Received: 26 October 2016 | Revised: 17 September 2017 | Accepted: 4 October 2017

DOI: 10.1002/cpe.4368

WILEY

SPECIAL ISSUE PAPER

A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment

Fatemeh Ebadifard | **Seyed Morteza Babamir**

Department of Computer, University of Kashan, Kashan, Iran

Correspondence
Seyed Morteza Babamir, Department of Computer, University of Kashan, Kashan, Iran.
Email: babamir@kashanu.ac.ir

Funding information
University of Kashan, Grant/Award Number: 577242

Summary

Dynamic on-demand resource provisioning is one of the primary goals of the cloud computing task scheduling process. Task scheduling is a nondeterministic polynomial time (NP)-hard problem and is responsible for assigning tasks to virtual machines (VMs) in a way that increases the resource utilization and performance, reduces response time, and keeps the whole system balanced. In this paper, we present a static task scheduling method based on the particle swarm optimization (PSO) algorithm where the tasks are assumed to be non-preemptive and independent. We have improved the performance of the basic PSO method using a load-balancing technique. We have compared our proposed method with round robin (RR) task scheduling, improved PSO task scheduling and a load-balancing technique. The simulation results show that our method outperforms these algorithms by an increase of resource utilization of 22% and a decrease of makespan by 33%, compared with the basic PSO algorithm. The results illustrate that our proposed method converges to the near optimal solution faster than the basic PSO algorithm and is more efficacious with more tasks.

KEYWORDS

cloud computing, load balancing, makespan, particle swarm optimization, resource utilization, task scheduling

1 | INTRODUCTION

Cloud computing employs parallel and distributed computing concepts to provide users with shared resources through the internet.[1] A "pay as you go" model has made cloud services ubiquitous. Cloud providers, service providers, and users are three distinct entities incorporating in the software as a service (SaaS) level provisioning. Cloud providers exploit virtualization technology and supply their clients with computing resources in the form of virtual machines (VMs). Service providers, on the other hand, benefit from these VMs to provide users with application level services. To assign users' tasks to VMs, reduce the response time, provide promising quality of service (QoS), and make the most of the resource, service providers exploit task scheduling techniques. Therefore, the task scheduling algorithm is one of the core elements of each cloud infrastructure.[1]

Although there are several scheduling methods for a variety of computing environments, they should be adjusted for a cloud environment if they have not been specifically designed for it. For example, there are some independent and homogeneous computers in a cluster, which together act as a single system, and these systems use their local resources, while a cloud environment contains a set of distributed heterogeneous resources and users can use all of them; thus, a scheduling algorithm that can be good for a cluster may not be appropriate for a cloud environment. The problem space should be mapped into the algorithm elements so that the algorithm can be applied to the structural features of the cloud environment.

The increase in heterogeneous VMs and the diversity of the size of tasks leads to a huge number of task arrangements. Finding the suitable permutation with the minimum completion time among all permutations is a NP-hard problem. Many related studies have been conducted using meta-heuristic algorithms in conjunction with scheduling in a cloud environment; however, in this paper, we have presented a scheduling method based on the PSO algorithm[2] that has been improved using a load-balancing method, leading to the faster convergence of the optimal solution. In fact, we can use an appropriate load-balancing technique and a meta-heuristic algorithm to consider two objectives meeting the interest of service providers (resource utilization) and increasing the satisfaction of users (makespan reduction).

In summary, our main objectives in this article are as follows:

1. To provide an improved load-balancing algorithm for the suitable distribution of requests regarding the compatibility of any request by the machine. The difference between our proposed load-balancing algorithm and previous studies (e.g., Krishna[3] and Mittal and Katal[4]) is that we chose the VM with the highest compatibility for hosting a request (see Section 3.2). Using this relationship increased the load balancing between VMs over the previous studies leading to a greater reduction in makespan.
2. To consider the vector of resources (central processing unit [CPU], storage, random access memory [RAM], and bandwidth) rather than just processor resources to check the compatibility of users' requests with VMs. This makes our model more realistic for the cloud environment.
3. To consider the interests of service providers and users simultaneously by defining a fitness function to decrease makespan and increase the resource utilization.
4. To reduce the complexity of previous methods using a single-objective algorithm for task scheduling problems with simultaneous consideration of the interests of service providers and users.
5. To provide the effective combination of the PSO algorithm and a load-balancing algorithm.

The results show that our method has advantages over the basic PSO task scheduling algorithm by increasing resource utilization by 22% and reducing makespan by 33% at the same time. Our innovation includes presenting a thorough survey of the related literature along with a novel task scheduling method using the PSO algorithm improved using a load-balancing technique.

The rest of this paper is organized as follows: Section 2 presents related work. Section 3 introduces the PSO algorithm briefly and provides the problem formulation and method description in detail. Section 4 demonstrates simulation results and evaluation of the proposed method, and Section 5 deals with conclusions and future work.

2 | RELATED WORK

In cloud computing terminology, optimal assignment of requests to data center recourses is called task scheduling. Requests are assigned to different kinds of resources considering the service they might need. There are two groups of limitations: client-defined limitations and provider-defined limitations. A large body of research has been carried out concerning the task scheduling area and can be categorized in two main groups: (1) those that deploy a heuristic/meta-heuristic algorithm and (2) those that combine a heuristic/meta-heuristic algorithm with a non-heuristic one. We discuss some of the most important related work in each of these groups along with some of the load-balancing algorithms for the cloud computing environment to state why we used a combination of the PSO algorithm with a load-balancing technique.

The first group consists of work exploiting heuristic/meta-heuristic algorithms. Since task scheduling is an optimization problem belonging to the class of NP-hard problems, it cannot be solved using traditional methods in a polynomial time.[5] Therefore, heuristic or meta-heuristic methods should be used.[5] However, solutions generated by heuristic methods are often trapped in local optima, which is not near global optima.[5] Accordingly, meta-heuristic algorithms, such as the genetic algorithm (GA), ant colony, and particle swarm optimization (PSO), are the best candidates for task scheduling problems, specifically when we face a diversity of requests and VMs.[6-8] It has been proven that meta-heuristic algorithms are the most effective approach to escaping the local optima.[9,10] In addition, heuristic algorithms are problem-dependent techniques while meta-heuristic ones are problem-independent techniques.[11]

Our motivation to use a PSO-based meta-heuristic algorithm is explained in the following section. Mandal and Acharyya presented a new scheduling method using a firefly meta-heuristic algorithm to schedule requests in the cloud environment, considering the objective of reducing makespan.[12] They compared their results with the cuckoo search algorithm and simulated annealing (SA) and showed that their algorithm has better performance. However, Jones and Boizant compared the performance and efficiency of PSO with the firefly algorithm; their results showed that PSO is nearly twice as fast as firefly in terms of optimal response and has a lower rate of overheads compared with firefly.[13] The selection of the PSO algorithm for the scheduling problem is more suitable than the firefly algorithms. Moreover, PSO has better performance when compared with ant colony and SA algorithms based on comparisons carried out in previous studies.[14]

