DEADLINE BASED RESOURCE PROVISIONING AND SCHEDULING ALGORITHM FOR SCIENTIFIC WORKFLOWS ON CLOUDS: A REVIEW

Supriya S. Bichkule¹, R. V. Mante²

Master of Technology scholar, Department of Computer Science and Engineering, Government College of Engineering, Amravati, Amravati(MH), India-444604, supriyabichkule@gmail.com

Assistant Professor, Department of Computer Science and Engineering, Government College of Engineering, Amravati, Amravati(MH), India-444604, mante.ravi@gmail.com

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. It is 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 this 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 modelled 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 those different user defined qualities 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 make-span (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.

This paper work is based on the meta-heuristic optimization technique, particle swarm optimization (PSO). PSO was first introduced by Kennedy and Ebehart in [3] and is inspired on the social behaviour 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 [3] 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 [4], pattern recognition [5] and data mining [6], among others.

This paper develop a static cost-minimization, deadline-constrained heuristic for scheduling a scientific workflow application in a cloud environment. This 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 modelled 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. This paper introduces, 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.

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 [6]. 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. [7] propose a solution based on the workflow’s dynamic critical paths, Chen and Zhang [8] 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 [9].

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 [10]. 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. [11]. 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.

While the algorithms presented by Mao and Humphrey [10] and Malawski et al. [11] 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. [12] 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.

This paper has used PSO to solve the workflow scheduling problem. Pandey et al. [13] 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. Wu et al. [14] 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. [13], it assumes an initial set of VMs is available beforehand and hence lacks in utilizing the elasticity of IaaS clouds.

3. PROPOSED APPROACH

3.1 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, 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 \( R_{\text{initial}} \) 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 \( R_{\text{initial}} \) 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, it uses a version of PSO that incorporates the constraint-handling strategy proposed by Deb et al. [15]. 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. This problem specifies a single constraint, meeting the application’s deadline. Therefore, it defines the overall constraint violation value of a solution to be the difference between the solution’s make-span and the workflow’s deadline. In this way, a solution whose make-span is closer to the deadline will be favored over a solution whose make-span is further away.

### 3.2 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 \( r_i \in R_{\text{initial}} \). This is expressed as a matrix in which the rows represent the tasks, the columns represent the resources and the entry \( \text{ExeTime}[i, j] \) 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 \( \text{TransferTime}[i, j] \) 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.
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, where 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 \( ST_{ti} \) and end \( ET_{ti} \) times of the task.

\[
\begin{align*}
\text{Algorithm 2} \\
\text{SCHEDULE GENERATION} \\
\text{Input:} \text{ Set of workflow tasks } T \\
\text{Initial resource pool } R_{initial} \\
\text{An array } pos[|T|] \text{ representing a particle's position} \\
\text{Output:} \text{ A Schedule } S \\
1. \text{ Initialize schedule components} \\
   1.1. R = \emptyset, M = \emptyset \\
   1.2. TEC = 0, TET = 0 \\
2. \text{ Calculate \( ExeTime[|T| \times |R_{initial}|] \)} \\
3. \text{ Calculate \( TransferTime[|T| \times |T|] \)} \\
4. \text{ for } i = 0 \text{ to } i = |T| - 1 \\
4.1. \text{ } t_i = T[i], r_{pos[i]} = R_{initial}[pos[i]] \\
4.2. \text{ if } t_i \text{ has no parents} \\
   \quad ST_{ti} = LET_{r_{pos[i]}} \\
   \text{ else} \\
   \quad ST_{ti} = \max \{ \max \{ ET_{tp} \mid t_p \in \text{parents}(t_i) \}, LET_{r_{pos[i]}} \} \\
   \text{ end if} \\
4.3. \text{ } exe_i = \text{exeTime}[[i][pos[i]]] \\
4.4. \text{ for each child } t_c \text{ of } t_i \\
   \text{ if } t_c \text{ is mapped to a resource different to } r_{pos[i]} \\
   \quad transfer_{i} = \text{TransferTime}[i][c] \\
   \text{ end if} \\
\text{ end for each} \\
4.5. \text{ } PT_{ti} = exe_i + transfer_{i} \\
4.6. \text{ } ET_{ti} = PT_{ti} - ST_{ti} \\
4.7. \text{ } m_{pos[i]}^{\text{best}} = (t_i, \text{parents}(t_i), ST_{ti}, ET_{ti}) \\
4.8. \text{ } M = M \cup \{ m_{pos[i]}^{\text{best}} \} \\
4.9. \text{ if } r_{pos[i]} \notin R \\
   \quad \text{LST}_{r_{pos[i]}} = \max (ST_{r_{pos[i]}}, \text{bootTime}) \\
   \quad R = R \cup \{ r_{pos[i]} \} \\
\text{ end if} \\
4.10. \text{ } LET_{r_{pos[i]}} = PT_{ti} + \text{LST}_{r_{pos[i]}} \\
5. \text{ Calculate } TEC \text{ according to equation (4)} \\
6. \text{ Calculate } TET \text{ according to equation (5)} \\
7. \text{ } S = (R, M, TEC, TET)
\end{align*}
\]
The start time value \( ST_{ti} \) 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 \( ti \).

The value of \( ET_{ti} \) is calculated based on the total processing time and the start time of the task. To determine the processing time \( PT_{r_{pos[i]} ti} \) 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 \( ti \) which is mapped to run in a resource different to \( r_{pos[i]} \). These two values are then added to obtain \( PT_{r_{pos[i]} ti} \) as defined in Equation (3). Finally it gains the value of \( ET_{ti} \) by subtracting \( ST_{ti} \) from \( PT_{r_{pos[i]} ti} \).

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.

4. CONCLUSION

This paper 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, solution proposed in this paper considers other characteristics typical of IaaS platforms such as performance variation and VM boot time. Furthermore, this heuristic is as successful in meeting deadlines as SCS, which is a dynamic algorithm. Also, in the best scenarios, when this heuristic, SCS and IC-PCP meet the deadlines, it is able to produce schedules with lower execution costs.

IJRISE| www.ijrise.org|editor@ijrise.org [495-502]
REFERENCES


