Such matrix is represented as a weighted adjacency matrix of the workflow DAG where the entry $TransferTime[i, j]$ contains the time it takes to transfer the output data of task t_i to task t_j. This value is calculated using Equation (2) and is zero whenever $i = j$ or there is no directed edge connecting $t_i$and t_j. An example of these matrices is shown in Fig. 4.

$$exeTime = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \end{array} \begin{array}{c} \begin{array}{ccc} r_1 & r_2 & r_3 \end{array} \\ \begin{bmatrix} 2 & 1 & 4 \\ 4 & 3 & 6 \\ 10 & 6 & 15 \\ 7 & 4 & 12 \\ 8 & 4 & 10 \\ 3 & 2 & 7 \\ 12 & 7 & 18 \\ 9 & 5 & 20 \\ 13 & 8 & 19 \end{bmatrix} \end{array}$$

(a)

$$transferTime = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 \end{array} \\ \begin{bmatrix} 0 & 9 & 9 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

(b)

Fig. 4. Sample execution and data transfer time matrix. (a) Execution time matrix. (b) Data transfer time matrix.

ALGORITHM 2
SCHEDULE GENERATION

Input: Set of workflow tasks T
Initial resource pool $\mathcal{R}_{initial}$
An array $pos[|T|]$ representing a particle's position
Output: A Schedule S

1. Initialize schedule components
   - 1.1. $\mathcal{R} = \emptyset, \mathcal{M} = \emptyset$
   - 1.2. $TEC = 0, TET = 0$
2. Calculate $ExeTime[|T| \times |R_{initial}|]$
3. Calculate $TransferTime[|T| \times |T|]$
4. **for** $i = 0$ to $i = |T| - 1$
   - 4.1. $t_i = T[i]$, $r_{pos[i]} = \mathcal{R}_{initial}[pos[i]]$
   - 4.2. **if** t_i has no parents
     $ST_{t_i} = LET_{r_{pos[i]}}$
     else
     $ST_{t_i} = max\left(max\left\{ET_{t_p}: t_p \in parents(t_i)\right\}, LET_{r_{pos[i]}}\right)$
     end if
   - 4.3. $exe = exeTime[i][pos[i]]$
   - 4.4. **for each** child t_c of t_i
     if t_c is mapped to a resource different to $r_{pos[i]}$
     $transfer += TransferTime[i][c]$
     end if
     end for each
   - 4.5. $PT_{t_i}^{r_{pos[j]}} = exe + transfer$
   - 4.6. $ET_{t_i} = PT_{t_i}^{r_{pos[j]}} - ST_{t_i}$
   - 4.7. $m_{t_i}^{r_{pos[i]}} = (t_i, r_{pos[i]}, ST_{t_i}, ET_{t_i})$
   - 4.8. $M = M \cup \{m_{t_i}^{r_{pos[i]}}\}$
   - 4.9. **if** $r_{pos[i]} \notin R$
     $LST_{r_{pos[i]}} = max(ST_{t_i}, bootTime)$
     $R = R \cup \{r_{pos[i]}\}$
     end if
   - 4.10. $LET_{r_{pos[i]}} = PT_{t_i}^{r_{pos[j]}} + LST_{r_{pos[i]}}$
5. Calculate *TEC* according to equation (4)
6. Calculate *TET* according to equation (5)
7. $S = (R, M, TEC, TET)$

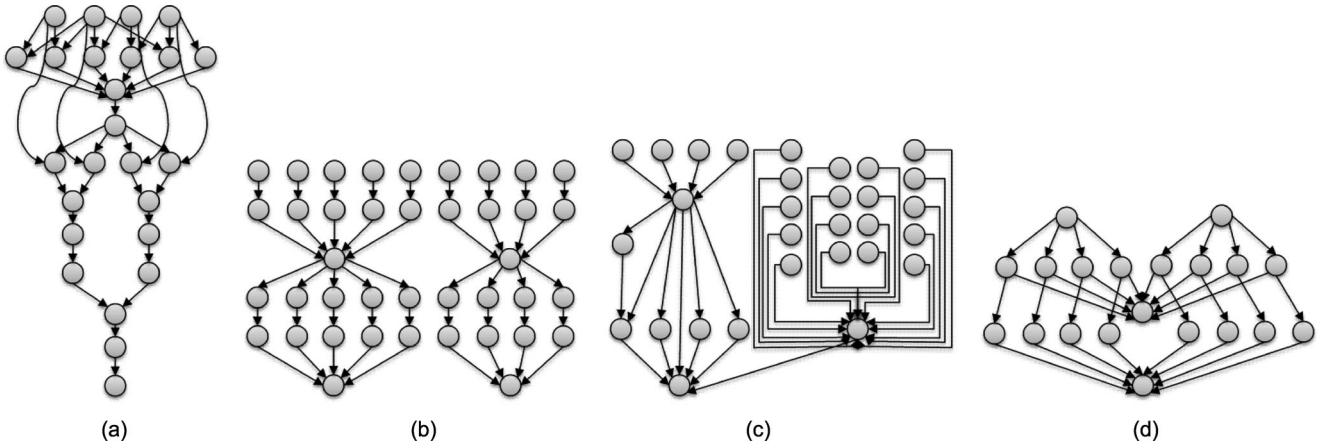

Fig. 5. Structure of the four different workflows used in the experiments. (a) Montage. (b) LIGO. (c) SIPHT. (d) CyberShake.