Therefore, we used a PSO-based algorithm in our proposed method. Since our proposed method is a combination of the PSO algorithm and a new load-balancing technique, we deal with PSO scheduling methods and load-balancing techniques used in recent years for task scheduling. Aiming to reduce computation and communication costs, Pandey et al. proposed a PSO-based task scheduling algorithm for workflow application.[15] Their results show the proposed method leads to lower computational costs when compared with a simple "best resource selection" algorithm. The main drawback of this method is that it sacrifices the completion time of tasks to decrease the computational cost. Zhang et al. presented a PSO-based solution for scheduling tasks in a grid environment and showed that their PSO-based method resulted in a better schedule than the GA-based method in a grid environment.[16] Guo et al. proposed a PSO-based task scheduling method in cloud computing aimed at reducing completion time by minimizing execution and transmission times.[17]

The second group contains works benefit from combining a meta-heuristic algorithm with a meta-heuristic or non-heuristic one. Zhan and Huo[14] proposed a mixed method using both PSO and simulated annealing (SA) algorithms. The proposed method tends to reduce tasks' average

execution time and increase convergence speed to the optimal solution. This study shows that the improved PSO-based method has better efficiency comparing to the genetic algorithm, simulated annealing, and ant colony, although combining PSO and SA algorithms results in more computational complexity. Mixing PSO with SA algorithms, Kaur and Sharma[18] proposed a new task scheduling method for cloud environment. Their primary goal was to optimize resource utilization and maximize providers' profit. In our method, in addition to taking into account the interests of service providers, we considered requirements of quality service for users. Abdi et al.[19] proposed an improved PSO-based task scheduling method using "shortest job to fastest processor" algorithm for generating the initial population. The method's goal was to reduce makespan. Their results show that the proposed method has lower makespan comparing to GA-based and simple PSO-based solutions.

Some meta-heuristic studies for task scheduling in the cloud computing environment considered only the benefits for provider's benefits or client satisfaction, not both. These include the genetic algorithm (GA),[6] ant colony,[7,8] and PSO.[15,16] Others have used a meta-heuristic algorithm to improve another meta-heuristic or a non-heuristic algorithm.[12,18,19]

Some other meta-heuristic studies used multi-objective scheduling methods to consider both users' and providers' interests. Bilgaiyan et al. presented a workflow scheduling algorithm using the cat swarm optimization algorithm for workflow scheduling in the cloud environment.[20] They aimed to simultaneously reduce makespan, cost, and idle time of the processor. By comparing their results with those of the multi-objective PSO (MOPSO) algorithm, they showed that their method could work better. Udomkasemsub et al. presented an artificial bee colony (ABC) multi-objective scheduling method to consider the conflicting objectives of cost and makespan using the Pareto optimizer algorithm.[21] Wu et al. proposed a multi-objective scheduling method using the random-drift PSO (RDPSO) algorithm for scheduling workflow in the cloud aimed at reducing the cost and makespan.[22] Yassa et al. presented a PSO-based multi-objective workflow scheduling method to reduce costs, energy, and makespan.[23] They used a technique to reduce the cost and then compared the proposed method to the heterogeneous earliest finish time (HEFT) method, which is a greedy-based heuristic method. Most previous studies have focused on the cost and makespan as two important objectives in scheduling.

In addition to considering the interests of users, Khalili and Babamir considered the requirements of QoS for service providers.[24] They presented the gray wolf optimizer (GWO)-based algorithm as a multi-objective scheduling algorithm with a Pareto front with the aim of reducing makespan and cost, and increasing throughput. They compared their results with those of the strength Pareto evolutionary algorithm (SPEA2).

Even though the use of multi-objective algorithms considers the interests of users and service providers simultaneously, it has higher complexity, which makes obtaining a set of near optimal solutions more time consuming than a single objective method; therefore, we used a single objective algorithm. The difference between our work and previous studies is that by exploiting a load-balancing mechanism, we can provide a new scheduling method based on the single-objective PSO algorithm including an appropriate objective function. The proposed method can provide a faster response time for users with the satisfaction of improved resource utilization for service providers. Moreover, the load-balancing mechanism we use enables us to distribute the workload evenly among VMs leading to an increase of resource utilization and makespan reduction simultaneously. Using a single-objective algorithm to concurrently reduce the makespan and increase resource utilization mitigates computational complexity and thus increases the speed of the scheduling algorithm.

Load-balancing algorithms can be classified into static and dynamic classes. The static algorithms are suitable for homogenous and stable environments. In this group of algorithms, only initial attributes and capacities of resources are considered. The dynamic algorithms, on the contrary, are suitable for heterogeneous and dynamic environments. Dynamic algorithms tend to find better solutions regarding resource attributes both in their initial state and during the run-time.[25]

Now, we address the load-balancing algorithms. A lot of related work has been done on load balancing with different objectives such as reducing response time,[3,26,27] scalability,[26,28] and reducing migration time of requests.[28,29] Since our objective in load balancing is to reduce the response time, we state some new studies in this field. Nakai et al. have presented a load-balancing mechanism based on the reserve policy to disperse requests among replicated servers.[26] This allowed overloaded servers to reserve a part of the capacity of remote servers before receiving a new request, and, if requests were higher than the amount of average capacity of remote servers, some of the requests were discarded. Simulation results showed that their proposed method reduces the response time and increases overload. Even though it reduced the response time, some requests were discarded; therefore, this method was not appropriate for our work.

Daraghmi and Yuan tried to improve the performance of the load-balancing algorithm.[27] They considered the network structure in addition to the technical factors of load balancing. Thus, they provided a method which was efficiently applicable in the network and reduced the level of overhead in the network in addition to reducing response time. This proposed algorithm is also not suitable for the cloud environment due to its low productivity and lack of scalability.

Mittal and Katal proposed a method for scheduling requests with the objective of reducing makespan using a load-balancing algorithm for the cloud platform.[4] They compared their results to those of some most well-known algorithms of load balancing such as Min-Min,[30] Max-Min[31] and improved algorithms of Max-Min,[32,33] and robust alternative splicing analysis (RASA).[34] Simulation results showed that their algorithm had better performance than other related algorithms. We compare our proposed algorithm with this algorithm.

Our load-balancing algorithm is inspired by the honeybee behavior proposed in previous study[3] by Krishna for the reduction of the makespan. Assigning requests to VMs based on the RR algorithm and moving requests from overloaded VMs to those with alight caseload are the main ideas in his work. In our approach, an improvement of the honeybee-based load-balancing algorithm is used.

In this paper, we propose a PSO-based task scheduling algorithm by benefiting from a load-balancing algorithm to generate a more suitable initial population and choosing a more appropriate goal function. Despite all the above mentioned works which consider either clients' benefits or providers' benefits, our method decreases makespan duration and increases resource utilization at the same time.

3 | PSO-BASED TASK SCHEDULING

First, we explain the static PSO-based task scheduling method; then, we present the problem formulation and our proposed method based on PSO in detail.

