Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 1934784, 16 pages
https://doi.org/10.1155/2018/1934784

Research Article

Workflow Scheduling Using Hybrid GA-PSO Algorithm in Cloud Computing

Ahmad M. Manasrah[1] and Hanan Ba Ali[2]

[1]*Network and Information Security Department, Yarmouk University, Irbid 21163, Jordan*
[2]*Computer Sciences Department, Yarmouk University, Irbid 21163, Jordan*

Correspondence should be addressed to Ahmad M. Manasrah; ahmad.a@yu.edu.jo

Received 27 September 2017; Accepted 11 December 2017; Published 8 January 2018

Academic Editor: B. B. Gupta

Cloud computing environment provides several on-demand services and resource sharing for clients. Business processes are managed using the workflow technology over the cloud, which represents one of the challenges in using the resources in an efficient manner due to the dependencies between the tasks. In this paper, a Hybrid GA-PSO algorithm is proposed to allocate tasks to the resources efficiently. The Hybrid GA-PSO algorithm aims to reduce the makespan and the cost and balance the load of the dependent tasks over the heterogonous resources in cloud computing environments. The experiment results show that the GA-PSO algorithm decreases the total execution time of the workflow tasks, in comparison with GA, PSO, HSGA, WSGA, and MTCT algorithms. Furthermore, it reduces the execution cost. In addition, it improves the load balancing of the workflow application over the available resources. Finally, the obtained results also proved that the proposed algorithm converges to optimal solutions faster and with higher quality compared to other algorithms.

1. Introduction

The needs for computing and huge storage resources are fast growing. Therefore, cloud computing gets the attention due to the high performance computing services and facilities that are provided to the users as Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS) [1–3]. Various applications can be modeled as workflow applications of a set of tasks with dependencies between them in the sense that before one task can execute, dependant tasks have to complete their execution first. Workflow applications are being used in a range of domains, such as astrophysics, bioinformatics, and disaster modeling and prediction. Moreover, complicated problems like complex scientific applications are emerging recently through combining various methods and techniques in a single solution. For such a need, this type of applications has been executed on supercomputers, clusters, and grids [4]. Fortunately, with the advent of clouds, such workflow applications are executed in the cloud. The workflow applications are the mechanism of a large-scale business process execution, consisting of a set of events or tasks in which information is distributed from one task to another based on some technical rules, to achieve a general goal [5]. The workflow application tasks are dependent on each other, where the output of some tasks is the input to another. Therefore, the order of their execution must be considered when assigning the tasks to VM processors in a multiprocessor environment. Assigning the dependent tasks to the most appropriate VM processors is known to be an NP-complete problem as discussed by Verma and Kaushal [6]. The scheduling processes of the workflow applications are a multiobjective optimization problem (also known as Pareto optimization), where users might wish to minimize the money cost and the execution time for the whole workflow application with efficient load balancing over the VMs in the cloud environment. The optimal decision for the multiobjective workflow optimization is the trade-off between the three objectives; therefore, the objectives must be rated based on their importance to the users to select the best Pareto solutions because, for instance, minimizing the overall cost may lead to maximizing the execution time and the load over a specific VM [7, 8]. The workflow scheduling problem is an inherited problem from the heterogeneous computing environments, for which different research efforts were made to address the scheduling problem [9–11]. However, heterogeneous computing environments are not easy to set up, and their capability of giving more uniform performance with less failure is quite limited in comparison to the cloud environments [12, 13]. Moreover, the main objective of the various previous efforts in addressing the workflow scheduling problem in heterogeneous environments is to only minimize the finish time. Therefore, with the wide adoption of the cloud environments and their services as a pay-per-use model, there is a need to consider both the total monetary cost and the execution makespan. As a result, several metaheuristic algorithms were proposed to solve the scheduling problem of the workflow tasks and to get an efficient solution for tasks distribution over the different VMs in the cloud environment. For instance, Genetic Algorithm (GA) [14], Ant Colony Optimization [15], Swarm Intelligence [16], and Artificial Bee Colony (ABC) [17] are few examples of the various proposed solutions of workflow scheduling problem that addresses the total monetary cost and the execution makespan.

The main objective of this paper is to propose an algorithm that addresses the workflow scheduling problem. The proposed algorithm should also reduce the total makespan execution time and balances the load over the VMs with minimum total monetary cost. Therefore, this paper proposes a Hybrid GA-PSO algorithm through combining the strengths of both algorithms to address the workflow scheduling problem. The efficiency of the proposed algorithm is evaluated against other algorithms to prove its effectiveness in solving the workflow scheduling problem in the cloud environment.

The remainder of the paper is organized as follows. The problem description and the state-of-the-art in workflow scheduling are described along with the challenges when applying the existing common scheduling algorithms on IaaS platforms which are also highlighted in Section 2. This is followed in Section 3 by the design of the workflow scheduling algorithm and definitions of the proposed algorithm. Section 4 provides details of the performance evaluation of the multiobjective scheduling problem in cloud along with the experimental results and their discussion, and the paper is concluded and the future work is summarized in Section 5.

2. Related Work

Workflow scheduling problems are considered one of the main challenges in cloud environments. Many heuristic algorithms were proposed to solve the tasks scheduling problem using different strategies. However, the problem becomes obvious when the tasks are dependent on each other (i.e., workflow application). The dependent tasks require a specific execution order due to the relationship between them. There are two types of workflow scheduling: the best-effort workflow scheduling and the quality of services (QoS) constraint workflow scheduling [5, 18]. However, the best-effort workflow scheduling focuses on reducing the execution time of the whole workflow tasks regardless of other factors. Many types of research were based on the best-effort workflow scheduling to reduce the execution time, such as Braun et al. [16] who use the min-min algorithm for workflow scheduling. Their proposed approach executes the small tasks first and delays the larger tasks for a longer time. On the other hand, Mao et al. [19] use the max–min algorithm for task scheduling to execute the large tasks first and the small tasks are delayed for a longer time. In an attempt to resolve the aforementioned issues, Kumar and Verma [20] combined the min-min and max–min algorithms along with the Genetic Algorithm to improve the scheduling of multiple jobs over multiple virtual machines efficiently. The authors employ the min-min and the max–min algorithms to generate the GA individual and to provide better initial population rather than randomly chosen initial population. The achieved results were better than GA-based algorithms; however, it requires a lot of computation steps that consume time. This makes it unsuitable for cloud computing pay-per-use models. Guo et al. [21] proposed a Particle Swarm Optimization (PSO) based algorithm for solving the task-scheduling problem with an objective of reducing the total execution and transfer time. The optimization process is based on a heuristic scheduling combined with the PSO, to allocate the tasks to the different available resources. They practically proved that the PSO could run faster and give a better solution than GA. However, the PSO algorithm might get trapped in the a local optimal solution [22].

Different types of research, based on the QoS constraint for workflow scheduling, were considered to reduce the execution time under different predefined constraints, such as the following: user's predefined budget constraints, user predefined deadline constraints, or workflow scheduling considering the reliability, time, cost, load balance, and fault recovery constraints. In this regard, Pandey et al. [23] presented a heuristic algorithm based on Particle Swarm Optimization to solve the workflow tasks scheduling over cloud resources. The conducted experiment shows that the computation cost using the PSO algorithm is three times better than the "Best Resource Selection" (BRS) algorithm under user predefined time constraints. However, the obtained result was not completely accurate due to the fast convergence towards the solution, which may cause PSO to get stuck in the local optimal solutions, and even the results cannot reflect the real performance of PSO. Arabnejad and Barbosa [24] presented a Heterogeneous Budget-Constrained Scheduling (HBCS) algorithm. The algorithm computes two possible schedules for the DAG (Directed Acyclic Graph) of the workflow. One schedule produces the minimum execution time with the maximum cost, while the other produces the minimum cost. The user, therefore, is able to decide which schedule to use to execute his task before the required deadline and within the cost range. The HBCS algorithm reduces the makespan by 30% and the cost within the user's specified budget constraint. Furthermore, it reduces the time complexity compared to other budget-constrained algorithms.

Researchers such as Verma and Kaushal [6] realize that the priority of the tasks determined their execution order.

Consequently, they presented a *Bicriteria Priority Based Particle Swarm Optimization (BPSO)* algorithm, to schedule the workflow tasks over the available cloud resources. The BPSO algorithm represents the trade-off between the execution time and the execution cost under the user's predefined budget and deadline constraints. The proposed scheduling algorithm significantly reduces the execution cost and the makespan through selecting the best-known scheduling solution from the heuristic solutions under the predefined deadline and budget constraints compared to BHEFT (Budget-constrained Heterogeneous Earliest Finish Time) [31] and PSO algorithms [22, 26]. However, the BPSO algorithm does not consider the various loads of the available resources. Consequently, Xu et al. [25] developed a multiobjective heuristic algorithm based on the min-min algorithm. The proposed algorithm uses four real-world scientific workflows to evaluate its performance. The conducted experiments evaluate the performance of the makespan and the execution cost with fault recovery procedure. The heuristic algorithm, based on the min-min algorithm, is considered a better choice only when both the cost and the makespan are considered.

