Resourse Provisioning and Scheduling Algorithm in Cloud

Tukaram H. Pawar¹

¹Master Student, Department of Computer Engineering, SSVPS’S B.S.Deore College of Engineering, India

Abstract
Cloud Computing is the latest and emerging trend in information technology domain. It offers utility-based IT services to user over the Internet and tremendous opportunities to solve large-scale scientific problems. Workflow scheduling is one of the major problems in cloud systems and it presents various challenges that need to be addressed in order to be efficiently utilized for workflow applications. A good scheduling algorithm must minimize the execution time and cost of workflow application along with QoS requirements of the user. Although the workflow scheduling problem has been widely studied, there are very few initiatives made to order 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. The system consider deadline as the major constraint and propose a score based deadline constrained workflow scheduling algorithm that executes workflow within manageable cost while meeting user defined deadline constraint and resource provisioning strategy as a service (IaaS) clouds. 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 experimental results show that proposed algorithm exhibits less execution time and also reduces the failure rate of workflow application within manageable cost which is better than the current state-of-the-art algorithms.

General Terms - Cloud Computing, Deadline, Scheduling, Scientific Workflow

Keywords - Cloud computing, resource provisioning, scheduling, scientific workflow, deadline constraint, QoS.

INTRODUCTION
Workflow 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 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. 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. 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. 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.

RELATED WORK
Workflow scheduling on distributed systems has been usually planned over the years and is NP-hard by a reduction from the multiprocessor scheduling problem [3]. 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 considerable range of the proposed solutions target locations 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. propose a solution based on the workflow’s dynamic critical paths, Chen and Zhang 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 [4].

In 2008, Yun Yang et al focused on scheduling for cost constrained transaction intensive cloud workflows by considering the characteristics of cloud computing. These types of workflows have large number of workflow instances which are bounded by budget for execution. The purpose of this algorithm is to minimize the cost under certain user-designated deadlines. The algorithm has characteristics as, they consider the pay per use feature of cloud workflow, enables the compromising of execution cost and time [9].

Lin and Lu [5] proposed SCPOR, a scientific workflow scheduling algorithm that is able to schedule workflows in need of elastically changing compute resources. But it misses the capability of considering the cost for utilization of resources.

To schedule the scientific workflows in Software as a Service cloud S. Abrishami et al proposed a new algorithm based on the partial critical path in 2012. This algorithm tries to minimize the workflow execution cost by meeting its deadline. This algorithm first schedules the critical path of the workflow and then finds the partial critical path to each task on critical path [3].

In 2013, Salid Abrshami, Mahmoud Naghibzadeh and Dick H. J. E pema proposed Deadline-Constrained Workflow Scheduling Algorithm for Infrastructure Service. In this paper, execution time is minimized by maintaining the user defined deadline. The author implements two different algorithms based on PCP. First is Cloud Partial Critical Path and second with Deadline Distribution [4]. The disadvantage of this algorithm is that they didn”t consider the data transfer time during provisioning and scheduling.

The author Rajkumar buyya et al proposed a combine resource provisioning and scheduling for scientific workflow execution. They used Meta heuristic optimization algorithm, Particle Swarm Optimization to minimize the execution cost. This algorithm performs better for smaller sized
In order to reduce the limitations of previous algorithms, proposed algorithm takes replication of tasks in idle time of virtual machine such that the tasks are complete its execution before deadline. Our work is also different from cloud-targeted autoscaling solution in that it consider ensembles of workflows rather than unpredictable workloads containing workflows. it also consider budget constraints rather than cost minimization as a goal. In other words, it assume that there is more work to be done than the available budget, so some work must be rejected. Therefore, cost is not something it optimize (i.e. an objective), but rather a constraint.