3.1 | Classic PSO

PSO is a meta-heuristic algorithm introduced by Eberhart and Kennedy[2] at 1995. It simulates the social behavior of animals like flock of birds or shoal of fishes. Each particle in problem (search) space is similar to a bird or fish in the nature and has a position vector and a velocity vector. Every particle can be considered as a possible solution for the problem, and it searches the problem space for the optimized solution. Position of each particle is affected by its own best position so far in the problem space (*pbest*) and the position of the best particle in the problem space (*gbest*). Using a fitness function, performance of each particle is evaluated at the end of every iteration. Therefore, definition of fitness function is problem specific. Figure 1 shows the PSO algorithm where particles' positions and velocity vectors are calculated in every iteration using Equations 1 and 2.

$$v_i^{k+1} = wv_i^k + c_1\,\mathrm{rand}_1 \times \left(\mathrm{pbest}_i - x_i^k\right) + c_2\,\mathrm{rand}_2 \times \left(\mathrm{gbest} - x_i^k\right), \tag{1}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1}, \tag{2}$$

where

v_i^k	velocity of particle *i* at iteration *k*
v_i^{k+1}	velocity of particle *i* at iteration *k* + 1
w	inertia weight
c_j	acceleration coefficient, *j* = 1, 2
rand_i	random number between 0 and 1, *i* = 1, 2
x_i^k	current position of particle *i* at iteration *k*
pbest_i	best position vector of particle *i*
gbest_i	position vector of the best particle in the population
x_i^{k+1}	position vector of particle *i* at iteration *k* + 1

3.2 | The proposed load-balancing algorithm

Load balancing is one of the most important aspects of task scheduling problems in the cloud environment. Load balancing is a technique which disperses workload among multiple servers in a way that maximum resources are efficiently used and we can have the best response time and throughput. An appropriate load-balancing algorithm can (1) prevent overload; (2) reduce the waiting time of requests; and (3) increase the efficiency of VMs.

Our proposed load-balancing algorithm is based on the honeybee behavioral model of James and Brian Johnson.[35] In essence, the tasks of honeybees and VMs are related to food sources. Allocating a task to a VM is similar to the exploring of food sources by bees. Overloading of VMs is similar to obtaining honey from an empty food source, and therefore, there is the need to move a task from an overloaded VM to one with a light caseload, comparable to the exploration of a bee to find a new food source, removing tasks from the heavy one. To move tasks from an overloaded VM, an appropriate VM with a light caseload should be found. A lack of load balancing can be discovered in the cloud environment through calculating the standard deviation of the system's load (Equation 3).

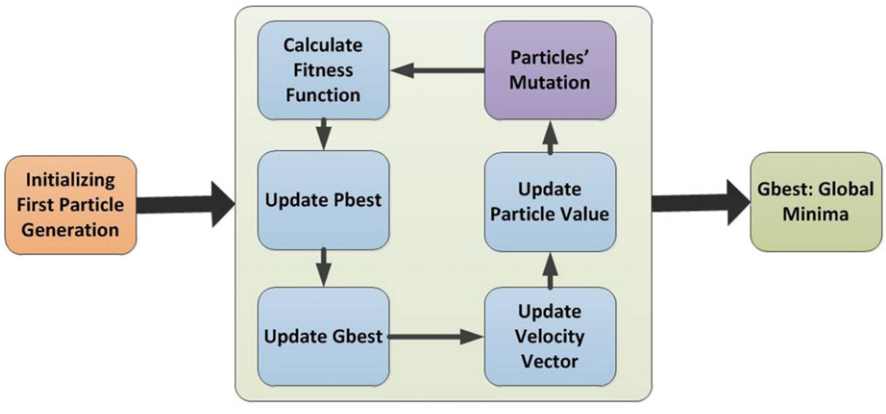

FIGURE 1 The PSO algorithm

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(PT_i - PT)^2}, \tag{3}$$

where m is the number of VMs, PT_i is the processing time of ith VM, and PT is the processing time of the whole system. The system is unbalanced if its standard deviation is greater than the desired threshold and we need to move requests from overloaded machines to those with a light caseload. VMs are divided into three categories of load: underloaded, balanced, and overloaded, where requests are moved from overloaded VMs to underloaded ones using the following method.

If CPU, RAM, storage, and bandwidth are resource types of a system, we show (1) resource usage pattern of task T_i as vector $\overrightarrow{T_i}$ whose four members contain the usage amounts of CPU, RAM, storage, and bandwidth; and (2) usage resources pattern of n tasks as $\overrightarrow{R_{used}}$ (Equation 4). The available resources pattern is shown as $\overrightarrow{R_{avail}}$ (Equation 5) where $\overrightarrow{R}$ denotes the vector of total resources.

$$\underset{R_{used}}{\rightarrow} = \underrightarrow{\sum_{i=1}^{n} T_i}, \tag{4}$$

$$\underset{R_{avial}}{\rightarrow} = \underset{R}{\rightarrow} - \underset{R_{used}}{\rightarrow}. \tag{5}$$

Based on Equation 6, we use cosine similarity[36] to calculate the similarity between resource usage pattern vector of task T_k (indicated by $\overrightarrow{T_k}$) and total resource vector $\overrightarrow{R}$. In Relation 6, the smaller value the angle has, the greater similarity (better compatibility) exists between $\overrightarrow{T_k}$ and $\overrightarrow{R}$.

$$angle = \cos^{-1}\left(\frac{\underset{T_K}{\rightarrow} . \underset{R_{avail(i)}}{\rightarrow}}{\left|\underset{T_k}{\rightarrow}\right|\left|\underset{R_{avail(i)}}{\rightarrow}\right|}\right). \tag{6}$$

We use the *angle* for compatibility of a user's request (task) and the VM on which the request could be scheduled. In other words, the resource usage pattern of a task has the most affinity with available resources pattern on the VM. In the proposed method, we considered utilization of the resources in addition to compatibility between the requested resources and the available resources of VMs. Thus, we select an underloaded VM with the best compatibility for task T_k using Equation 7 where value α is set to 0.5 in this paper. We obtain the value of the angle in each execution period of our algorithm.

$$\text{Compatibility} = \alpha \times angle + (1-\alpha) \times \text{resource utilization}. \tag{7}$$

We used our proposed load-balancing method to prevent the overload of VMs and to select the most appropriate VM to make the transition requests to underloaded VMs based on resource usage pattern of requests and available resources pattern of VMs, as well as the utilization of VMs.

3.3 | Task-resource scheduling formulation

Although there are attempts to trade off makespan reduction and increase of resource utilization as conflicting objectives,[37,38] there is a reverse linear relationship between these two objectives (Equation 9) so there is no need to trade-off and use a multi-objective algorithm; accordingly, we use a single-objective one. According to Equation 9,[1] resource utilization of a VM is defined as a proportion of the overall completion time of tasks (makespan) during which the VM is busy. Considering the use of a load-balancing algorithm to reduce makespan, we accordingly increased the resource utilization. Therefore, we face the optimization of one objective.