The multiobjective optimization is a very promising direction to tackle the problem of workflow scheduling. In this regard, Ge and Wei [27] used a Genetic Algorithm to optimize the tasks scheduling in the job queue. They used a centralized scheduler (i.e., master node) to distribute the waiting tasks to the different available resources (i.e., slave nodes) based on the resources status messages. Their results show that the proposed schedule was better than the First-In-First-Out (FIFO) and the Delay scheduling that distributes the load over all resources in the cloud. However, the proposed algorithm requires a lot of processing time to reach the optimal solution. Later, Fard et al. [28] suggested a heuristic static multiobjective scheduling algorithm for scientific workflows in heterogeneous environments. The proposed algorithm adopted the strategy of maximizing and minimizing the distance between the constraints for each of the four objectives (i.e., makespan, economic cost, energy consumption, and reliability). The researchers analyzed and categorized the different objectives based on their impact on the optimization process. The results showed that most of the generated solutions are within the predefined deadline and budget constraints. However, the proposed algorithm is not efficient with a small number of tasks and processors. Wu et al. [29], therefore, suggested a Revised Discrete Particle Swarm Optimization (RDPSO) algorithm to schedule the workflow applications over the different available resources. The experiments were conducted over a set of workflow applications with different data communication and computation costs. The result showed that the proposed RDPSO algorithm reduces the cost and yields better makespan compared to the standard PSO and BRS (Best Resource Selection) algorithm. However, the proposed algorithm is not efficient with large search space. Continuously, Chitra et al. [26] proposed a local minima jump solution using PSO (i.e., JPSO) for workflow scheduling in the cloud to schedule the tasks and load balance the workflow applications, to reduce the makespan. The JPSO algorithm overcomes getting trapped in the local minimal solution problem through making a jump in the g_{best} value to avoid the poor convergence of the g_{best} values. The results show that the proposed algorithm is more efficient compared to the GA algorithm by 3.8% with a small number of tasks. However, the GA algorithm shows the better result with a large number of tasks.

Many researchers attempted to solve the multiobjective optimization problem of the workflow applications using a different number of objectives. In this paper, a Hybrid GA-PSO algorithm is proposed to schedule the workflow tasks over the available resources. The proposed algorithm aims to achieve three objectives: reducing the makespan, reducing the cost, and balancing the load of the workflow tasks on heterogeneous VMs in the selected cloud DC. In summary, the GA-based algorithms provide better results than other algorithms when the number of iterations is large. However, increasing the number of iterations means that the GA algorithm will consume more time to reach the optimal solution. On the other hand, the PSO-based algorithms provide better results than the other algorithms and in less time. However, the results may not be accurate due to the fast convergence of the PSO-based algorithms to the solution, which may cause being stuck in the local optimal solution. Therefore, the proposed Hybrid GA-PSO algorithm is distinguished by the characteristics of the GA and the PSO algorithms. The Hybrid GA-PSO algorithm is expected to work faster with different sizes of workflow applications compared to other algorithms with the same objectives. Moreover, the Hybrid GA-PSO algorithm may not get trapped in the local optimal solution, because of the use of the GA mutation operator that enhances the accuracy of the solutions. Table 1 summarizes the review of the literature works along with their pros and cons.

3. The Proposed Algorithm

Many researchers used random workflows graph or real-world workflows graph to represent the workflow applications using the Pegasus framework [7]. Pegasus framework provides the DAG of different real workflow applications and defines the number of the workflow tasks, the sizes of transmission data between the tasks, and the execution time of each task. These workflows will be used for measuring the performance of the proposed Hybrid GA-PSO algorithms. There are five real workflow applications that are used in the scientific domains, namely, Montage [32], CyberShake [33], Epigenomics [34], LIGO Inspiral Analysis Workflow [35], and SIPHT [36], as portrayed in Figure 1. The Montage application created by NASA/IPAC closes together multiple input images to form custom mosaics of the sky [32]. The CyberShake workflow is used by the Southern California Earthquake Center to distinguish earthquake threatening a region [33]. The Epigenomics workflow created by the USC Epigenome Center and the Pegasus framework is used to automate the different operations in genome sequence processing [34]. LIGO's Inspiral Analysis workflow is used to create and analyze gravitational waveforms from data gathered during the coalescing of compact binary systems [35]. The SIPHT workflow, from the bioinformatics project at Harvard, is used to automate the search for small untranslated

Table 1: Literature review summary.

Author	Name of Algorithm	Objective	Advantages	Limitation
Braun et al. [16]	min-min algorithm	Time	12% better than GA	Delayed large tasks for long time
Kumar and Verma [20]	Combination of min-min and max–min strategies in Genetic Algorithm	Time	Faster than the GA	Time consuming
Guo et al. [21]	Particle Swarm Optimization (PSO) algorithm	Execution and transfer time	Faster than the M-PSO and L-PSO algorithms in a large scale	Stuck in local optimal solution
Pandey et al. [23]	Heuristic algorithm based on particle swarm optimization	Time and cost	Three times better cost compared to BRS, good load distribution over resources	Stuck in local optimal solution
Arabnejad and Barbosa [24]	Heterogeneous Budget-Constrained Scheduling (HBCS) algorithm	Execution time and cost	Reduction of 30% in execution time while maintaining the same budget	Not considering the load over resources
Verma and Kaushal [6]	Bicriteria Priority Based Particle Swarm Optimization (BPSO) algorithm	Time and execution cost	Decreasing the execution cost compared to BHEFT and PSO	Not considering the load over resources
Xu et al. [25]	Heuristic algorithm based on the min-min algorithm	The fault recovery, the time, and the cost	Fault recovery has a significant impact on the two performance criteria	Better choice only when both cost and makespan are considered
Chitra et al. [26]	The PSO algorithm	Load balance and the makespan	Better than GA and PSO	Time consuming
Ge and Wei [27]	The Genetic Algorithm	Load balance and makespan	Better than FIFO	Time consuming to reach to optimal solution
Fard et al. [28]	The heuristic algorithm	Makespan, economic cost, energy consumption, and reliability	Improve all four objectives	Not efficient with small number of tasks and processors
Wu et al. [29]	The Revised Discrete Particle Swarm Optimization (RDPSO) algorithm	Makespan, communication costs, and computation costs	Better than the standard PSO and BRS (Best Resource Selection) algorithm	Not efficient with large search space
The proposed algorithm	Genetic and particle swarm optimization algorithm	Makespan, communication costs, load balance, and execution and transfer time	Faster convergence to the solution in comparison with other approaches	Supports one data center without considering the dynamic workflow

RNAs (sRNAs) for bacterial replicons in the NCBI database [36].

The main steps of the GA-PSO algorithm are shown in Figure 2. The GA-PSO algorithm starts with generating a random population and defines a specific number of iterations as a parameter to the algorithm. The population represents several solutions to the workflow tasks problem and each solution is a distribution of the whole workflow tasks over the available VMs. The initialized population is passed through the GA algorithm with the first half of the defined iterations; that is, if the number of the iterations is (n), then the GA algorithm will be repeated ($n/2$) times. The reason behind using ($n/2$) iteration is to reduce the complexity of the proposed algorithm, as the performance of the GA algorithm depends mainly on the method used to encode solutions into chromosomes and particles and what the fitness function is measuring, as well as the size of the population, that is, the number of iterations, as proved by Alajmi and Wright [37]. These parameter values can be adjusted after evaluating the algorithm's performance on a few trial runs. Through experiments, the GA-PSO algorithm's performance was the best, when the defined number of iterations is divided equally between the GA and PSO algorithms. This also agrees with the concept of the divide and conquer that divides one problem into two subproblems to produce a complexity equal to $T(n) = 2xT(n/2) + F(n)$, where $F(n)$ is the divide and conquer time. The solution for such equation depends on $F(n)$ and the complexity is $O(F(n))$ if $|F(n)| \geq n$ (by master theory). Moreover, it is also known that both the GA and the PSO require many function evaluations because each needs to evaluate the objective of every population member in the current sample. Therefore, decreasing the population size in a GA or PSO (i.e., decreasing the number of iterations) is a common practice to avoid degrading the GA or PSO