At this point the algorithm has all the information needed to begin decoding the particle's position and constructing the schedule. To achieve this, it iterates through every coordinate i in the position array pos and updates R and M as follows. First, it determines which task and which resource are associated to the current coordinate and its value. This is accomplished by using the encoding strategy depicted earlier, whi states that coordinate i corresponds to task t_i and its value $pos[i]$corresponds to resource $r_{pos[i]} \in R_{initial}$. Now that the first two components of a mapping tuple are identified, the algorithm calculates the value of the remaining two, the start ST_{t_i} and end ET_{t_i} times of the task.

The start time value ST_{t_i} is based on two scenarios. In the first case, the task has no parents and therefore it can start running as soon as the resource it was assigned to is available; this value corresponds to the current end of lease time of resource $r_{pos[i]}$, which is $LET_{r_{pos[i]}}$. In the second case, the task has one or more parents. In this situation, the task can start running as soon as the parent task that is scheduled to finish last completes its execution and the output data is transferred. However, if the resource is busy with another task at this time, the execution has to be delayed until such VM is free to execute t_i.

The value of ET_{t_i} is calculated based on the total processing time and the start time of the task. To determine the processing time $PT_{t_i}^{r_{pos[j]}}$ we first need to compute the execution and the data transfer times. The former is simply the value in $ExeTime[i, pos[i]]$ whereas the latter is computed by adding the values in $TransferTime[i, child(i)]$ for every child task $t_{child(i)}$ of t_i which is mapped to run in a resource different to $r_{pos[i]}$. These two values are then added to obtain $PT_{t_i}^{r_{pos[j]}}$ as defined in Equation (3). Finally we obtain the value of ET_{t_i} by subtracting ST_{t_i} from $PT_{t_i}^{r_{pos[j]}}$.

Now that we have computed all the elements of $m_{t_i}^{r_{pos[i]}} = (t_i, r_{pos[i]}, ST_{t_i}, ET_{t_i})$ we need to update two parameters associated to $r_{pos[i]}$ and add the resource to R if necessary. The first parameter is the time when the VM should be launched, $LST_{r_{pos[i]}}$. If the resource already exists in R then this means that it already has a start time and $LST_{r_{pos[i]}}$ does not need to be updated. However, if the resource is new and $t_i$is the first task to be assigned to it then R is updated so that it contains the new resource $r_{pos[i]}$ and $LST_{r_{pos[i]}}$ is set to be equal to either the start time of the task or the VM boot time, whichever is bigger. In this way, we account for VM boot time and do not assume that a resource is available to use as soon as it is requested. The second parameter we need to update is $LET_{r_{pos[i]}}$, this is the time when the resource is scheduled to finish executing the last task assigned to it and therefore is free to either run another task or be shutdown. The new lease end time of $r_{pos[i]}$ is thus the time it takes to process the task t_i $(PT_{t_i}^{r_{pos[j]}})$, plus the time when the resource is scheduled to start running, $LST_{r_{pos[i]}}$.

Once the algorithm finishes processing each coordinate in the position vector, R will contain all the resources that need to be leased as well as the times when they should be started and shutdown. Additionally, the entire task to resource mapping tuples will be in M and each task will have a resource assigned to it as well as an estimated start and end times. With this information, the algorithm can now use Equations (4) and (5) to compute the execution cost TEC and time TET associated to the current solution. After this, the algorithm has computed R, M, TEC and TET and therefore it can construct and return the schedule associated to the given particle's position.

Finally, Algorithms 1 and 2 are combined to produce a near optimal schedule. In step 3 of Algorithm 1, instead of calculating the fitness value of the particle, we generate the schedule as outlined in Algorithm 2. Then we use TEC as a fitness value in steps 4 through 6 and introduce the constraint handling mechanism in step 4, ensuring that TET doesn't exceed the application's deadline.

6 Performance Evaluation

In this section we present the experiments conducted in order to evaluate the performance of the proposed approach. We used the CloudSim framework [19] to simulate a cloud environment and chose four different workflows from different scientific areas: Montage, LIGO, SIPHT, and CyberShake. Each of these workflows has different structures as seen in Fig. 5 and different data and computational characteristics. The Montage workflow is an astronomy application used to generate custom mosaics of the sky based on a set of input images. Most of its tasks are characterized by being I/O intensive while not requiring

much CPU processing capacity. The LIGO workflow is used in the physics field with the aim of detecting gravitational waves. This workflow is characterized by having CPU intensive tasks that consume large memory. SIPHT is used in bioinformatics to automate the process of searching for sRNA encoding-genes for all bacterial replicons in the National Center for Biotechnology Information[1] database. Most of the tasks in this workflow have a high CPU and low I/O utilization. Finally, CyberShake is used to characterize earthquake hazards by generating synthetic seismograms and may be classified as a data intensive workflow with large memory and CPU requirements. The full description of these workflows is presented by Juve et al. [1].

When leasing a VM from an IaaS provider, the user has the ability to choose different machines with varying configurations and prices. The first variation of our algorithm, referred to as PSO, considers this heterogeneous environment. However, in order to evaluate if this resource heterogeneity has an impact on the makepsan and cost of a workflow execution, we propose a second variation of our algorithm called PSO_HOM. PSO_HOM considers a homogeneous environment in which only a single type of VM is used.

We used the IC-PCP [14] and SCS [12] algorithms as a baseline to evaluate our solution. As mentioned in Section 2, IC-PCP was designed for the same problem addressed in this paper: schedule a single workflow instance in an IaaS cloud whilst minimizing the execution cost and meeting the application's deadline. What is more, the algorithm considers many of the characteristics typical of IaaS resource models. For instance, the IC-PCP algorithm accounts for heterogeneous VMs which can be acquired on demand and are priced based on a predefined interval of time. It also considers data transfer times in addition to computation times of each task. However, IC-PCP does not account for VM startup time.