To formulate the problem, we denoted tasks: the set of $\{T_1, T_2, T_3, \ldots, T_n\}$, which are non-preemptive and independent, and VMs as the set of $\{VM_1, VM_2, VM_3, \ldots, VM_m\}$. If we denote the processing time of task T_i on VM_j as PT_{ij} and the completion time of VM_j as CT_j,[3] Equations 8 and 9 define the overall makespan and the VM utilization. The objective is to minimize the overall makespan to maximize the resource utilization according to Relation 9. The average utilization is defined by Equation 10.[1]

$$Makespan = \max\{CT_j | j = 1, 2, \ldots, m\}, \tag{8}$$

$$Utilization_{VM_j} = \frac{\sum_{i=1}^{n} PT_{ij}}{makespan}, \tag{9}$$

$$Average\ utilization = \frac{\left(\sum_{j=1}^{m} Utilization_{VM_j}\right)}{m}. \tag{10}$$

Next, we define the fitness function of the PSO algorithm as Equation 11 according to which a particle has a better position, if it has a lower fitness value.

$$\text{Fitness function} = \frac{\text{makespan}}{\text{Average utilization}}. \quad (11)$$

Using the following example, we show how the makespan reduction can lead to increased resource utilization in the presence of load balancing. Assume that we want to schedule a workflow with five tasks on three VMs. Table 1 shows the time that each VM takes to complete the tasks. If tasks are allocated to machines according to the second column of Table 2, the last column of the table shows the utilization of each machine based on Relations 9 and 10.

Table 2 shows that the makespan is 11 seconds where the average utilization is (0.45 + 1 + 0.18)/3 = 0.54. Consider another task scheduling (Table 3) where we have both a reduction of makespan by 9 seconds and an increase of average utilization by 0.8 (0.6 + 1 + 0.8)/3 = 0.8. It can be concluded that a load-balancing algorithm can enable us to assign tasks to VMs so that both the makespan is reduced and utilization is increased. We reduced makespan in the proposed algorithm using a load-balancing algorithm, and, based on the definition of utilization (Relation 9), the reduction of makespan increases utilization. Indeed, through selection of an appropriate objective function, we could select solutions with lower makespan and higher resource utilization. The details of the proposed method are explained in the next subsection.

Note that in our method, we considered acceleration coefficients as $c_1 = c_2 = 1.49445$ and random numbers as $rand_1$ and $rand_2$ between 0 and 1 (see Section 3.1 for the explanation of these parameters).

3.4 | The proposed PSO-based load balancing

Detailed explanation of the proposed method and pseudocode of its algorithm are presented in this section. The steps of the algorithms are as following:

Step 1. Defining of particles, position vectors, and velocity vectors

For a problem with n tasks, we define each particle as an N-dimensional vector $X = (X_1, X_2, \ldots, X_n)$, where X_i ($i \in \{1, 2, \ldots, n\}$) represents index of the VM on which task i would be processed. Position and velocity of particles are initialized randomly. For example, Figure 2 shows the initialization of a particle's position vector for 6 tasks and 3 VMs.

Step 2. Improving position vector of particles and balancing load of VMs

Load of every VM at time t is calculated using Equation 12 where $NT(t)$ is defined as the number of tasks in the VM's queue at time t and $SR_{VM_j}(t)$ is defined as service rate of VM_j at time t.

TABLE 1 Completion time of tasks on by VMs

	TASK1	TASK2	TASK3	TASK4	TASK5
VM1	5	6	2	1	9
VM2	3	2	3	0.5	7
VM3	7	6	8	2	13

TABLE 2 The first task scheduling

Machine	Assigned Tasks	Completion Time	Resource Utilization
VM1	T3, T5	2 + 9 = 11 s	$\frac{11}{11} = 1$
VM2	T1, T2	3 + 2 = 5 s	$\frac{5}{11} = 0.45$
VM3	T4	2 s	$\frac{2}{11} = 0.18$

TABLE 3 The second task scheduling

Machine	Assigned Tasks	Task Completion Time	Resource Utilization
VM1	T1, T4	5 + 1 = 6 s	$\frac{6}{9} = 0.6$
VM2	T2, T5	2 + 7 = 9 s	$\frac{9}{9} = 1$
VM3	T3	8 s	$\frac{8}{9} = 0.8$

Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
VM 2	VM 1	VM 1	VM 3	VM 2	VM 2

FIGURE 2 Values of a randomly initialized particle

$$L_{VM_i,t} = \frac{NT(t)}{SR_{VM_j}(t)}. \tag{12}$$

Comparing the load of each VM to average load of the whole system, the VM's state is recognized as underloaded (VM load < average system load), overloaded (VM load > average system load), or balanced (VM load = average system load); then, the tasks of overloaded VMs will be replaced to underloaded VMs. We use proposed load-balancing algorithm in Section 3.2 for transmit requests from overloaded VM to underloaded VM. The task replacement procedure will be continued until either all the VMs are balanced or no underloaded VM exists to replace tasks of overloaded VMs to them.

Step 3. Calculating fitness function and specifying *pbest*

Fitness value of each particle will be calculated using Equation11. Comparing the current fitness value of each particle with its own *pbest* value, the particle's best position (*pbest*) will be distinguished.

Step 4. Specifying *gbest*

Comparing the current fitness value of the whole population together, the lowest fitness value will be specified as global best position (*gbest*).

Step 5. Updating particles' position and velocity vectors

Since the PSO algorithm has been considered for continuous space, the values of particle positions cannot be used for the task scheduling problem in a cloud environment. Therefore, we used the method in Mohamad et al.[39] to convert continuous values of position vector to discrete values. In the proposed method, the position of each particle is determined as a vector with a length of the number of VMs and each member of the vector contains a binary bit string with a length of the number of requests. A bit with the value of 1 in the *i*th place of a bit string of a VM denotes the assignment of the *i*th request to the VM. The velocity vector of each particle is calculated according to Relation 1. Finally, the position vector of each particle is calculated using the sigmoid function stated as Equation 13.

$$sig\left(V_i^{k+1}\right) = \frac{1}{1+e^{-V_i^{k+1}}}, \tag{13}$$

$$\text{If } V_i^{k+1} > \text{ran.}nextInt(2),\ x_i^{k+1} = 1. \quad \text{Otherwise,} \quad x_i^{k+1} = 0.$$

Step 6. Terminating condition

Steps 3 to 6 will be repeated until the maximum number of iterations is reached. Pseudocode of our proposed algorithm is shown in Figure 3. Position initialization of our method has an advantage over the base PSO task scheduling method because base PSO task scheduling method initializes particles' position vectors randomly, but our proposed method executes a load-balancing step after random initialization to improve the particles' positions. Therefore, each initial particle can present a proper solution which will be improved by moving in the problem space. The simulation results, presented in Section 4, show that compared to the PSO-based task scheduling method, our method leads to more efficient completion time and resource utilization.

4 | PERFORMANCE EVALUATION

In this section, we present the simulation setup and evaluation of the proposed algorithm based on the results of conducted tests. It is worth mentioning that the experiments are conducted in SaaS level.

Input: $\mathrm{VM} = \{\mathrm{VM}_1, \mathrm{VM}_2, \mathrm{VM}_3, \ldots, \mathrm{VM}_j\}$, $\mathrm{Task} = \{\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \ldots, \mathrm{T}_i\}$

Output: best position of Tasks on the VM(Gbest)

Start:

1: set particle dimension equal to the size of ready tasks, Initialize particles position and velocity randomly,

2 : for each particle, run load balancing algorithm for balancing particle position,

3: for all particles, calculate its fitness value by in Equation 11,

If (fitness value < **pbest**)

set the current fitness value as the new **pbest,**

4 : select the best particle's position as gbest,

5: for all particles, calculate velocity using Relation 1 and update their positions using Equation 13,

6: repeat steps 3 to 6 if termination condition or maximum iteration is not satisfied.