This work is related to bi-criteria scheduling and multicriteria scheduling of workflows. These approaches are similar to ours in that it will have two scheduling criteria: cost and makespan. The challenge in multi-criteria scheduling is to derive an objective function that takes into account all of the criteria. In our case one objective (amount of work completed) is subject to optimization, whereas time and cost are treated as constraints. Other approaches use metaheuristics that usually run for a long time before producing good results, which makes them less useful in the scenarios consider in this system. Our work can also be regarded as an extension of the budget-constrained workflow scheduling [20] in the sense that we are dealing with workflow ensembles and the deadline constraint is added.

ANALYSIS OF PROBLEM

Existing research in execution of scientific workflows in Clouds either try to minimize the workflow execution time ignoring deadlines and budgets. Typical Cloud environments do not present regular performance in terms of execution and data transfer times. This is caused by technological and strategic factors and can cause performance variation of execution times and data transfer time. Problem it analyse with the existing system are a parent tasks and child tasks dependency rules, Impossible to find optimal solution in polynomial time; focus on near optimal ones and transfer time affects execution time

A workflow application $W = (T, E)$ is modeled as a DirectedAcyclic 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 $t_j$.

Each workflow $W$ has a deadline $\delta_w$ associated to it.

A deadline is defined as a time limit for the execution of the workflow. The IaaS Cloud provider offers a range of VM types. A VM type $VM_i$ is defined in terms of its processing capacity $PVM_i$ and cost per unit of time $CVM_i$. 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]. The unit of time $\tau$ 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 2 periods of 60 minutes, that is, 120 minutes.
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 \( S = (R, M, TET, TEC) \) in terms of a set of resources, a task to resource mapping, the total execution cost and the total execution time. Based on the previous definitions, the problem can be formally defined as follows: find a schedule \( Y \) with minimum \( TEC \) and for which the value of \( TET \) does not exceed the workflow’s deadline.

**PROPOSED WORK**

System architecture of the proposed system. User has to provide the XML file representing Directed Acyclic Graph (DAG) structure of workflow. DAX files contains list of tasks, dependencies between tasks, their computation time and size of the input and output files generated by the tasks. The DAX file contains information about task as task id, its runtime and name of the task. The tasks are extracted from the XML file and partial critical path is obtained. It is provided as an input to the first step of EIPR algorithm. Optimized schedule is then found and replication precedence of task is optimized. Lastly based on the optimized schedule, tasks are submitted to the VMs in cloud.

**PARTICLE SWARM OPTIMIZATION**

Particle Swarm Optimization (PSO) 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. In each iteration, the position and velocity of a particle are updated based in Equations (7) and (8) respectively.
PSO MODELLING

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 2-dimensional particle is specified by 2 coordinates, the position of a 3-dimensional one is specified by 3 coordinates and so on. As an example, the particle depicted in Figure 3 represents a workflow with 9 tasks; the particle is a 9-dimensional one and its position is defined by 9 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. 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 associated to the schedule derived from the particle’s position. How this schedule is generated is explained later in this section.

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.
SCHEDULE GENERATION

Initially, the set of resources to lease $Z$ and the set of task to resource mappings $6$ are empty and the total execution cost $\$&;2$ and time $\$&$ are set to zero. After this, the algorithm estimates the execution time of each workflow task on every resource $2[0 \in Z-0-0F0Ua]$. This is expressed as a matrix in which the rows represent the tasks, the columns represent the resources and the entry $\&k/\$BA/\[B, p]$ represent the time it takes to run task $0$ on resource $2[1]$. 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 $\$jCE/\/[\$BA/\[B, p]$ contains the time it takes to transfer the output data of task $0$ to task $2)$. 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 $B$ in the position array $2\square E$ and updates $Z$ and $262$ as follows. Firstly, 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, which states that coordinate $B$ corresponds to task $0$ and its value $2\square E[\[B]2$ corresponds to resource $22[aPc[0] \in Z-0-0F0Ua]$. Now that the first two components of a mapping tuple are identified, the algorithm calculates the value of the remaining two, the start and end times of the task. The start time value $Y$ 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 $2[aPc[0]$, which is $2$ and the end of the current time. 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 $2$ . The value of $&ST$: is calculated based on the total processing time and the start time of the task. To determine the processing time $7$: we first need to compute the execution
and the data transfer times. The former is simply the value in &k/$BA/[B, ∃E[B]] whereas the latter is computed by adding the values in $[jCE{l}$BA/[B, qη Bz<(B)] for every child task )in0ao(0) of )0 which is mapped to run in a resource different to [aPc[0]. These two values are then added to obtain 7SF: leee[G] a defined in Equation (3). Finally we obtain the value of &SF: by subtracting YSF: from 7SF: leee[G] Once the algorithm finishes processing each coordinate in the position vector, Z 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 6 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 $&; and time $&$ 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 $&; as a fitness value in steps 4 through 6 and introduce the constraint handling mechanism in step 4, ensuring that $&$ doesn’t exceed the application’s deadline.