The IC-PCP algorithm begins by finding a set of tasks or critical paths associated to each exit node of the workflow (an exit node is defined as a node with no children tasks). The tasks on each path are scheduled on the same VM and are preferably assigned to an already leased instance which can meet the latest finish time requirements of the tasks. However, if this cannot be achieved, the cheapest instance that can finish the tasks before their latest finish time is leased and the path assigned to it. Finally, a critical path for each unassigned task on the scheduled path is calculated and the process is repeated until all tasks have been scheduled. At the end of this process, each task has been assigned to a VM and has a start and end times associated to it. Additionally, each VM has a start time determined by the start time of its first scheduled task and an end time determined by the end time of its last scheduled task.

SCS on the other hand, is a dynamic algorithm designed to schedule a group of workflows, instead of a single one. It can however be used to schedule a single instance without any modification. We chose this algorithm as it has the same scheduling objectives and considers the same cloud model as in this paper. What is more, it is of interest to compare the performance of a static approach versus a dynamic one.

TABLE 1
Type of VMs Used in the Experiments

Name	EC2 Units	Processing Capacity (MFLOPS)	Cost per Hour
m1.small	1	4400	$0.06
m1.medium	2	8800	$0.12
m1.large	4	17600	$0.24
m1.xLarge	8	35200	$0.48
m3.xLarge	13	57200	$0.50
m3.doubleXLarge	26	114400	$1.00

The SCS algorithm first identifies and bundles tasks with a one-to-one dependency into a single one. This is done in order to reduce data transfer times. After this, the overall deadline of the workflow is distributed over the tasks, with each task receiving a portion of the deadline based on the VM which is most cost-efficient for the task. The technique used to achieve this deadline assignment is done based on the work by Yu et al. [24]. The next step is to define a load vector for each VM type; this load vector indicates how many machines are needed in order for the tasks to finish by their assigned deadline at a given point in time. This value is calculated based on the execution interval derived from the deadline assignment phase and the estimated running time of a task on the specific instance type. Afterwards, the algorithm proceeds to consolidate partial instance hours by merging tasks running on different instance types into a single one if one of the VMs has idle time and can complete the additional task by its original deadline. Finally, earliest deadline first is used to map tasks onto running VMs; the task with the earliest deadline is scheduled as soon as an instance of the corresponding type is available.

We modeled an IaaS provider offering a single data center and six different types of VMs. The VM configurations are based on current Amazon EC2[2] offerings and are presented in Table 1. We used the work by Ostermann et al. [17] to estimate the processing capacity in MFLOPS based on the number of EC2 compute units. Each application was evaluated using three different workflow sizes, small, large, and extra large. Small workflows have 50 tasks on average whereas large ones have 100 and extra large ones 1,000.

We used a VM billing interval of 1 hour and a boot time of 97 seconds. We chose the latter value based on the results obtained by Mao and Humphrey [22] for Amazon's EC2 cloud. Each experiment was executed 20 times and the results displayed are those obtained for the large workflows.

One of the assumptions made by the evaluated algorithms is that the execution time of the tasks is known in advance; considering they target scientific workflows with well-known tasks and structures, it is reasonable to make such assumption and expect a quite accurate estimation. However, we acknowledge that such estimations are not 100 percent accurate and therefore introduce in our simulation a