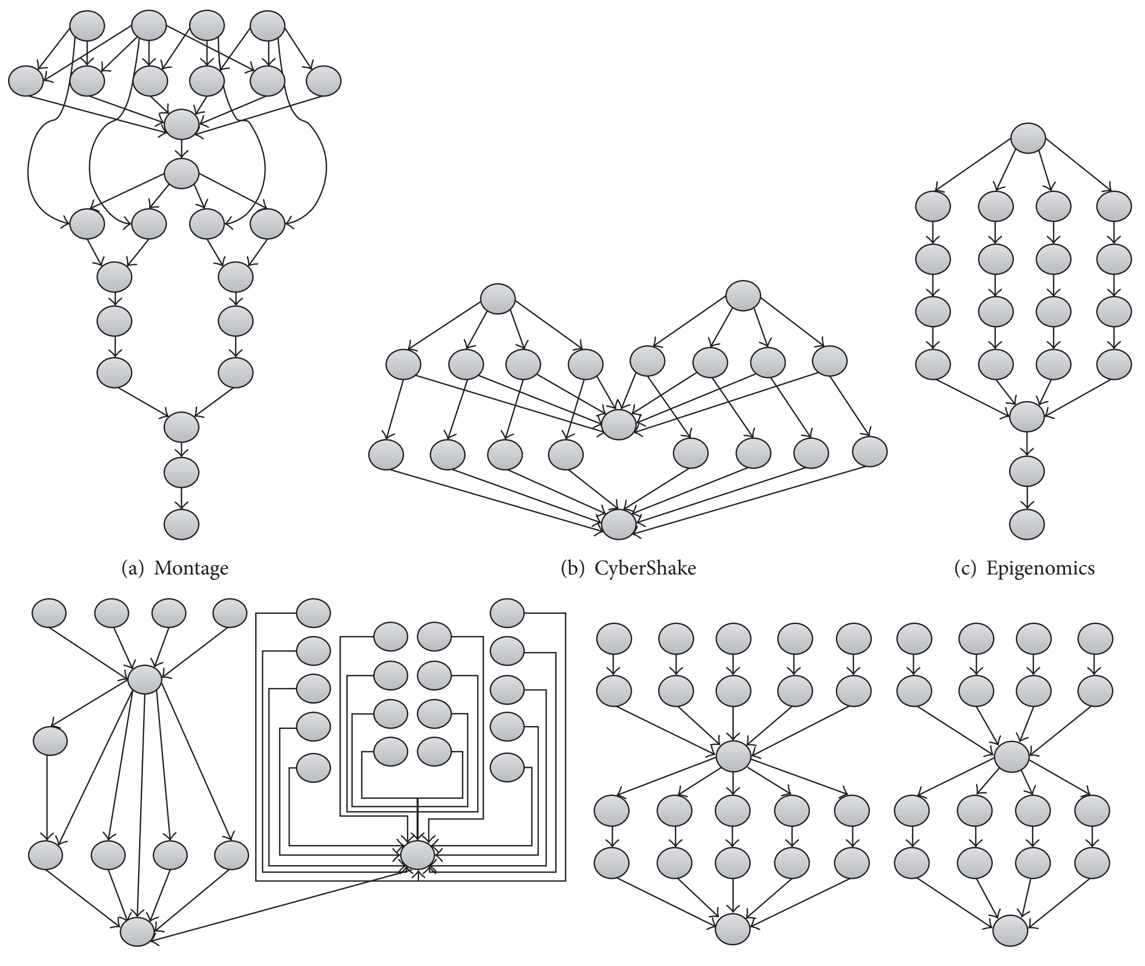
(a) Montage (b) CyberShake (c) Epigenomics (d) SIPHT (e) LIGO

Figure 1: Known scientific workflows.

performance in terms of the accurate results and reduction rate.

In the GA algorithm, the solutions are called chromosomes; the chromosomes are enhanced gradually at each iteration through the GA operators (i.e., selection, crossover, and mutation). The resulting chromosomes are passed to the PSO algorithm at the second half of the defined iterations. In the PSO algorithm, the chromosomes are called particles; the particles are enhanced gradually at each iteration through the PSO algorithm. The particle with the minimum fitness value is selected to represent the solution of the workflow task problem.

3.1. Initializing Population. The Hybrid GA-PSO algorithm is initialized to a specific number of iterations. A solution is initiated randomly at the first iteration. After the first iteration, a sequence of new populations are created and recursively enhanced using the previous solutions to form a set of suggested solutions as illustrated in Figure 3. The population in the GA algorithm is called chromosome. The length of the chromosomes is equal to the number of the workflow tasks, and the genes of each chromosome represent the different VMs. The randomly generated chromosomes represent the input to the proposed GA-PSO algorithm. The GA algorithm represents the first part of the proposed GA-PSO algorithm which will be used to generate different solutions to the workflow scheduling problem.

3.2. Applying the GA Algorithm. At the first phase, the GA is applied to the whole generated population for $(n/2)$ of the determined iterations, to generate the optimal solution from existing solutions, which is required to solve the scheduling problem. The PSO is applied to the whole generated population for the following $(n/2)$ of the determined iterations, which is generated by the GA algorithm. The PSO algorithm keeps in memory the best and the worst solutions, which can be useful for the fast solution convergence when GA generates bad solutions. The solutions in GA that define the scheduling solution of our problem are represented by several chromosomes with length equal to the number of the whole workflow tasks. Each chromosome consists of several genes representing the hosts' VMs. In each iteration, the GA passes the chromosomes between three different operators: the selection, crossover, and mutation operator. The first operator

```
Input: the chromosomes
Output: fitnesschromosome
Set the tournamentSize = n
  For i = 0 to tournamentSize
    id = Math.random( )∗ chromosome.size( )    // select chromosome randomly
    tournament[i] = get_chromosome (id)
  End For
fitness ← tournament.getFitnest( ) //return the fitness value in tournament group.
```

Algorithm 1: The tournament selection method.

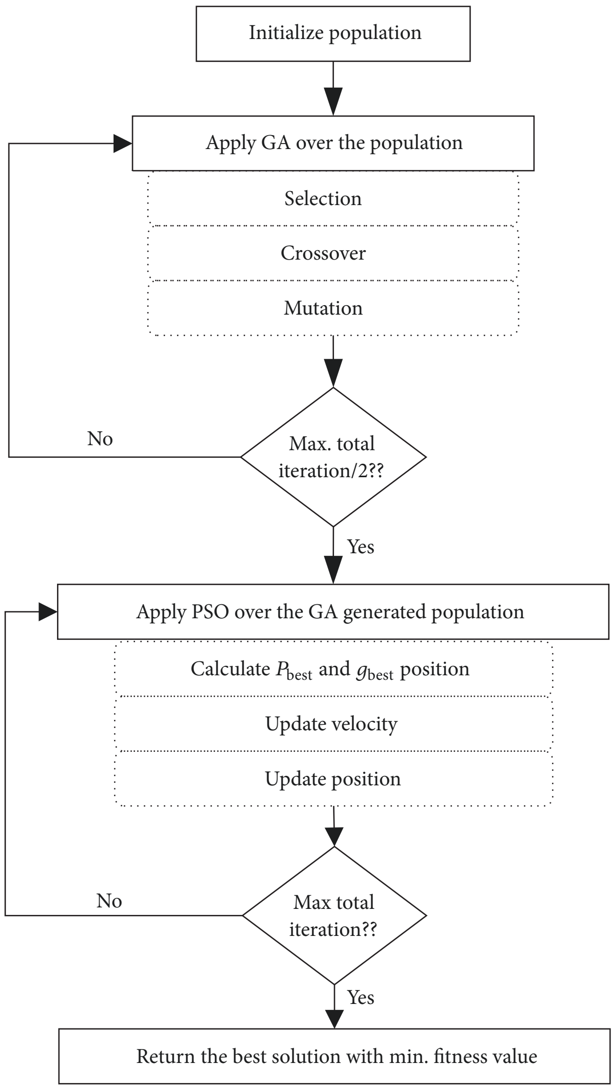
Initialize population
Apply GA over the population
Selection
Crossover
Mutation
No
Max. total iteration/2??
Yes
Apply PSO over the GA generated population
Calculate P_{best} and g_{best} position
Update velocity
Update position
No
Max total iteration??
Yes
Return the best solution with min. fitness value

Figure 2: The flowchart of the GA-PSO algorithm [30].

of the GA algorithm is the selection operator that is used to select different solutions from the existing chromosomes and used to build the next generation of the chromosomes for the next iteration.

3.2.1. Selection Operator. In the GA algorithm, not all generated chromosomes are evolved through the GA operators in each iteration. Therefore, the chromosomes are passed through the tournament selection to select the best chromosome from a group of chromosomes. The function selects a random (id) after running several tournaments between few chromosomes. The selected ids represent the index of the selected chromosome from a set of chromosomes. The best chromosome in the group is selected for crossover operator based on its fitness value as shown in Figure 4 and Algorithm 1.

3.2.2. The Crossover Operator. The crossover operator aims to generate new chromosomes through changing the position of the genes inside every two chromosomes. In the crossover, a random number is selected in the range of the number of the chromosome genes, to represent the division point of each chromosome into two parts. The crossover returns an offspring chromosome of two parts that contains both chromosomes genes, that is, VMs. The first group of VMs takes the first chromosome until the index, which is determined by the random number. The second chromosome has the second group of the VMs starting from the index, which is determined by the random number, until the end of the chromosome. The implementation of the crossover method is illustrated in Algorithm 2.

3.2.3. The Mutation Operator. The mutation operator aims to make unusual modifications in the new chromosomes that are generated from the previous crossover operator with better fitness value than the existing chromosomes. The mutation operator operates over the returned chromosome from the selection method, and the occurrence of the mutation is based on the mutation rate variable. The mutation process starts with a number that is randomly generated to be less than or equal to the mutation rate. Two genes, that is, VMs, are selected randomly from the same chromosome and checked to be different. If they are the same, their places are swapped to generate new chromosome, which represents a different distribution of the tasks over the available VMs. The generated chromosome is then passed to the next stage of the algorithm. The implementation of the mutation method is illustrated in Algorithm 3.