**SIMULATION ON CLOUDSIM**

CloudSim is a new open source toolkit developed using java that generalized, and advanced simulation framework allows simulation of Cloud computing and application services. CloudSim is a simulation tool for creating cloud computing environment and used as the simulator in solving the workflow scheduling problem. CloudSim allows us to create a data center with a set of hosts and number of virtual machines as resources. Each task of a workflow can be assigned to appropriate virtual machine once it’s all parent tasks get executed.

**Simulation Description**

Virtual Machine - It is implemented virtual software of a computer that executes application programs same as a physical machine. Cloudlet - Cloudlet is input job or set of tasks to be executed in cloud environment. Cloudlet has its own unique Cloudlet_id and Cloudlet_length. The result analysis was conducted on Dell PC with 2.0 GHz Intel i3 CPU and 4 GB of memory running windows 8 and CloudSim. Cloudsim is used to construct six virtual machines in single data center. The XML files of workflow are given as Input to algorithm. These workflows contain the number of tasks, and these tasks are provided for scheduling. Result evaluation is done based on the execution time required for different techniques such as scheduling without replication, with replication, increasing budget for replication, and by changing the order for replication. Table 1 shows the execution time required by four different scientific workflows. In this table Without Rep denotes the execution of workflow without performing replication step. Bud=10 denotes Replication Budget and Rank Change denotes the execution of workflow with change in order of replication. Table 2 shows the execution cost required by the four different scientific workflows.

In this section we will describe processes for Load balancing using bays therom for load balancing which is used to deploy task requests received by the cloud data center into optimal target physical hosts in the IaaS cloud computing data center. Its algorithm is what combines Bayes theorem with clustering. It has achieved the overall load balancing of the entire network from the perspective of cloud data centers’ Figure 1 shows the working flow of the proposed system.

**EXPERIMENTAL RESULTS**

**Deadline Constraint Evaluation**

To understand the algorithms in terms of meeting the user defined deadline, we consider the percentage of deadlines met for each workflow and deadline interval. The results are displayed in Figure 6. For the Montage workflow, ICPCP fails to meet all of the deadlines. PSO_HOM meets
fewer than 50% of the deadlines on interval 1 but improves its performance on interval 2 and achieves a 100% 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% on the last two intervals. The results for the SIPHT application again show that IC-PCP fails to meet the most number of deadlines and its performance is significantly worse than that of the other algorithms. SCS, PSO and PSO_HOM all meet the deadline over 95% 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 3 deadlines with a 0% hit rate. Its performance improves considerably for the 4th interval where it achieves a 100% hit rate. As for the CyberShake workflow, IC-PCP meets the least amount of deadlines with the highest percentage being 30% 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%. 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% hitrate 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% 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

**Makespan Evaluation**

The values obtained for the makespan of each of the workflows are displayed in Figure 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 ICPCP 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% 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 Sipt 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 differences are 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.

Cost evaluation

The average execution costs obtained for each workflow are shown in Figure 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 given 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 3 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 Figure 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 ICPCP 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 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.

CONCLUSION
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 wellknown 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.

REFERENCES