1. National Center for Biotechnology Information http://www.ncbi.nlm.nih.gov/.

2. Amazon EC2. http://aws.amazon.com/ec2/.

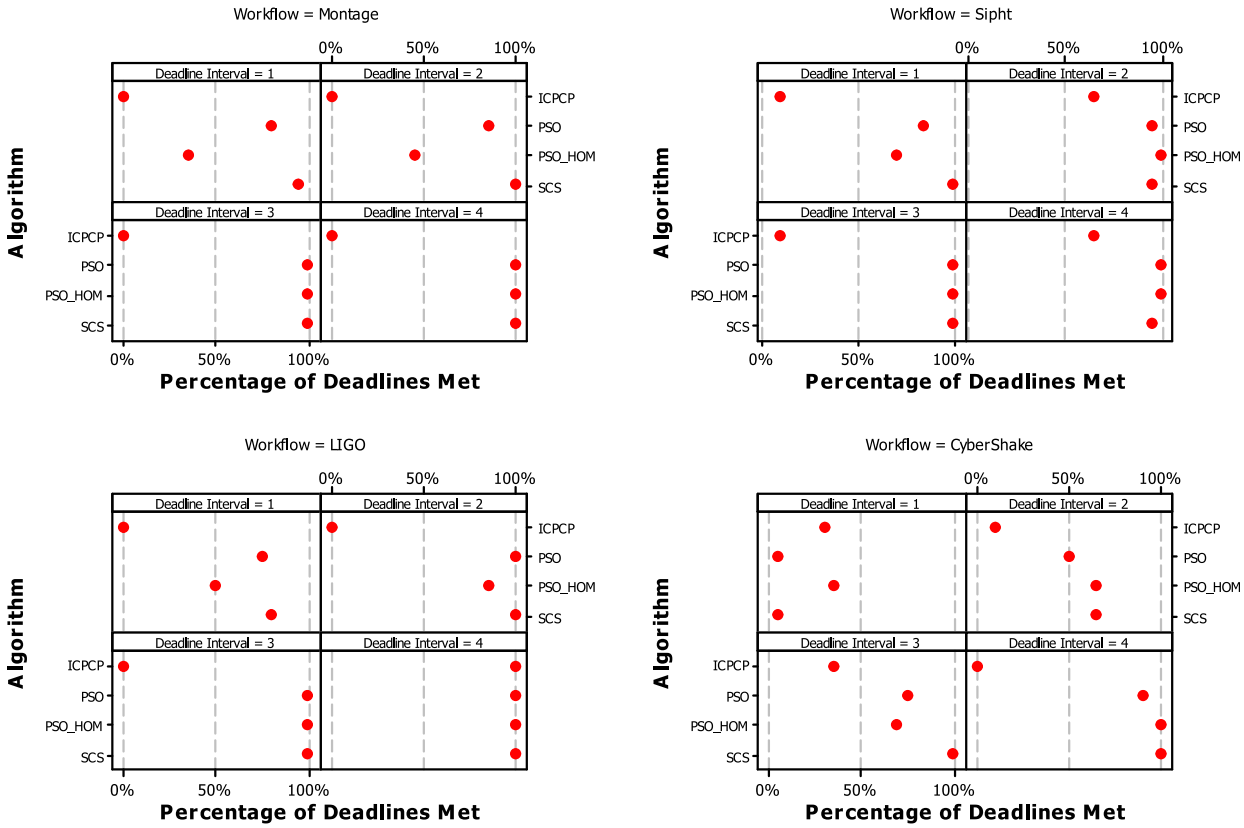

Fig. 6. Individual value plot of deadlines met for each workflow and deadline interval.

variation of ±10 percent to the provided task size based on a normal distribution.

Performance variation was modeled after Schad et al. [21] findings; the performance of each VM in the datacenter was diminished by at most 24 percent based on a normal distribution with mean 12 percent and standard deviation of 10 percent . In addition to this performance degradation, we modeled a data transfer variation of 19 percent [21]; the bandwidth available for each data transfer within the data center was subject to a degradation of at most 19 percent based on a normal distribution with mean 9.5 percent and a standard deviation of 5 percent.

The experiments were conducted using four different deadlines. These deadlines were calculated so that their values lie between the slowest and the fastest runtimes. To calculate these runtimes, two additional policies were implemented. The first one calculates the schedule with the slowest execution time; a single VM of the cheapest type is leased and all the workflow tasks are executed on it. The second one calculates the schedule with the fastest execution time; one VM of the fastest type is leased for each workflow task. Although these policies ignore data transfer times, they are still a good approximation to what the slowest and fastest runtimes would be. To estimate each of the four deadlines, the difference between the fastest and the slowest times is divided by five to get an interval size. To calculate the first deadline interval we add one interval size to the fastest deadline, to calculate the second one we add two interval sizes and so on. In this way we analyze the behavior of the algorithms as the deadlines increase from stricter values to more relaxed ones.

To select the best value for the PSO parameters c_1, c_2, and ω, we defined an artificial workflow with 100 tasks and ran the algorithm with different parameter values. The values of c_1 and c_2 were varied from 1.5 to 2.0 and ω ranged from 0.1 to 1.0. To compare the results we considered the average workflow execution cost after running each experiment 10 times. We found the impact of c_1 and c_2 to be negligible. The inertia value ω had a slightly higher impact with the lowest average cost obtained with a value of 0.5. Based on this we define $c_1 = c_2 = 2.0$ and $\omega = 0.5$ and use these parameter values in the rest of our experiments. The number of particles was set to 100.

6.1 Results and Analysis

6.1.1 *Deadline Constraint Evaluation*

To analyze the algorithms in terms of meeting the user defined deadline, we plotted the percentage of deadlines met for each workflow and deadline interval. The results are displayed in Fig. 6. For the Montage workflow, ICPCP fails to meet all of the deadlines. PSO_HOM meets fewer than 50 percent of the deadlines on interval 1 but improves its performance on interval 2 and achieves a 100 percent hit rate for both intervals 3 and 4. PSO and SCS are the best performing algorithms in terms of deadlines met with PSO meeting slightly less deadlines on intervals 1 and 2 but achieving 100 percent on the last two intervals.

The results for the SIPHT application again show that ICPCP fails to meet the most number of deadlines and its performance is significantly worse than that of the other