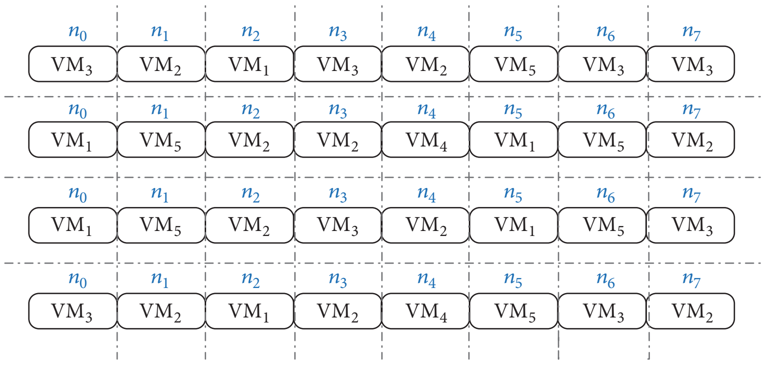
n_0 n_1 n_2 n_3 n_4 n_5 n_6 n_7
VM_3 VM_2 VM_1 VM_3 VM_2 VM_5 VM_3 VM_3
VM_1 VM_5 VM_2 VM_2 VM_4 VM_1 VM_5 VM_2
VM_1 VM_5 VM_2 VM_3 VM_2 VM_1 VM_5 VM_3
VM_3 VM_2 VM_1 VM_2 VM_4 VM_5 VM_3 VM_2

FIGURE 3: An example of a randomly initiated population.

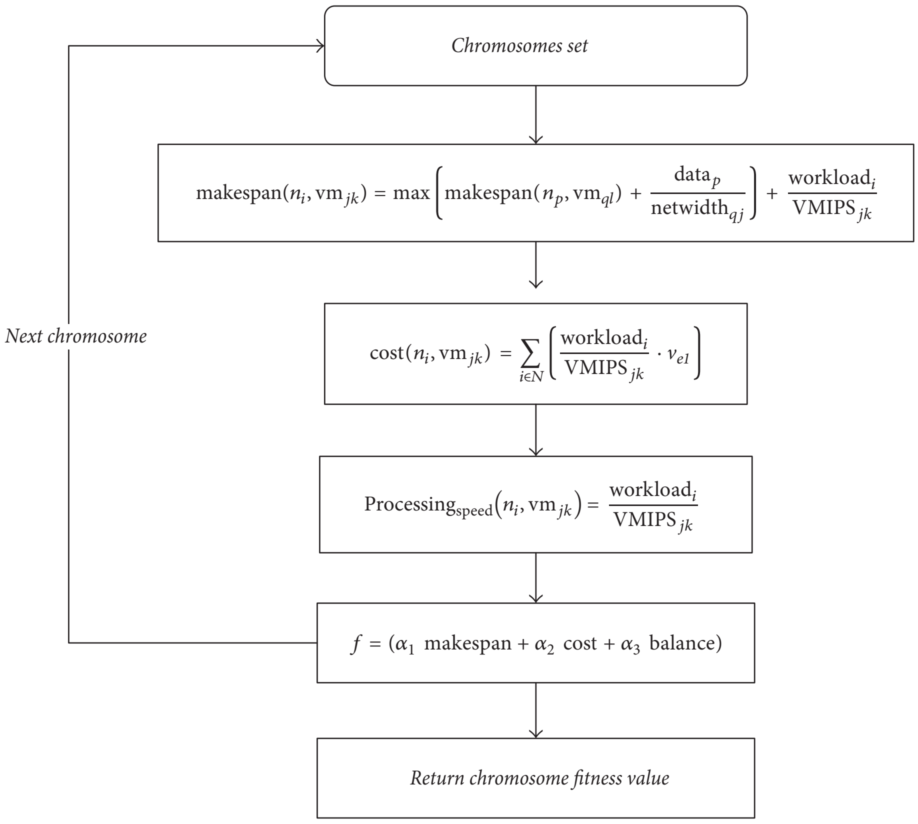
Chromosomes set
$\text{makespan}(n_i, \text{vm}_{jk}) = \max\left(\text{makespan}(n_p, \text{vm}_{ql}) + \frac{\text{data}_p}{\text{netwidth}_{qj}}\right) + \frac{\text{workload}_i}{\text{VMIPS}_{jk}}$
$\text{cost}(n_i, \text{vm}_{jk}) = \sum_{i\in N}\left(\frac{\text{workload}_i}{\text{VMIPS}_{jk}} \cdot v_{el}\right)$
$\text{Processing}_{\text{speed}}(n_i, \text{vm}_{jk}) = \frac{\text{workload}_i}{\text{VMIPS}_{jk}}$
$f = (\alpha_1\ \text{makespan} + \alpha_2\ \text{cost} + \alpha_3\ \text{balance})$
Return chromosome fitness value
Next chromosome

FIGURE 4: The schema of the fittest value.

```
**Input**: two chromosomes
**Output**: offspring chromosome
$r$ = (Math.random( ) * chromosome.length)
    **For** $i = 0$, $j = 0$ to $r$
        offspring _chromosome[$j$] = chromosome1[$i$]
    **End For**
    **For** $i = r$ to chromosome.length
        offspring_chromosome[$j$] = chromosome2[$r$]
    **End For**
```

ALGORITHM 2: The crossover method.

3.3. Applying the PSO Algorithm. The solutions that are returned from the GA algorithm are fed into the PSO algorithm with the rest of the determined iterations, to find the optimal solution from the GA generated solutions. In the PSO algorithm, the solutions are called particles, the individuals of each particle represent the VMs of the DC, and the index of each VM represents a workflow task. The PSO algorithm consists of three stages as follows.

3.3.1. Evolve ($\mathbf{g}_{\text{best}}$) *and* ($\mathbf{p}_{\text{best}}$) *of the Particles.* In each iteration, a new generation of the particles is produced based

```
Input: offspring_chromosome          //returned from crossover operator
Output: Newchromosome
Set mutationRate = 0.5
    If (Math.random( ) ≤ mutationRate)
        $t_1$= Rand [0, 1]∗ offspring_chromosome.length    // select a random number $t_1$
        $t_2$= Rand [0, 1]∗ offspring_chromosome.length    // select a random number $t_2$
            If offspring_chromosome [$t_1$] !=offspring_chromosome[$t_2$]
                Swap (offspring_chromosome [$t_1$], offspring_chromosome[$t_2$])
            End If
    End If
```

Algorithm 3: The mutation method.

```
Input: particles
Output: ($\mathbf{g}_{\text{best}}$) and ($\mathbf{p}_{\text{best}}$) vlues
Set $\mathbf{p}_{\text{best}}$ = null; $\mathbf{g}_{\text{best}}$= null; $k$=0;          //$k$ is the index of the particles.
 While not Reach max particles.size do
        If $\mathbf{p}_{\text{best}}[k]$ == null ‖  $\mathbf{p}_{\text{best}}[k]$.getFitness() > particles[$k$].getFitness()
            $\mathbf{p}_{\text{best}}[k]$ =particles[$k$];
        End If
        If $\mathbf{g}_{\text{best}}$ == null ‖  $\mathbf{p}_{\text{best}}$ [$k$].getFitness() < $\mathbf{g}_{\text{best}}$.getFitness()
          $\mathbf{g}_{\text{best}}$ = particle($\mathbf{p}_{\text{best}}[k]$);
          $k = k + 1$;
        End If
 Repeat    // until the last particle
```

Algorithm 4: Evolve ($\mathbf{g}_{\text{best}}$)and ($\mathbf{p}_{\text{best}}$) values.

on their velocity and position in the previous iteration. The changes in the velocity and position of the particles are based on the values of (g_{best}) and (p_{best}) which are evolved in each iteration. The implementation of evolving (g_{best}) and (p_{best}) values of the particles is illustrated in Algorithm 4.

The progress of the particles in the PSO algorithm is based on the values of ($\mathbf{g}_{\text{best}}$) and ($\mathbf{p}_{\text{best}}$) that keep changing with each iteration. At the first iteration, $\mathbf{p}_{\text{best}}[k]$ is equal to the solutions that are generated from the GA algorithm, where k is to distinguish each solution from the other. The ($\mathbf{g}_{\text{best}}$) is equal to the solution with the smallest fitness value. Furthermore, at each iteration, the comparison between the previously generated particles and the newly generated particles is taking place based on the fitness value. The particle with the best fitness value is stored in ($\mathbf{p}_{\text{best}}$). The ($\mathbf{g}_{\text{best}}$) store the best particle from the whole generation of the particles at each iteration by comparing their fitness value and the one with the best value in ($\mathbf{P}_{\text{best}}$). At each iteration, the comparison process ensures that all particles are moving towards the best solution, to reach the optimal solution.

3.3.2. Update the Velocity and Position Matrix. After generating the initial particles velocity and position values randomly and calculating both ($\mathbf{p}_{\text{best}}$) and ($\mathbf{g}_{\text{best}}$), the velocity of each particle, in each iteration, is updated accordingly. The implementation of the update process of the velocity matrix is illustrated in Algorithm 5.

The process of updating the velocity of the particles aims to generate a new generation from the different VMs locations that have better fitness value than the previous one. Each individual in the particles is compared with its ($\mathbf{p}_{\text{best}}$) value, which was generated at the previous iteration. The velocity value for each individual is decreased when both individuals in ($\mathbf{p}_{\text{best}}$) and the particle are equal; otherwise, the velocity value is increased. Similarly, a comparison of each individual in the particles and their ($\mathbf{g}_{\text{best}}$) values, from the previous iteration, is taking place. The velocity value for each individual is decreased when both individuals in ($\mathbf{g}_{\text{best}}$) and the particle are equal; otherwise, the velocity value is increased. Accordingly, the position of the VMs of each particle is changed based on the updated values of the velocity. The implementation of the update position matrix is illustrated in Algorithm 6.