FIGURE 3 The PSO-based load-balancing and task scheduling pseudocode

4.1 | Statistical analysis

All experiments were carried out 10 times, and the mean and standard deviation were obtained. Variance analysis of independent data with three replications was performed using the ANOVA method and a completely randomized factorial design. All statistical analyses were performed with the SAS 9.1 software (SAS, Cary, NC, USA) where the least significant difference (LSD) was considered with ($P < .05$). We carried out the Fisher test for all experiments.

4.2 | Simulation results

For our experiment, we have used CloudSim[40] to simulate cloud computing SaaS level. This simulator gives capability of simulating a virtualized environment, and it supports on-demand resource provisioning. We have extended the CloudSim simulator for modeling our method. The purpose of our method, as mentioned before, is to present an efficient PSO-based task scheduling algorithm for cloud computing environment which can reduce completion time of the longest task (makespan) and increase the average resource utilization of the cloud data center. We have defined 10 particles for the PSO population. The termination condition is set to 100 iterations. To analyze our method, several experiments are conducted in two different test setups. Test setups and experiments are described in the rest of this section.

- Test setup 1

In the first test setup, we defined a cloud data center consisting of three hosts, each capable of supporting virtualization technology and sharing its resources among several VMs. Hardware details of hosts are presented in Table 4. Sixteen VMs are hosted and run on three hosts consisting of distinct details. Each VM executes applications with a different number of instructions varying from 500 to 4500. We used a simulated workload for these series of tests. Parameters for the simulation of the PSO algorithms are given in Table 5.

Since our proposed method is a combination of the PSO algorithm and a new load-balancing technique, we compared our method with load-balancing techniques and recent PSO task scheduling methods: (1) simple round robin (RR) scheduling; (2) improved PSO scheduling[41]; (3) the PSO-

TABLE 4 Hosts' technical specifications

HostId	Processing Cores	Processing Speed, Mips	RAM, MB	Hard, MB	Bandwidth, Mbps
1	4	5000	204 800	1 048 576	102 400
2	2	2500	102 400	1 048 576	102 400
3	1	1000	51 200	1 048 576	102 400

TABLE 5 The algorithm parameters

Parameter	Value
Population size (no. of solution)	10
Maximum iterations	100
C_1, C_2	1.49445
r_1, r_2	Random numbers between 0 and 1

based method consisting of improvement of the convergence of PSO-based method through taking advantageous of the SA algorithm[14]; (4) honeybee load-balancing algorithm[3]; and (5) RASA improved load-balancing algorithm.[4]

Figure 4 shows the makespans obtained using the 5 methods stated above and ours for different number of tasks.

Since the honeybee load balancing[3] uses the RR task scheduling at first, its performance depends on the RR initial task assignment. As the figure shows, the honeybee load-balancing method is more efficient than the improved PSO[41] because the honeybee load balancing is capable of distinguishing the overloaded and light caseload VMs while the improved PSO does not have such capability. According to the figure, our method is qualified to balance the load of each particle in the beginning of every iteration; therefore, it provides a better makespan than the other methods especially with more tasks. This is because the less number of tasks is, the less number of machines becomes overloaded; therefore, the existence of load-balancing and task scheduling methods is not of great importance but in situations with large number of tasks. The proposed method affects the makespan more effectively than others.

Table 6 presents the makespan value for each algorithm with the different number of tasks. The values in this table denote the mean value obtained from 10 executions. The superscript characters indicate the significant class that the value belongs to. Each class shows a range of numbers where "a" and "p" denote the highest and the lowest ranges, respectively. For the makespan, a lower class is more desirable. The double superscript characters such as "ml" denote that the value belongs to both classes "m" and "l". Notation "±" denotes that the deviation has exists between the values obtained from the three executions based on the LSD (i.e., $\alpha = 0.05$).

Consider that the lower class a method belongs to, the lower makespan it has. According to Table 6, there is not a significant difference between our method and Zhan and Huo[14] when the number of requests is less than 20 because both belong to class "p". However, our method performs better than others when the number of tasks is more than 20 because, in this case, results of our method belong to lower classes than others. The more requests (tasks) we have, the more significant the difference between our method and others; the most significant difference is observed when the number of tasks is 80.

Figure 5 represents response times for the different numbers of tasks for the RR, improved PSO,[14,41] honeybee load balancing,[3] and the proposed algorithms. Our method provides a lower response time in comparison with the three other methods. Table 7 presents the response time value for each algorithm with a different number of tasks. The explanation for this table is the same as the one for Table 6. According to

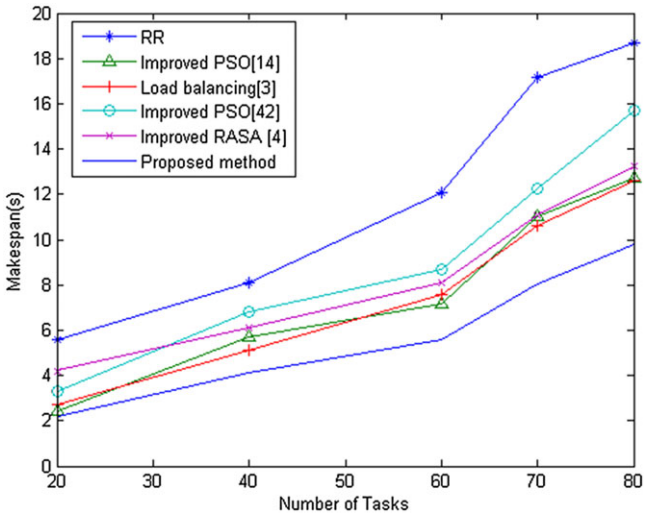

FIGURE 4 Makespan comparison

TABLE 6 Makespan values for different methods

		Number of Tasks				
		20	40	60	70	80
Simple	Makespan	5.6 ± 0^{ml}	8.1 ± 0^{hi}	12.1 ± 0^{e}	17.15 ± 0^{b}	18.66 ± 0^{a}
Improved PSO[12]		2.42 ± 0.14^{p}	5.31 ± 0.4^{ml}	7.1 ± 0.34^{kj}	11.11 ± 0.29^{f}	12.4 ± 0.35^{de}
Load balancing[24]		2.71 ± 0^{op}	5.1 ± 0^{m}	7.6 ± 0^{ij}	10.6 ± 0^{f}	12.6 ± 0^{de}
Improved PSO[41]		3.6 ± 0.3^{o}	6.48 ± 0.68^{k}	8.6 ± 0.7^{h}	12.58 ± 1.03^{e}	15.11 ± 1.19^{c}
Improved RASA[42]		4.21 ± 0^{n}	6.1 ± 0^{l}	8.1 ± 0^{hi}	11.1 ± 0^{f}	13.27 ± 0^{d}
Proposed method		2.13 ± 0.04^{p}	4.1 ± 0.032^{n}	5.6 ± 0.06^{ml}	8.01 ± 0.04^{hi}	9.8 ± 0.077^{g}

Values correspond to mean ± standard error of three ($n = 10$) measurements. Within each column, the superscript letters indicate significant difference according to least significant difference ($\alpha = 0.05$).