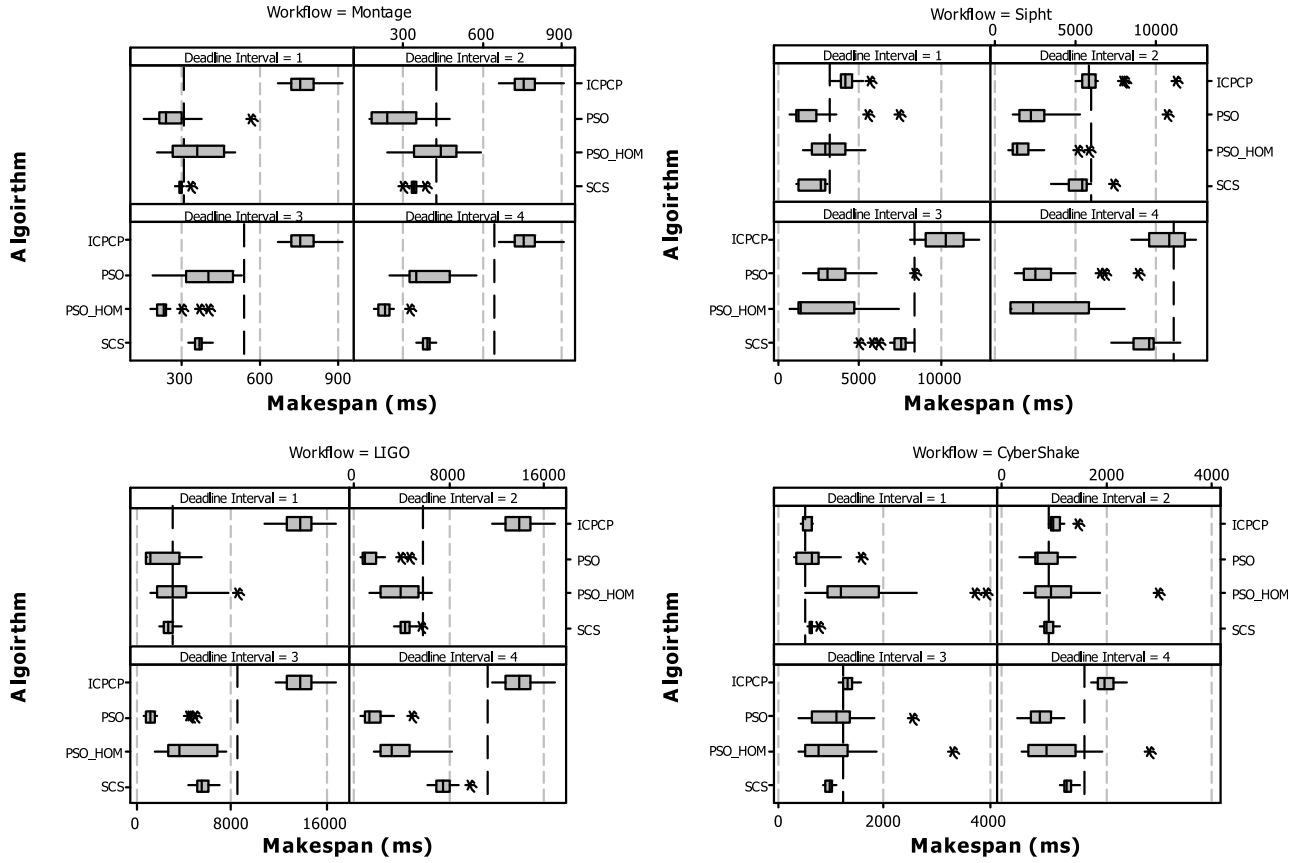

Fig. 7. Boxplot of makespan by algorithm for each workflow and deadline interval. The reference line on each panel indicates the deadline value of the corresponding deadline interval.

algorithms. SCS, PSO and PSO_HOM all meet the deadline over 95 percent of the times for intervals 2, 3 and 4.

The results obtained for the LIGO workflow and the SCS, PSO and PSO_HOM algorithms are very similar to those obtained for the SIPHT workflow. IC-PCP on the other hand is unable to meet any of the first three deadlines with a 0 percent hit rate. Its performance improves considerably for the 4th interval where it achieves a 100 percent hit rate.

As for the CyberShake workflow, IC-PCP meets the least amount of deadlines with the highest percentage being 30 percent on deadline interval 3. For deadline interval 1, SCS and PSO have the lowest percentages; however, as opposed to IC-PCP, SCS and the PSO based algorithms perform better as the deadline becomes more relaxed. The performance of PSO, PSO_HOM and SCS is similar from deadline interval 2 onwards. Overall, IC-PCP is outperformed by the other three algorithms. The percentage of deadlines met by this algorithm greatly differs from its counterparts and is in most cases under 50 percent. A possible explanation for this is the fact that IC-PCP fails to capture the dynamicity of the cloud by ignoring performance variation. Another feature that is not considered by the algorithm is the VM startup time, delay which is not negligible and might have a significant impact on the schedule, especially when a large number of VMs are needed to meet the specified deadline. SCS on the other hand, has a 100 percent hit rate in most of the cases. This are the results expected of an online algorithm as it was designed to adapt to the conditions of the cloud to ensure the deadlines are met. Both PSO and PSO_HOM have a very similar performance to SCS having a 100 percent hit rate in most of the cases. Even though our approach is offline, as is IC-PCP, we succeed in considering the dynamic nature of the cloud and the variability of CPU performance. Overall, PSO, PSO_HOM and SCS meet the most number of deadlines for all of the workflows, making them the most appealing algorithms for the scheduling problem stated in this paper. There is a considerable difference in the percentage of deadlines met by IC-PCP and these three algorithms.

6.1.2 Makespan Evaluation

The values obtained for the makespan of each of the workflows are displayed in Fig. 7. The dotted line on each panel of each graph corresponds to the deadline value for the given interval. Evaluating the makespan with regards to this value is essential as all of the algorithms were designed to meet the given deadline. For the LIGO and Montage workflows, IC-PCP fails to meet this goal by producing schedules which take longer time to execute on average than the workflow's deadline. In fact, for the four deadline intervals, the Q1 (first quartile), median and Q3 (third quartile) values obtained with IC-PCP are considerably higher than the deadline. For the LIGO workflow, both of the PSO approaches and SCS have medians well below the deadline value for the four intervals and Q3 values smaller than the deadline on intervals 2, 3 and 4. This means that in at least 75 percent of the cases, they are able to produce schedules that finish on time. What is more, PSO has the lowest

average makespans on all the intervals followed by PSO_HOM for intervals 2 through 4. The results are similar for the Montage workflow.

In the Sipht workflow case, PSO and PSO_HOM have the lowest average makespans while IC-PCP has the highest on every case. SCS has higher average makespans than both of the PSO approaches. PSO, PSO_HOM and SCS have median values lower than the deadline in every case and for deadline intervals 2 through 4 they have lower Q3 values.

IC-PCP performs better with the CyberShake workflow. It seems to have the best performance for deadline interval 1 with an average execution time close to the deadline. However, PSO performs similarly with a slightly higher median. For deadline interval 2, IC-PCP has the hi-guest Q1 value which is also higher than the deadline. PSO generates the fastest schedules on average with the lowest Q1, median and Q3 values. The results are similar for the deadline interval 3 but SCS has a lower Q3 value and less variation than PSO and PSO_HOM, resulting in a smaller average makespan. The average execution time for IC-PCP on deadline interval 4 is above the deadline value, while both of the PSO based approaches and SCS have Q3 values smaller than the deadline.