Two VMs that have the maximum velocity values are swapped within each particle of all particles within the produced population. The termination criteria of the GA-PSO algorithm are represented by reaching the maximum number of iterations. When the termination criteria are satisfied, the solution that has the smallest fitness value within the population, which was generated at the last iteration, is presented as the scheduling solution of the workflow application. Otherwise, the ($\mathbf{g}_{\text{best}}$) and ($\mathbf{p}_{\text{best}}$) values evolve repeatedly until the termination condition is achieved. The complete GA-PSO algorithm is illustrated in Algorithm 7.

```
Input: velocity values
Output: updated velocity values
Set $k = 0; c_1 = 1; c_2 = 1.1; r_1, r_2 = \text{rand}[0, 1];$    //$k$ is the index of the particles individuals
    While not reach max particles.length do
        If Particle[$k$] == $\mathbf{p}_{\text{best}}[k]$
            velocity[$k$] $-= c_1 * r_1$;          //$r_1, r_2$ are random numbers
            Else
            velocity[$k$] $+= c_1 * r_1$;          //$c_1, c_2$ are acceleration coefficient
        End If
        If Particle[$k$] == $\mathbf{g}_{\text{best}}[k]$
            velocity[$k$] $-= c_2 * r_2$;
            Else
            velocity[$k$] $+= c_2 * r_2$;
        End If
    $k = k + 1$;
  Repeat                                        //until the last particle
```

Algorithm 5: Update the velocity matrix.

```
Input: updated velocity values
Output: updated particles position
Set $j = 0$;                                   // $j$ is the index of the particle individuals
While not reach max particles.size ( ) do
        Maxvelocity$_1$ ← get_max$_1$(Particle$_j$, velocity values)
        Maxvelocity$_2$ ← get_max$_2$(Particle$_j$, velocity values)
        Swap(Particle$_j$[Maxvelocity$_1$], Particle$_j$[Maxvelocity$_2$])
            $j = j + 1$;
Repeat          //until the last particle
```

Algorithm 6: Update the position matrix.

```
Input: workflow $W\{N, E\}$ and set of resources $\{VM_1, VM_2, VM_3, \ldots, VM_j\}$
Output: $\mathbf{g}_{\text{best}}$                       // the best solution to allocate $W$ over $VM_j$
  For $i = 0$ to $p$
    population ← randomize() // initialize population, $P$ is the population size
  End For
  While not Reach $n/2$ do                 // $n$ is number of iterations
        While not Reach max $P$ do
        chromosom$_j$ ← tournament(population)    //selection operator
        chromosome$_i$ ← tournament(population)
        offspring_chromosome$_j$ ← crossover(chromosome$_j$, chromosome$_i$)
        Newchromosome$_j$ ← mutation(offspring_chromosome$_j$)
        Repeat
  Repeat
Set Newchromosome$_j$ as particle$_j$          // $j$ is the index of the particles
Initialize particles position and velocity randomly
Calculate the ($\mathbf{g}_{\mathbf{best}}$) and ($\mathbf{p}_{\mathbf{best}}$) values
  While not Reach $n$ do
  velocity matrix ← update(particle$_j$ velocity)
  position matrix ← update(particle$_j$ position)
  Repeat
```

Algorithm 7: The proposed algorithm.

The algorithm is bounded by the GA operations (i.e., mutation, crossover, and selection). However, calculating the complexity of the GA or PSO algorithms is unlikely to be useful and worse probably deceptive. Moreover, because of the complexity (i.e., NP-complete) of the workflow scheduling problem, it is very challenging to develop an optimized workflow scheduling algorithm for workflow tasks distribution to the available resources within a reasonable overhead, that is, CPU time. However, since the main goal of the proposed scheduling algorithm is to optimize the overall cost (i.e., may not be optimal), it is, therefore, a practical trade-off between the overhead of the task-scheduling algorithm and the optimization on the running cost of the data center. Therefore, as will be demonstrated in our simulation experiments, we will evaluate the time complexity by measuring and averaging the runtime with a different number of tasks, as discussed in Section 4.2.

4. Performance Evaluation

For the purpose of evaluating the proposed algorithm, the proposed GA-PSO algorithm was implemented using the WorkflowSim [38]. The WorkflowSim extends the existing CloudSim simulator [39] by providing a higher layer of workflow management, through providing a suitable environment for applying different scheduling algorithms. Furthermore, to evaluate the performance of the proposed GA-PSO algorithm, the obtained results of the proposed GA-PSO algorithm have been compared with existing work scheduling algorithms, such as GA proposed in [27] and PSO proposed in [21]. In addition, the performance of the proposed GA-PSO algorithm was also compared with other related works, as discussed in Section 4.3.

4.1. Environment Setup. To evaluate the impact of the proposed algorithm on the workflow scheduling problem in comparison with other algorithms, we ran extensive experiments on real workflow applications using the simulation parameters in Table 2.

Table 2: Simulation parameters.

Parameter	Value
Number of tasks in application	25–1000
The number of VMs	16
MIPS	250–1500
RAM	256–1024 (MB)
BW	250–1500 (mbps)
Processor speed	10,000
Number of processors	4
VM policy	TIME_SHARED

These parameters were used to identify the characteristics of the VMs and the workflow applications in the experiments. A real workflow application—Montage workflows application—was created with different numbers of tasks to evaluate three objectives: (1) reducing the makespan of the application, (2) optimizing the processing cost, and (3) balancing the load on the different resources with respect to the different heterogeneous resources characteristics. The parameters defined in Table 3 were used throughout the GA-PSO evaluation experiments.

Table 3: GA-PSO algorithm parameters.

Parameter	Value
Population size	100
Mutation rate	0.05
Crossover	Single point
Number of iterations	100
Number of executions	500
$C1$	1
$C2$	1.1
$r1$, $r2$	[0, 1]
$\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$	0.4
$\boldsymbol{\alpha}_3$	0.2

The algorithm starts with 100 random solutions, called population. The single point crossover method was chosen in the GA phase (Section 3.2). The mutation operator rate was defined as 0.05 in the mutation stage. In the PSO algorithm phase (Section 3.3), the acceleration coefficients (C_1) and (C_2), as well as the random number which is used in the update velocity and position equation, were also defined, as in Table 3. Furthermore, the degree of importance of each objective in the fitness function was defined as "$\alpha_1, \alpha_2 = 0.4$ and $\alpha_3 = 0.2$" for the makespan, execution cost, and the load balance rate, respectively. The Montage workflow application is used in the evaluation with a different number of tasks (25–1000), to enlarge the size of the workflow and evaluate the algorithm under these different cases. The number of iterations for the GA-PSO algorithm was defined to 100 iterations to reach the optimal solution. The experiments were repeated 500 times, and the average results were compared with other algorithms. Four experiments were conducted based on the characteristic of the VMs, as in Table 2. The size of the workflow will be changed to examine the ability of the proposed algorithm in reducing the makespan, execution cost, and load balance for the small and large size workflow applications. For this purpose, four test scenarios using Montage workflow application with a different number of tasks and different number of edges and data sizes were used. The characteristics of the Montage workflow application that were used in the experiments are summarized in Table 4.

4.2. Performance Analysis. All the four scenarios were executed to evaluate the reduction in the makespan, the execution cost, and the load balance using the proposed GA-PSO algorithm in comparison with the GA and PSO algorithms. The results of the executed experiments for the four scenarios are reported in Table 5.

For each scenario, the number of tasks in the Montage workflow was increased. Scenario One, for instance, represents a small search space that makes the process of reaching the optimal solution fast and straightforward

Table 4: The characteristics of the Montage workflows.

Scenarios	Number of tasks	Number of edges	Average data size (MB)
Scenario One	25	95	3.43
Scenario Two	50	206	3.36
Scenario Three	100	433	3.23
Scenario Four	1000	4485	3.21

Table 5: The result of the executed experiments.

Algorithm	Makespan (sec)	Execution cost ($)	Load balance (rate)
Scenario One			
GA-PSO	95.09	16.85	9.76
GA	197.65	52.68	52.58
PSO	101.21	18.16	21.33
Scenario Two			
GA-PSO	116.01	49.89	13.81
GA	250.89	86.34	61.93
PSO	155.31	62.86	18.23
Scenario Three			
GA-PSO	233.78	127.74	33.03
GA	345.72	137.09	49.2
PSO	253.44	133.55	41.82
Scenario Four			
GA-PSO	1585.6	1021.42	73.83
GA	2402.28	1529.23	134.67
PSO	1802.31	1200.41	90.15

(i.e., the simplest case). On the other hand, Scenario Four represents a large number of tasks in the Montage workflow to expand the search space (i.e., the worst case). The large search space makes the process of finding the optimal solution a challenging task for the optimization algorithm. The results in Table 5 show minor differences in the makespan, the execution cost, and the load balance between the GA and PSO algorithms with Scenario One. However, there is a slight improvement for the GA-PSO algorithm compared with GA and PSO algorithms. On the other hand, the results show significant differences for the GA-PSO algorithm compared with the GA algorithm with Scenario Two and Three. These significant differences could be due to the unnecessary diversity caused by an inappropriate mutation rate. There is also a slight difference in the result between the GA-PSO and the PSO algorithm with Scenarios Two and Three as well. This slight difference is due to the fact that the GA-PSO algorithm depends mainly on the PSO algorithm in converging the solutions towards the optimal solution. In Scenario Four, the large number of tasks expands the search space to represent the worst case scenario. The GA-PSO algorithm still achieves a better result compared to the GA algorithm. This result is due to the fast solution convergence that avoids the unnecessary diversity of the solutions. In addition, GA-PSO

Table 6: Average results in makespan, execution cost, and load balance for the different algorithms.