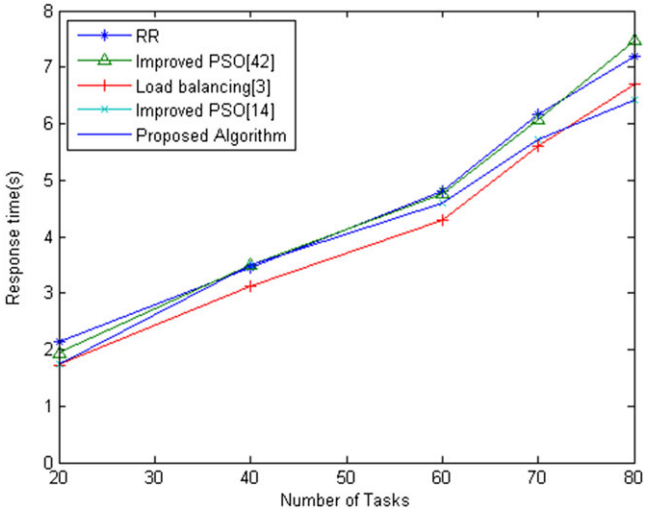

FIGURE 5 Response times obtained using the 4 methods and ours

TABLE 7 Response time values for different methods

		Number of Tasks				
		20	40	60	70	80
RR	Response time	2.14 ± 0^{l}	3.36 ± 0.14^{j}	4.81 ± 0^{h}	6.16 ± 0^{e}	7.2 ± 0^{b}
Improved PSO[41]		2.07 ± 0.32^{m}	3.5 ± 0^{j}	4.69 ± 0.2^{h}	6.08 ± 0.26^{e}	7.39 ± 0.28^{a}
Load balancing[24]		1.73 ± 0^{m}	3.13 ± 0^{k}	4.30 ± 0^{i}	5.6 ± 0^{gf}	6.70 ± 0^{c}
Improved PSO[12]		1.7 ± 0.88^{m}	3.35 ± 0.16^{j}	4.54 ± 0.19^{h}	5.83 ± 0.22^{f}	6.6 ± 0.37^{d}
Proposed method		1.48 ± 0.22^{n}	3.08 ± 0.1^{k}	4.13 ± 0.06^{i}	5.54 ± 0.15^{g}	6.47 ± 0.13^{cd}

Values correspond to mean ± standard error of three (n = 10) measurements. Within each column, the superscript letters indicate significant difference according to least significant difference (α = 0.05).

Table 7, our method performs better than others when the number of tasks is 20, and the same as Krishna[3] and better than others when they are more than 20.

Comparison between resource utilization among the proposed method, the improved PSO algorithm,[41] and RR algorithm is shown in Figure 6. It is evident that both improved PSO-based task scheduling methods result in higher resource utilization when compared with the RR algorithm. Because of its ability to balance the load of VMs, our proposed method has better performance in comparison with the improved PSO algorithm. Table 8 presents the resource utilization value for each algorithm with a different number of tasks. The explanation for this table is the same as the one for Table 6, with the difference that the higher classes are more desirable. According to this table, results of our proposed method belong to higher classes than others.

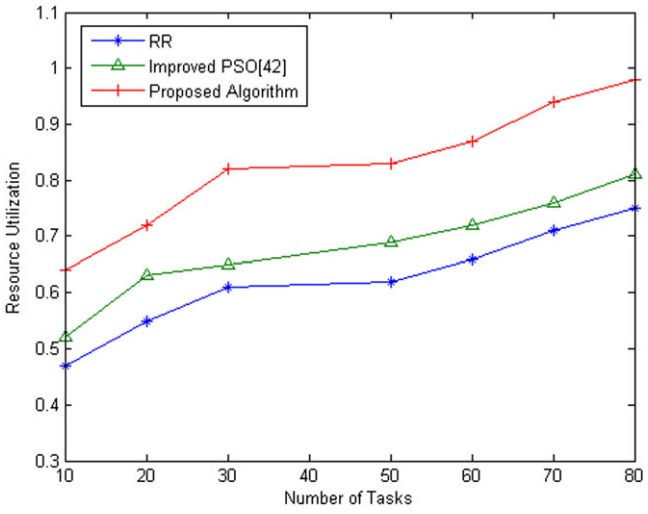

FIGURE 6 Resource utilization values obtained using the 2 methods and ours

TABLE 8 Resource utilization values for different methods

		Number of Tasks						
		10	20	30	50	60	70	80
RR	Resource utilization	0.47 ± 0^{k}	0.55 ± 0^{j}	0.61 ± 0^{i}	0.62 ± 0^{i}	0.66 ± 0^{h}	0.71 ± 0^{g}	0.75 ± 0^{ef}
Improved PSO[41]		0.52 ± 0.04^{j}	0.61 ± 0.03^{hi}	0.66 ± 0.02^{h}	0.67 ± 0.05^{g}	0.71 ± 0.29^{fg}	0.77 ± 0.03^{e}	0.83 ± 0.03^{d}
Proposed method		0.65 ± 0.02^{hi}	0.73 ± 0.03^{fg}	0.85 ± 0.02^{d}	0.84 ± 0.02^{d}	0.87 ± 0.01^{c}	0.93 ± 0.017^{b}	0.98 ± 0^{a}

Values correspond to mean ± standard error of three (n = 10) measurements. Within each column, the superscript letters indicate significant difference according to least significant difference (α = 0.05).

Figure 7 illustrates the convergence rate comparison of the improved PSO[41] and our proposed algorithm. It is clearly observable that our proposed method reaches to its global minima in the 30th iteration and the makespan does not change afterwards. The improved PSO algorithm, however, finds its global minima after 60 iterations. Therefore, our proposed task scheduling method is more efficient from a computational complexity point of view.

- Test setup 2

To analyze performance of the proposed method under real workloads, we have conducted another experiment with a new configuration. We considered 4 hosts on which 40 VMs are running and working in a cloud data center. Table 9 shows the technical details of the hosts, and Table 10 does hardware details of VMs.

We have used a workload that was logged by NASA Ames Research Center from October to December in 1993.[43] It contains 42 240 jobs. Each job is converted to a Cloudlet regarding to its completion time and processing rate of existing processors. Each Cloudlet is a task that can be used in CloudSim simulator.