These results are in line with those analyzed in the deadline constraint evaluation section, from which we were able to conclude that IC-PCP is not very efficient in meeting the deadlines whereas the other three heuristics are. For the LIGO and Montage workflows, the difference is substantial on every case. For the CyberShake and SIPHT workflows on the other hand, when larger than the deadline, the average IC-PCP makespan is only slightly larger than the deadline. Furthermore, IC-PCP and SCS being heuristic based algorithms are much more predictable in the sense that the execution time does not vary much from run to run. PSO_HOM and PSO on the contrary exhibit a larger makespan variation, which is expected as it is a meta-heuristic based approach with a very large search space.

6.1.3 Cost Evaluation

The average execution costs obtained for each workflow are shown in Fig. 8. We also show the mean makespan as the algorithms should be able to generate a cost-efficient schedule but not at the expense of a long execution time. The reference line on each panel displaying the mean makespan is the deadline corresponding to the given deadline interval. We present this as there is no use in an algorithm generating very cheap schedules but not meeting the deadlines; the cost comparison is made therefore, amongst those heuristics which managed to meet the particular deadline in a give case.

For the Montage workflow, IC-PCP execution costs are the lowest ones for the four deadline intervals but its execution times are on average much higher than each of the four deadlines. Amongst the algorithms that do comply with the deadline constraint, PSO obtains the lowest cost on deadline interval 1; PSO_HOM on deadline interval 2 and finally, PSO for deadline intervals 3 and 4. From the results, it is clear that the PSO based strategies perform better than SCS in terms of cost by generating much cheaper schedules. The differences in costs are very pronounced for this workflow, with the costs obtained by the algorithms being significantly different to each other, especially when comparing the cheapest and the most expensive one. A possible explanation for this might be the fact that heuristics such as SCS and PSO lease more VMs in order to meet the deadline; this, combined with the fact that the Montage tasks are relatively small, means that the machines are only used for a small amount of time but charged for the full billing period. The reason for IC-PCP generating schedules with such a low cost and large makespan might be accounted to it not considering VM performance variation or boot time when estimating the schedule times, causing these to greatly differ from the actual execution ones.

The results for SIPHT show that our solution has the best performance in the first three deadline intervals; it achieves the lowest costs while having an average makespan smaller than the deadline. IC-PCP exhibits the lowest cost for the four deadlines; however, the average makespan obtained by this algorithm is higher than the deadline value for the first three intervals. Hence, IC-PCP outperforms the other heuristics with the most relaxed deadline but, on average, fails to produce schedules that meet the three tighter deadlines. The other three algorithms succeed on meeting the deadline constraint on average in every case; this is supported by the boxplot of makespan depicted in Fig. 7. Since SCS, PSO and PSO_HOM all meet the deadline; the best performing algorithm is that capable of generating the schedule that leads to the lowest cost. For deadline interval 1, PSO_HOM achieves this, for deadline interval 2 and 3 PSO does, and finally, as stated before, IC-PCP has the lowest cost on deadline interval 4. Overall, both PSO and SCS are capable of meeting the imposed deadline; however, our solution with a bigger window between the makespan and deadline and a lower cost and SCS closer to the deadline and at a higher price.

For the CyberShake application, the results for the IC-PCP algorithm show a considerably higher cost when compared to the other algorithms for deadline interval 1. In this particular scenario, IC-PCP has the lowest average makespan, with it being only slightly higher than the deadline value. The other algorithms on the other hand have lower costs but at the expense of having longer execution times which do not comply with the deadline constraint. PSO and SCS perform similarly in this particular scenario and obtain only slightly higher average makespans than IC-PCP but at much lower costs. For deadline intervals 2, 3 and 4, IC-PCP generates cheap schedules but it fails to meet each of the specified deadlines. PSO has the lowest average cost for deadline intervals 2, 3 and 4, making it the most efficient algorithm by meeting the deadlines at the lowest cost. Aside from deadline interval 1, all algorithms have very similar results in terms of cost for this application; PSO outperforms all the other heuristics on intervals 2, 3 and 4 by not only generating the cheapest schedule but also the fastest one in some of the cases.

The performance of IC-PCP for the LIGO workflow is the same as for the SIPHT application; it achieves the lowest average cost in every case but produces the schedules with the longest execution times, which are well above the deadline value for the four intervals. PSO and SCS on the other hand meet on average the deadline on every case with PSO