Methods	Avg. makespan	Avg. execution cost	Avg. load balance
Hybrid GA-PSO	507.62	303.975	32.6075
GA	799.135	451.335	74.595
PSO	578.0675	353.745	42.8825

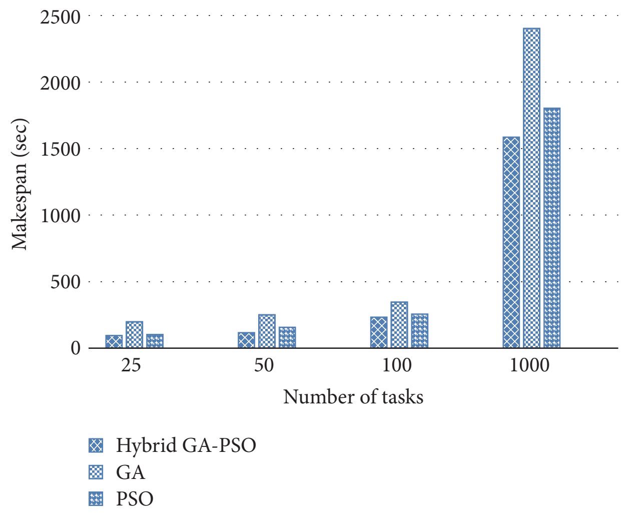

Figure 5: Comparison of the average makespan over different number tasks.

algorithm showed a significant enhancement compared to the PSO algorithm, because the PSO algorithm normally gets trapped in the local optimal solution. The above experiment was repeated several times, and the average results in terms of makespan, execution cost, and the load balance for the proposed GA-PSO, GA, and PSO algorithms with the four scenarios were calculated and consolidated in Table 6. The results in Table 6 also demonstrate the proposed algorithm ability in resolving the workflow task-scheduling problem in comparison with the GA and the PSO algorithms.

When comparing the improvement in the makespan using the proposed GA-PSO algorithm with the GA and PSO algorithm, one can notice that the GA-PSO algorithm achieves a significant enhancement of 16% better than the GA algorithm and 4% better than the PSO algorithm as illustrated in Figure 5. This is because the proposed GA-PSO algorithm always chooses the most appropriate VMs to execute the tasks without focusing only on fast VMs, which actually may overload one VM over the other and slow down the overall execution of the workflow application (i.e., increase the execution time).

In terms of execution cost, Figure 6 shows that the proposed GA-PSO algorithm is 13% better than the GA algorithm and 4% better than the PSO algorithm.

The improved result of the proposed GA-PSO algorithm is because the proposed algorithm chooses VMs to achieve a minimum execution cost to execute the selected tasks. Finally,

Table 7: The running time of the executed algorithms in seconds.

Method/number of Tasks	25 tasks	50 tasks	100 tasks	1000 tasks	2000 tasks	3000 tasks
GA	0.869465	0.888796	1.093582	21.321738	23.4338	28.28996
PSO	0.761534	0.871797	1.037802	18.515041	18.65318	23.99583
Hybrid GA-PSO	0.764333	0.873796	1.025266	17.576722	18.00189	22.44825

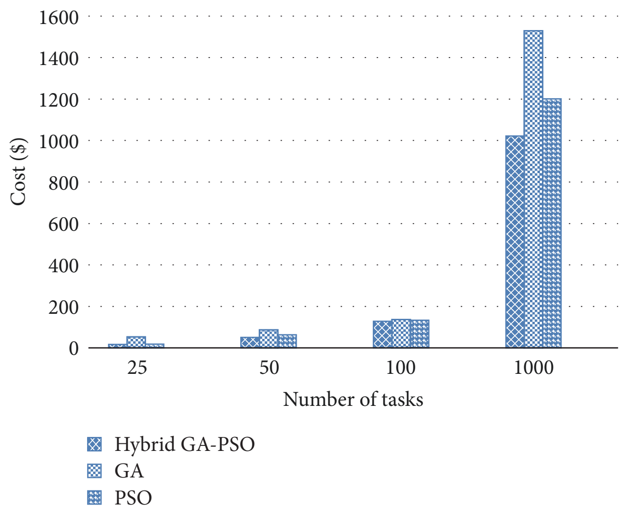

Figure 6: Comparison of the average execution cost over different number tasks.

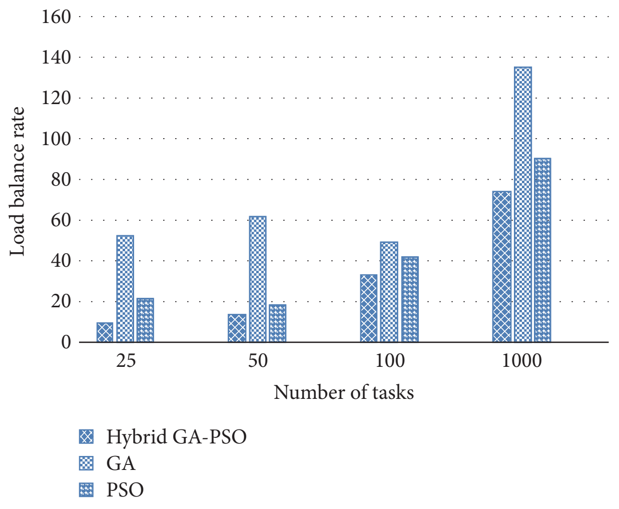

Figure 7: Comparison of the average load balancing rate over different number of tasks.

the proposed GA-PSO algorithm balances the load over the resources compared with GA and PSO algorithms as shown in Figure 7.

The average result of the load balancing obtained by the proposed GA-PSO algorithm is better than the GA algorithm by 28%, and the load balance was reduced by 4% compared to the PSO algorithm. This result is because the proposed GA-PSO algorithm converges to the solutions in a better way using the GA algorithm with avoiding the unnecessary diversity that may degrade the quality of the algorithm.

Finally, the CPU time is defined as the average running time of the proposed GA-PSO algorithm in comparison with the GA and the PSO algorithms running on hardware of the characteristics defined in Table 2. The result of the average running time for each algorithm using a different number of tasks is consolidated in Table 7. It can be noticed that the GA algorithm consumes more CPU time compared to the other algorithms. When the workflow size increases, the CPU time of the GA and PSO is also increased. For instance, with 3000 tasks, the GA took about 28.3 seconds and the PSO consumed 23.9 seconds while the proposed algorithm only took 22.4 seconds to reach the final solution.

The increase in the CPU time is actually because of the ($\mathbf{g}_{\text{best}}$) and ($\mathbf{p}_{\text{best}}$) update process. The update process calculates the fitness value for every particle with calculating the makespan and the cost at the same time. This is true because the proposed GA-PSO algorithm builds the solutions task by task, and hence the CPU time increases as the workflow size increases.

4.3. Comparison of Related Approaches. For the comparison purposes, three algorithms were evaluated for workflow tasks scheduling, namely, HSGA algorithm proposed in [40], WSGA algorithm proposed in [41], and MTCT algorithm proposed in [25] with the proposed GA-PSO algorithm. The comparison was carried out over two objectives: the makespan and the execution cost. The reason behind the selected objectives is that WSGA and MTCT algorithms optimize only the makespan and the execution cost, while HSGA optimizes only the load balancing and the makespan. The algorithms were implemented according to their description in the literature. The results show that the proposed GA-PSO algorithm converges to the optimal solution faster than the other algorithms and with higher quality in terms of load balancing as discussed in Section 4.2. All performance analyses were carried out over a workflow with different numbers of tasks, 25, 50, and 100 along with specific parameters, as defined in Table 8. The size of tasks, price, and the speed of resources are generated randomly to simulate a heterogeneous environment.

The workflow application was evaluated with a different number of tasks, to illustrate the impact of the proposed GA-PSO on the makespan and the load balancing rate in comparison with the HSGA algorithm. Table 9 and Figure 8 illustrate the average results of the experiment.

The result illustrated in Figure 8 shows that the proposed GA-PSO algorithm is able to solve the workflow problem with better makespan and load balancing than the HSGA by 11%

Table 8: GA-PSO versus HSGA simulation parameters.

Parameter	Value
Number of tasks in application	20–100
Task lengths	12–72 ($\times10^5$ MI)
Number of resources	30
Resource speeds	500–1000 (MIPS)
Bandwidth between resources	10–100 (mbps)

Table 9: GA-PSO versus HSGA experiment results.