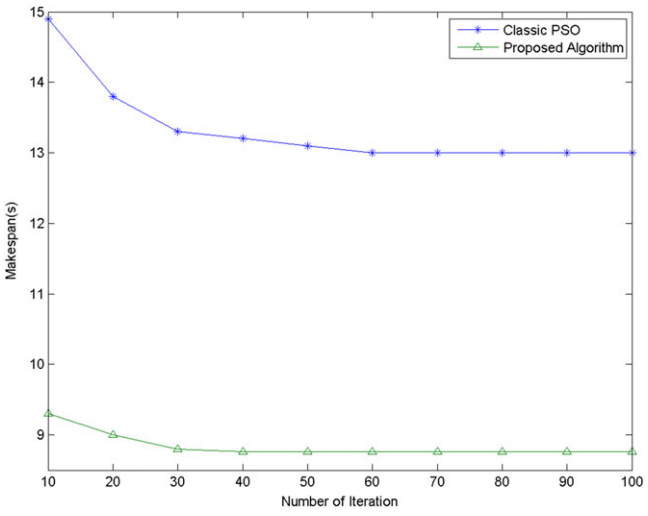

FIGURE 7 The convergence rate of the classic PSO and our proposed method

TABLE 9 Hosts' technical details

CPU	Number of Cores	Processing Speed, Mips	RAM, MB	Hard, MB	Bandwidth, Mbps
Core_2_Extreme_X6800	2	27 079	20 480	1 048 576	102 400
Core_i7_Extreme_Edition_3960X	6	177 730	1024	1 048 576	102 400
Core_i7_Extreme_Edition_980X	6	147 600	20 480	1 048 576	102 400
Core_i7_875K	4	92 100	20 480	1 048 576	102 400

TABLE 10 VMs' technical details

CPU	Number of Cores	Processing Speed, Mips	RAM, MB	Hard, MB	Bandwidth, Mbps
Core_i4_Extreme_Edition	1	9726	512	10 240	1024

Figure 8 shows the makespan comparison for the four algorithms. With a fixed number of VMs (40), the number of jobs is increased from 300 to 2500. Our proposed method works efficiently under real workloads and outperforms the other three algorithms in terms of makespan duration. Table 11 presents the makespan value for each algorithm with different numbers of tasks using the NASA log.

Figure 9 shows the values of makespan, resource utilization, and response time for up to 80 tasks and the makespan for NASA log up to 2500 tasks, which are obtained from application of the proposed method and others. As the figure shows, the proposed task scheduling method results in a higher resource utilization and lower makespan compared with the other algorithms. Letters A to E indicate the significant differences based on the LSD (i.e., $\alpha = 0.05$).

4.3 | Results analysis

Our proposed method is based on an improvement of the PSO meta-heuristic algorithm with the use of a load-balancing algorithm inspired by the honeybee algorithm. We compared and analyzed the current proposed method with (1) our previous work,[41] which used an improvement of the PSO algorithm through the linear descending method to calculate inertia weight; (2) the improved PSO in Zhan and Huo[14]; (3) the honeybee load-balancing algorithms in Krishna[3]; and the load-balancing method in Mittal and Katal,[4] for response time, makespan, and resource utilization.

Our proposed load-balancing technique was inspired by the load-balancing algorithm in Krishna[3] based on which requests move from overloaded VMs to underloaded ones, in the case of unbalanced loads. However, the difference between our work and the previous work is that the selection of an underloaded VM to moving requests to is based on (1) the request resource usage patterns; (2) the VM's available resource pattern; and (3) the VM's resource utilization pattern. Therefore, in our proposed method, the selected VM has the most compatibility with the user's request; while the load-balancing method in Krishna[3] only considers the resources of VMs. Improvement of the load-balancing algorithm in our proposed method led to proper distribution of workload on VMs. This causes the reduction of makespan and the increase of resource utilization, simultaneously. Moreover, the suitable use of a meta-heuristic algorithm and the proposed method tried to reduce the amount of makespan. Thus, as Figure 4 shows, the proposed method has a lower makespan compared with the honeybee load-balancing algorithm[3] and classical PSO algorithm.

Another contrast is to Zhan and Huo[14]; in this study,[14] the PSO algorithm has been combined with a one-directional SA meta-heuristic algorithm. Even though the use of the SA algorithm in creation of the initial population can act better than previous studies,[3,41] only the makespan has been considered in the objective function, and it is less efficient when compared with our proposed method. Our proposed method could increase resource utilization and reduce the amount of makespan by selecting resources utilization pattern in the objective function with the use of the load-balancing technique to achieve this goal.

To show the effectiveness of the current proposed method, we compared it with the best method that was selected to calculate inertia weight for the PSO algorithm in our previous method[41]; moreover, through the load-balancing method, we improved the creation of the initial population instead of a random population.

In Mittal and Katal,[4] a combination of RASA and Min-Min algorithms for scheduling were used, but the makespan reduction was less than the current proposed algorithm; furthermore, in this study,[4] the computational overhead increases by increasing the number of requests due to the voracious nature of the algorithm and the lack of consideration of compatibility of requests with VMs on which the requests are to be run.

In the case of response time, since the proposed method uses the load-balancing technique for suitable distribution of work on the VMs, the response time to some of requests may initially be higher than the time when there were less requests due to workload density, but it has a better overall response time than other algorithms due to reduction of makespan. The makespan reduction leads to increase of resource utilization. This was partly due to the objective function used in our proposed method. This causes better resource efficiency compared with the load-balancing

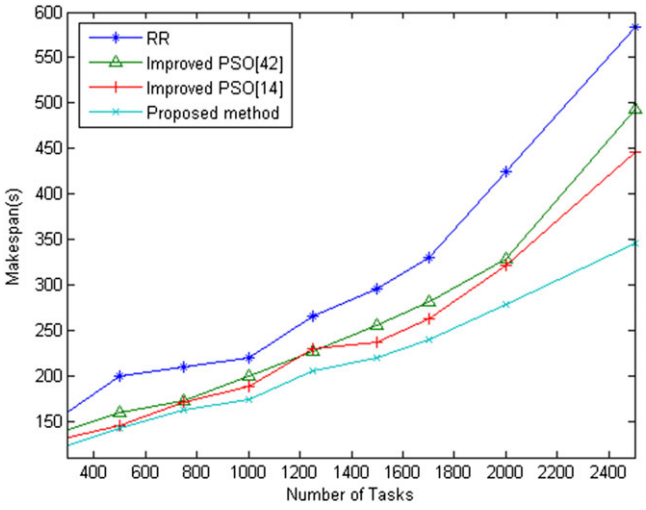

FIGURE 8 The makespan values obtained from real workload using the 3 methods and ours

TABLE 11 Makespan values for different methods

		Number of Tasks								
		300	**500**	**750**	**1000**	**1250**	**1500**	**1700**	**2000**	**2500**
Simple RR	Makespan NASA	161 ± 0^{t}	199 ± 0^{q}	210 ± 0^{o}	219 ± 0^{n}	265 ± 0^{j}	296 ± 0^{h}	330 ± 0^{f}	425 ± 0^{d}	583 ± 0^{a}
Improved PSO[37]		138.5 ± 5.6^{u}	157.2 ± 2.7^{t}	177.5 ± 4.6^{s}	199.7 ± 1.70^{pq}	229.66 ± 2.55^{n}	259.66 ± 5.02^{k}	281.66 ± 4.51^{i}	328 ± 3.09^{f}	497.66 ± 5.03^{b}
Improved PSO[13]		130.9 ± 2.37^{v}	141.6 ± 3.2^{u}	170.33 ± 2.16^{s}	185.66 ± 2.7^{r}	223.66 ± 6.27^{o}	240.6 ± 5.16^{l}	266.66 ± 4.42^{j}	317.66 ± 4.45^{g}	445.66 ± 6.15^{c}
Proposed method		123.1 ± 2.28^{w}	145.5 ± 4.17^{u}	159.3 ± 4.13^{t}	174.4 ± 4.11^{s}	205.33 ± 2.51^{R}	217.33 ± 2.17^{n}	236.66 ± 4.2^{l}	279 ± 2.00^{j}	346.66 ± 10.2^{e}

Values correspond to mean ± standard error of three (n = 10) measurements. The superscript letters in each column indicate the significant difference (according to the least significant difference ie, $\alpha = 0.05$).