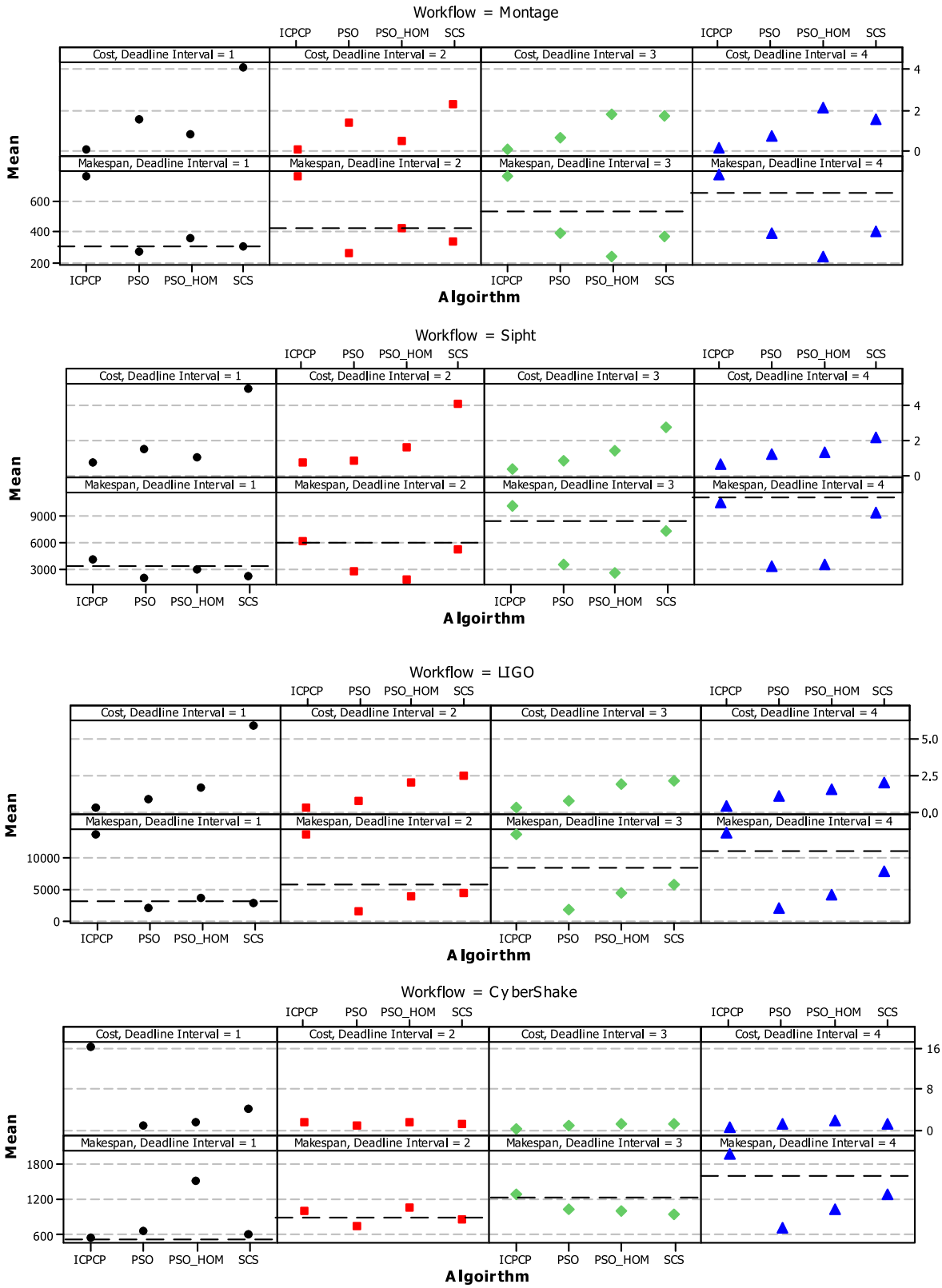

Fig. 8. Lineplot of mean makespan (ms) and mean cost (USD) for each workflow and deadline interval. The reference line on each panel indicates the deadline value of the corresponding deadline interval.

producing the most efficient schedules with shorter makespans and lower prices.

Overall, we found that IC-PCP is capable of generating low cost schedules but fails to meet the deadline in these cases. SCS is very efficient when generating schedules that meet the deadline but because it is a dynamic and heuristic based approach, its cost optimization is not as good as that obtained by our solution. We found that in every case in which both algorithms meet the deadline, our approach incurs in cheaper costs, in some cases generating not only

cheaper but faster solutions. As expected, PSO performs better than PSO_HOM.

6.1.4 Further Analysis

The results found for the small and extra large workflows were similar to the ones presented here, although, our approach seems to perform better with smaller sized workflows which is reasonable as bigger workflows mean larger search spaces. We also found that in some cases our solution tends to generate schedules with lower makespans and higher costs as long as the deadline is met. Some users might prefer this as a solution whereas others might prefer the makespan to be closer to the deadline and pay a lower price. As future work, experiments will be held to analyze the performance of the scheduling algorithm in these scenarios using different optimization techniques such as genetic algorithms.

Another finding is the significant impact that the selection of the initial pool of resources has on the performance of the algorithm. The same set of experiments was conducted with an initial VM pool composed of one VM of each type for each task. The results show that the algorithm takes longer to converge and find an optimal solution producing schedules with higher execution times and costs. We noticed this strategy leads the algorithm into leasing more VMs with lower utilization rate and hence incurring in higher costs and more delays such as startup and data transfer times.

Overall, in most of the cases, IC-PCP fails to meet the deadlines whereas our approach and SCS succeed; the reason why IC-PCP fails to meet so many deadlines might be accounted to the fact that it does not consider VM boot time or performance variation. When compared to SCS, our algorithm is capable of generating cheaper schedules and hence outperforms it in terms of cost optimization.

Regarding the computational complexity of the algorithm, in each PSO iteration, the position and velocity of all particles is updated and their fitness is evaluated. The number of particles N and their dimension D determine the number of calculations required to update the position and velocity of particles. The fitness function complexity is based on the schedule generation algorithm and depends on the number of tasks T, and the number of resources being used R. Based on this and the fact that $D = T$ in our formulation, the proposed algorithm has an overall complexity of order $O(N * T^2 * R)$ per iteration. The convergence time is also influenced by the number of tasks and compute resources. IC-PCP and SCS being heuristic based, run much faster than our meta-heuristic based proposal. While IC-PCP and SCS have a polynomial time complexity, PSO has an exponential one. However, considering that our solution is designed to generate an offline schedule for a single workflow, the high time complexity of PSO is acceptable and the benefits in terms of better schedules outweigh this disadvantage.