Methods	Avg. makespan	Avg. load balance
GA-PSO	28191.96	2.23
HSGA	35000	2.63

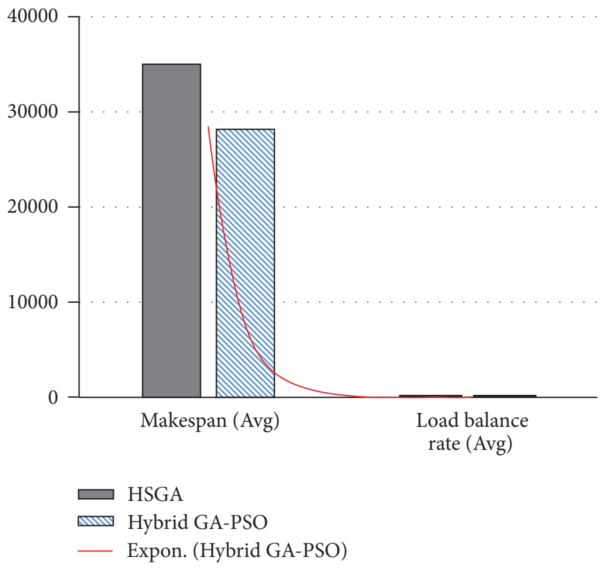

Figure 8: GA-PSO versus HSGA analysis of results.

Table 10: GA-PSO versus WSGA simulation parameters.

Parameter	Value
Population size	20
Selection method	Roulette wheel
Crossover method	Single point crossover
Mutation rate	0.1
The number of resources	3–14
Number of tasks in application	50–100
Number of iterations	200

Table 11: GA-PSO versus WSGA experiment results.

Methods	Avg. makespan	Avg. execution cost
Hybrid GA-PSO	84.875	5.195
WSGA	93	7.695

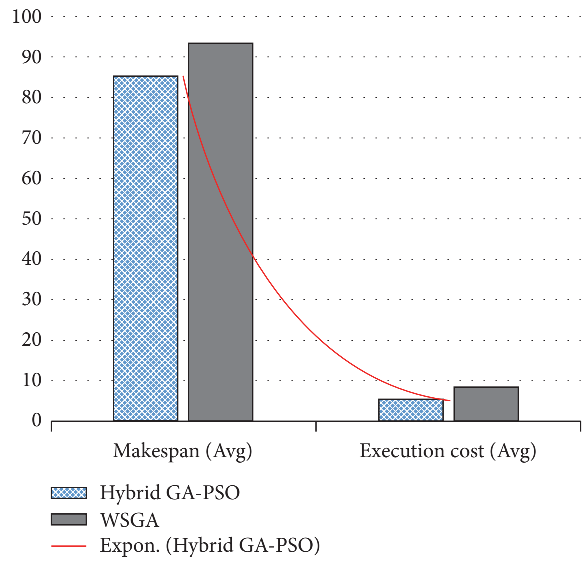

Figure 9: GA-PSO versus HSGA analysis of results.

and 9%, respectively. The improvement of the proposed GA-PSO algorithm is due to the fast convergence to the solution, as an advantage of employing the PSO algorithm, which avoids the unnecessary diversity that may occur in the HSGA algorithm and leads to reaching the best solution. Similarly, we compared the proposed GA-PSO algorithm with WSGA algorithm based on the simulation parameters in Table 10.

The different values of the workflow size and the resources configurations illustrate the impact of the proposed GA-PSO and the WSGA algorithms on the makespan and the execution cost. The average results of the experiment are shown in Table 11 and Figure 9.

The proposed GA-PSO algorithm obtained a solution for the workflow problem by 5% better value for the makespan and 9% better value for the execution cost in comparison with WSGA. It is worth mentioning that both the proposed GA-PSO and the WSGA algorithms are based on GA technique. However, the proposed GA-PSO algorithm uses the PSO algorithm to avoid the unnecessary diversity in the solution and enhances the obtained solutions, which might be scattered due to GA technique. Finally, the proposed GA-PSO algorithm was also compared with the MTCT algorithm, based on the simulation parameters in Table 12.

For the evaluation purposes, four different types of workflow applications were used to show the impact on the makespan and the execution cost of the proposed GA-PSO and the MTCT algorithm. The details of the workflow applications are illustrated in Table 13.

The makespan and the execution cost results, of the proposed GA-PSO and the MTCT algorithms, with the four types of workflow applications, are summarized in Table 14 and Figure 10.

The obtained results of the GA-PSO and the MTCT algorithms show that the GA-PSO algorithms enhance the makespan by 11% with 15% less in execution cost, in comparison with the MTCT algorithm, using the Montage workflow, whereas the proposed GA-PSO algorithm achieved an

Table 12: GA-PSO versus MTCT simulation parameters.

Parameter	Value
The number of resources	20
Resource speeds	500–1000 (MIPS)
Bandwidth between resources	20 (mbps)

Table 13: Workflows details.

Workflow	The number of tasks in different workflow sizes			
	Small	Medium	Large	XLarge
Montage	25	50	100	1000
CyberShake	30	50	100	1000
Epigenomics	24	46	100	1000
LIGO	30	50	100	977

Table 14: GA-PSO versus MTCT experiment results.

Methods	The makespan (sec)	The execution cost ($)
Montage		
Hybrid GA-PSO	1.12	1.04
MTCT	1.4	1.4075
CyberShake		
Hybrid GA-PSO	0.9875	1.12
MTCT	1.365	1.3725
Epigenomics		
Hybrid GA-PSO	1.23	1.112
MTCT	1.3525	1.36
LIGO		
Hybrid GA-PSO	1.1075	1.132
MTCT	1.4975	1.4225

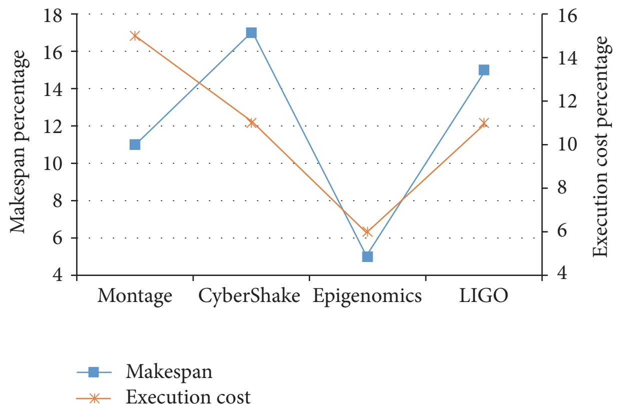

Figure 10: GA-PSO versus MTCT analysis of results.

improvement by 17% in terms of makespan and 11% less in the execution cost, compared with the MTCT algorithm, using the CyberShake workflow. Furthermore, the proposed GA-PSO algorithm schedules the Epigenomics workflow with 5% better makespan and 9% less execution cost than the MTCT algorithm. Finally, the results of the makespan and the execution cost of the LIGO workflow were better by 15% and 11% compared with the MTCT algorithm, respectively.

The results and the enhancements that were obtained by the proposed GA-PSO algorithm are because the proposed algorithm always selects the best solution for distributing the workflow tasks over the most suitable VMs regardless of the number of the workflow tasks. The proposed GA-PSO algorithm combines the suitable diversity and the fast convergence to optimal solutions, to find the optimal solution faster than any other algorithm.

5. Conclusion and Future Work

In this paper, a GA-PSO algorithm was proposed and implemented using the WorkflowSim simulator, for workflow task scheduling in cloud environments. The performance of the proposed algorithm was also compared with some known algorithms such as GA, PSO, HSGA, WSGA, and MTCT. The purpose of the proposed algorithm is to ensure a fair distribution of the workload among the available VMs, considering the order of the execution of the workflow tasks to reduce the makespan and the processing cost of the workflow applications in cloud computing environments. The GA-PSO algorithm selects the VMs to execute the workflow tasks in the minimum time based on the execution speed of the VMs and the size of the workflow tasks. The design of the GA-PSO algorithm tends to allow executing the tasks over the VMs with a balanced load distribution over the fast and slow VMs, without overloading some VMs over the others. This technique reduces the makespan through a fair utilization of the slow VMs instead of overloading the fast VMs and slowing down the overall execution of the tasks. The GA-PSO algorithm yields an optimal solution of the workflow task scheduling in terms of makespan compared with GA, PSO, HSGA, and WSGA algorithms by 16%, 4%, 11%, and 5%, respectively. In addition, the enhancements in the makespan using the Montage, CyberShake, Epigenomics, and LIGO workflow were averaged as 11%, 17%, 5%, and 15%, respectively, in comparison to MTCT algorithm. Moreover, the results prove that the GA-PSO algorithm minimizes the total execution cost of the workflow tasks compared to GA, PSO, and WSGA algorithms by 13%, 4%, and 9%, respectively. The GA-PSO algorithm also enhances the execution cost in comparison to MTCT algorithm using the Montage, CyberShake, Epigenomics, and LIGO workflow which are averaged at 15%, 11%, 9%, and 11%, respectively. The significance of the results, from the GA-PSO algorithm, are affected by the appropriate selection of the VM with a balance between cost and time through the fitness function of the GA-PSO algorithm. This goal was achieved by using the same weights for both the makespan and the execution cost in the fitness function. The proposed GA-PSO algorithm improves the load balancing of the workflow applications over the available resources, in contrast with GA, PSO, and HSGA algorithms, by allocating the tasks based on the VMs ability and the task sizes. The enhancements of the load balance in comparison with GA, PSO, and HSGA algorithms are averaged at 28%, 4%, and 9%, respectively. The design of the GA-PSO algorithm uses the standard deviation to select the best solution that keeps the variance of the distributed load, over the VMs, as low as possible taking into account the size of the tasks and the speed of each VM during the distribution of the tasks.

In the future, the work can be extended to more than one data center in a heterogeneous environment. Furthermore, the distribution of the workflow application can be extended into two levels: when workflow tasks reach the service broker and when the workflow tasks are distributed to the available VMs of each DC based on the size of the tasks and the speed of each VM. The justification can be verified over real-time cloud environment. In addition, the work can be improved through using dynamic workflow that allows more flexibility for the users to change the characteristics of the workflow tasks during the runtime.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

References

[1] A. H. Aljammal, A. M. Manasrah, A. E. Abdallah, and N. M. Tahat, “A new architecture of cloud computing to enhance the load balancingg,” *International Journal of Business Information Systems*, vol. 25, no. 3, pp. 393–405, 2007.

[2] J. Li, Z. Liu, X. Chen, F. Xhafa, X. Tan, and D. S. Wong, “L-EncDB: A lightweight framework for privacy-preserving data queries in cloud computing,” *Knowledge-Based Systems*, vol. 79, pp. 18–26, 2015.

[3] A. M. Manasrah, T. Smadi, and A. ALmomani, “A Variable Service Broker Routing Policy for data center selection in cloud analyst,” *Journal of King Saud University - Computer and Information Sciences*, vol. 29, no. 3, pp. 365–377, 2017.

[4] B. B. Gupta and T. Akhtar, “A survey on smart power grid: frameworks, tools, security issues, and solutions,” *Annales des Télécommunications*, vol. 72, no. 9-10, pp. 517–549, 2017.

[5] J. Yu, R. Buyya, and K. Ramamohanarao, “Workflow scheduling algorithms for grid computing,” in *Metaheuristics for scheduling in distributed computing environments*, pp. 173–214, Springer, 2008.

[6] A. Verma and S. Kaushal, “Cost-Time Efficient Scheduling Plan for Executing Workflows in the Cloud,” *Journal of Grid Computing*, vol. 13, no. 4, pp. 495–506, 2015.

[7] H. Ji, W. Bao, and X. Zhu, “Adaptive workflow scheduling for diverse objectives in cloud environments,” *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 2, Article ID e2941, 2017.

[8] A. M. Manasrah, “Dynamic weighted VM load balancing for cloud-analyst,” *International Journal of Information and Computer Security*, vol. 9, no. 1-2, pp. 5–19, 2017.

[9] W.-N. Chen and J. Zhang, “An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 39, no. 1, pp. 29–43, 2009.

[10] A. K. M. K. A. Talukder, M. Kirley, and R. Buyya, “Multiobjective differential evolution for scheduling workflow applications on global Grids,” *Concurrency and Computation: Practice and Experience*, vol. 21, no. 13, pp. 1742–1756, 2009.

[11] M. Wieczorek, A. Hoheisel, and R. Prodan, “Towards a general model of the multi-criteria workflow scheduling on the grid,” *Future Generation Computer Systems*, vol. 25, no. 3, pp. 237–256, 2009.

[12] P. Li, J. Li, Z. Huang, C.-Z. Gao, W.-B. Chen, and K. Chen, “Privacy-preserving outsourced classification in cloud computing,” *Cluster Computing*, pp. 1–10, 2017.

[13] C. Stergiou, K. E. Psannis, B.-G. Kim, and B. Gupta, “Secure integration of IoT and Cloud Computing,” *Future Generation Computer Systems*, vol. 78, pp. 964–975, 2018.

[14] K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam, “A genetic algorithm (GA) based load balancing strategy for cloud computing,” *Procedia Technology*, vol. 10, pp. 340–347, 2013.

[15] Z. Zhang and X. Zhang, “A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation,” in *Proceedings of the 2nd International Conference on Industrial Mechatronics and Automation (ICIMA ’10)*, vol. 2, pp. 240–243, May 2010.

[16] T. D. Braun, H. J. Siegel, N. Beck et al., “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems,” *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[17] M. Rana, S. Bilgaiyan, and U. Kar, “A study on load balancing in cloud computing environment using evolutionary and swarm based algorithms,” in *Proceedings of the 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies, ICCICCT 2014*, pp. 245–250, India, July 2014.

[18] Z. Zhu, G. Zhang, M. Li, and X. Liu, “Evolutionary multi-objective workflow scheduling in cloud,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2016.

[19] Y. Mao, X. Chen, and X. Li, “Max–Min task scheduling algorithm for load balance in cloud computing,” in *Proceedings of International Conference on Computer Science and Information Technology*, S. Patnaik and X. Li, Eds., vol. 225, pp. 457–465, Springer, New Delhi, India, 2014.

[20] P. Kumar and A. Verma, “Scheduling using improved genetic algorithm in cloud computing for independent tasks,” in *Proceedings of the 2012 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2012*, pp. 137–142, India, August 2012.

[21] L. Guo, S. Zhao, S. Shen, and C. Jiang, “Task scheduling optimization in cloud computing based on heuristic algorithm,” *Journal of Networks*, vol. 7, no. 3, pp. 547–553, 2012.

[22] L. Zhang, Y. Chen, R. Sun, S. Jing, and B. Yang, “A task scheduling algorithm based on PSO for grid computing,” *International Journal of Computational Intelligence Research*, vol. 4, no. 1, pp. 37–43, 2008.

[23] 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 *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications, AINA2010*, pp. 400–407, Australia, April 2010.

[24] H. Arabnejad and J. G. Barbosa, “A Budget Constrained Scheduling Algorithm for Workflow Applications,” *Journal of Grid Computing*, vol. 12, no. 4, pp. 665–679, 2014.

[25] H. Xu, B. Yang, W. Qi, and E. Ahene, “A multi-objective optimization approach to workflow scheduling in clouds considering fault recovery,” *KSII Transactions on Internet & Information Systems*, vol. 10, no. 3, 2016.

[26] S. Chitra, B. Madhusudhanan, G. R. Sakthidharan, and P. Saravanan, “Local minima jump PSO for workflow scheduling in cloud computing environments,” *Lecture Notes in Electrical Engineering*, vol. 279, pp. 1225–1234, 2014.

[27] Y. Ge and G. Wei, “GA-based task scheduler for the cloud computing systems,” in *Proceedings of the International Conference on Web Information Systems and Mining (WISM ’10)*, vol. 2, pp. 181–186, IEEE, October 2010.

[28] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, and T. Fahringer, “A multi-objective approach for workflow scheduling in heterogeneous environments,” in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012*, pp. 300–309, Canada, May 2012.

[29] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, “A market-oriented hierarchical scheduling strategy in cloud workflow systems,” *The Journal of Supercomputing*, vol. 63, no. 1, pp. 256–293, 2013.

[30] H. Ba_ali, *Workflow Load Balancing and Scheduling using Genetic Algorith (GA) and Particle Swarm Optimization (PSO) in Cloud Computing*, Yarmouk University, Irbid, Jordan, 2017.

[31] W. Zheng and R. Sakellariou, “Budget-Deadline Constrained Workflow Planning for Admission Control,” *Journal of Grid Computing*, vol. 11, no. 4, pp. 633–651, 2013.

[32] J. C. Jacob, D. S. Katz, T. Prince et al., *The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets*, Jet Propulsion Laboratory, National Aeronautics and Space Administration, Pasadena, Clif, USA, 2004.

[33] H. Magistrale, S. Day, R. W. Clayton, and R. Graves, “The SCEC southern California reference three-dimensional seismic velocity model version 2,” *Bulletin of the Seismological Society of America*, vol. 90, no. 6, pp. S65–S76, 2000.

[34] E. Deelman, K. Vahi, G. Juve et al., “Pegasus, a workflow management system for science automation,” *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[35] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, “A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis,” in *Workflows for e-Science*, pp. 39–59, Springer, 2007.

[36] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor, “High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs,” *PLoS ONE*, vol. 3, no. 9, Article ID e3197, 2008.

[37] A. Alajmi and J. Wright, “Selecting the most efficient genetic algorithm sets in solving unconstrained building optimization problem,” *International Journal of Sustainable Built Environment*, vol. 3, no. 1, pp. 18–26, 2014.

[38] W. Chen and E. Deelman, “WorkflowSim: A toolkit for simulating scientific workflows in distributed environments,” in *Proceedings of the 2012 IEEE 8th International Conference on E-Science, e-Science 2012*, USA, October 2012.

[39] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. de Rose, and R. Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[40] A. Ghorbannia Delavar and Y. Aryan, “HSGA: A hybrid heuristic algorithm for workflow scheduling in cloud systems,” *Cluster Computing*, vol. 17, no. 1, pp. 129–137, 2014.

[41] D. G. Amalarethinam and T. L. A. Beena, “Workflow Scheduling for Public Cloud Using Genetic Algorithm (WSGA),” *IOSR Journals (IOSR Journal of Computer Engineering)*, vol. 1, no. 18, pp. 23–27, 2016.