7 Conclusions and Future Work

In this paper we presented a combined resource provisioning and scheduling strategy for executing scientific workflows on IaaS clouds. The scenario was modeled as an optimization problem which aims to minimize the overall execution cost while meeting a user defined deadline and was solved using the meta-heuristic optimization algorithm, PSO. The proposed approach incorporates basic IaaS cloud principles such as a pay-as-you-go model, heterogeneity, elasticity, and dynamicity of the resources. Furthermore, our solution considers other characteristics typical of IaaS platforms such as performance variation and VM boot time.

The simulation experiments conducted with four well-known workflows show that our solution has an overall better performance than the state-of-the-art algorithms, SCS and IC-PCP. In every case in which IC-PCP fails to meet the application's deadline, our approach succeeds. Furthermore, our heuristic is as successful in meeting deadlines as SCS, which is a dynamic algorithm. Also, in the best scenarios, when our heuristic, SCS and IC-PCP meet the deadlines, we are able to produce schedules with lower execution costs.

As future work, we would like to explore different options for the selection of the initial resource pool as it has a significant impact on the performance of the algorithm. We would also like to experiment with different optimization strategies such as genetic algorithms and compare their performance with PSO. Another future work is extending the resource model to consider the data transfer cost between data centers so that VMs can be deployed on different regions. Extending the algorithm to include heuristics that ensure a task is assigned to a VM with sufficient memory to execute it will be included in the algorithm. Finally, we aim to implement our approach in a workflow engine so that it can be utilized for deploying applications in real life environments.

Acknowledgments

The authors would like to thank Dr. Rodrigo Calheiros, Mohammed Alrokayan and Deepak Poola for their comments on improving this paper.

References

[1] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2012.

[2] P. Mell, T. Grance, "The NIST definition of cloud computing—recommendations of the National Institute of Standards and Technology" *Special Publication 800-145*, NIST, Gaithersburg, 2011.

[3] R. Buyya, J. Broberg, and A. M. Goscinski, Eds., *Cloud Computing: Principles and Paradigms*, vol. 87, Hoboken, NJ, USA: Wiley, 2010.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. 6th IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[5] Y. Fukuyama and Y. Nakanishi, "A particle swarm optimization for reactive power and voltage control considering voltage stability," in *Proc. 11th IEEE Int. Conf. Intell. Syst. Appl. Power Syst.*, 1999, pp. 117–121.

[6] C. O. Ourique, E. C. Biscaia Jr., and J. C. Pinto, "The use of particle swarm optimization for dynamical analysis in chemical processes," *Comput. Chem. Eng.*, vol. 26, no. 12, pp. 1783–1793, 2002.

[7] T. Sousa, A. Silva, and A. Neves, "Particle swarm based data mining algorithms for classification tasks," *Parallel Comput.*, vol. 30, no. 5, pp. 767–783, 2004.

[8] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the NP-Completeness*, vol. 238, New York, NY, USA: Freeman, 1979.

[9] M. Rahman, S. Venugopal, and R. Buyya, "A dynamic critical path algorithm for scheduling scientific workflow applications on global grids," in *Proc. 3rd IEEE Int. Conf. e-Sci. Grid Comput.*, 2007, pp. 35–42.

[10] W. N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Trans. Syst., Man, Cybern., Part C: Appl. Rev.*, vol. 39, no. 1, pp. 29–43, Jan. 2009.
[11] J. Yu and R Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in *Proc. 1st Workshop Workflows Support Large-Scale Sci.*, 2006, pp. 1–10.
[12] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 1–12.
[13] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2012, vol. 22, pp. 1–11.
[14] S. Abrishami, M. Naghibzadeh, and D. Epema, "Deadline-constrained workflow scheduling algorithms for IaaS clouds," *Future Generation Comput. Syst.*, vol. 23, no. 8, pp. 1400–1414, 2012.
[15] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc. IEEE Int. Conf. Adv. Inform. Netw. Appl.*, 2010, pp. 400–407.
[16] Z. Wu, Z. Ni, L. Gu, and X. Liu, "A revised discrete particle swarm optimization for cloud workflow scheduling," in *Proc. IEEE Int. Conf. Comput. Intell. Security*, 2010, pp. 184–188.
[17] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Cloud Computing*, Berlin, Germany: Springer, 2010 pp. 115–131.
[18] A. Lazinica, Ed. *Particle Swarm Optimization*. Rijeka, Croatia: InTech, 2009.
[19] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw.: Practice Experience*, vol. 41, no. 1, pp. 23–50, 2011.
[20] K. Deb, A. Pratap, S. Agarwal, and T. A. M. T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[21] J. Schad, J. Dittrich, and J. A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 460–471, 2010.
[22] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proc. 5th IEEE Int. Conf. Cloud Comput.*, Jun. 2012, pp. 423–430.
[23] E. K. Byun, Y. S. Kee, J. S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Comput. Syst.*, vol. 27, no. 8, pp. 1011–1026, 2011.
[24] J. Yu, R. Buyya, and C. Tahm, "A cost based scheduling of scientific workflow applications on utility grids," in *Proc. 1st IEEE Int. Conf. e-Sci. Grid Comput.*, 2005, pp. 140–147.

Maria Alejandra Rodriguez received the bachelor's degree in computer science from Andes University, Colombia, in 2008 and the master's degree in distributed computing from the University of Melbourne, Australia. She is currently working toward the PhD degree at the CLOUDS laboratory in the Computing and Information Systems Department, The University of Melbourne, Australia. Her research interest includes distributed systems and scientific computing.

Rajkumar Buyya is currently a professor and a future fellow of the Australian Research Council, and the director of the Cloud Computing and Distributed System (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored more than 450 publications and four text books including *Mastering Cloud Computing* published by McGraw Hill and Elseiver/Morgan Kaufman, 2013 for Indian and international markets, respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index = 79, g-index = 160, 29,600 + citations).