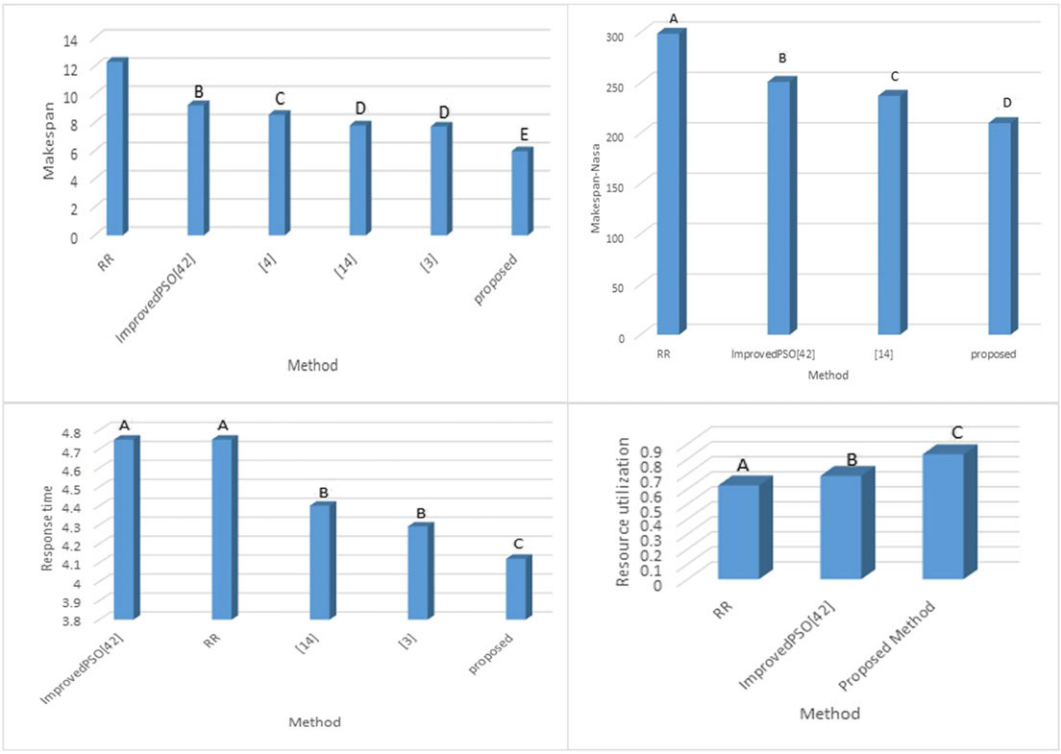

FIGURE 9 The parameters' values obtained from the proposed method and other algorithms

algorithm in Krishna.[3] According to our simulation results, in the current proposed method, the resource utilization is better compared with the improved PSO algorithm[41] due to balancing the load on VMs and the effect of resource utilization in selection of the population of each generation according to the objective function in Equation 11.

5 | CONCLUSION AND FUTURE WORK

Task scheduling plays a key role for cloud computing service providers as it must bring mutual benefits for both cloud providers and their clients. In this paper, we have presented a new PSO-inspired task scheduling method capable of balancing the load among VMs. Task-resource assignments are analogous to particle's position vectors. Initially, position vectors are generated randomly, and then at each iteration, position vectors are improved using a load-balancing technique. Benefiting from load-balancing technique, it is capable of decreasing makespan. In addition, it is qualified of increasing resource utilization and performance taking advantageous from proper fitness function. Our method evidently reduces makespan by 33% and increases the system utilization by 22% while it converges to the global minima in half-time comparing with PSO task scheduling method.[41] It also outperforms base round robin and load-balancing techniques. Our method is generic and scalable, as it can be deployed in data centers with any number of tasks and resources by increasing task-resource array dimension. Furthermore, our method is applicable for any cloud environments with independent and non-preemptive tasks. In the future, we plan to expand our method for workflow applications and taking other QoS criteria like fault tolerance capability and cost reduction into account.

ACKNOWLEDGMENT

Authors would like to thank University of Kashan for supporting this study under grant no. 577242.

ORCID

Seyed Morteza Babamir http://orcid.org/0000-0002-1645-4002

REFERENCES

1. Gomathi B, Krishnasamy K. Task scheduling algorithm based on hybrid particle swarm optimization in cloud computing environment. *J Theor Appl Inf Technol*. 2013;55(1):33–38.
2. Eberhart RC, Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995:39-43.
3. Krishna PV. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Appl Soft Comput*. 2013;13:2292-2303.

4. Mittal S, Katal A. An optimized task scheduling algorithm in cloud computing. In: 2016 IEEE 6th International Conference on Advanced Computing (IACC), 2016:197-202.
5. Singh P, Dutta M, Aggarwal N. *A Review of Task Scheduling Based on Meta-Heuristics Approach in Cloud Computing*. Knowledge and Information Systems, Springer; 2017 published online. https://doi.org/10.1007/s10115-017-1044-2
6. Zhao C, Zhang S, Liu Q, Xie J, Hu J. Independent tasks scheduling based on genetic algorithm in cloud computing. In: 2009 5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009:1-4.
7. Wang L, Ai L. Task scheduling policy based on ant colony optimization in cloud computing environment. In: Zhang Z, Zhang R, Zhang J, eds. *LISS 2012: Proceedings of 2nd International Conference on Logistics, Informatics and Service Science*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013:953-957.
8. Chaharsooghi SK, Meimand Kermani AH. An effective ant colony optimization algorithm (ACO) for multi-objective resource allocation problem (MORAP). *Appl Math Comput*. 6/15/ 2008;200:167-177.
9. Tsai C C, Rodrigues J. Metaheuristic scheduling for cloud: a survey. *IEEE Syst J*. 2014;8(1):279-291.
10. Kalraa M, Singh S. A review of metaheuristic scheduling techniques in cloud computing. *Egyptian Informatics Journal*. 2015;16(3):275-295.
11. Liu L, Zhang M, Lin Y, Qin L. A survey on workflow management and scheduling in cloud computing. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2014;837-846.
12. Mandal T, Acharyya S. Optimal task scheduling in cloud computing environment: meta heuristic approaches. In: 2015 2nd International Conference on Electrical Information and Communication Technologies (EICT), 2015:24-28.
13. Jones K.O, Boizant g. Comparison of firefly algorithm optimisation, particle swarm optimisation and differential evolution, presented at the Proceedings of the 12th International Conference on Computer Systems and Technologies, Vienna, Austria, 2011.
14. Zhan S, Huo H. Improved PSO-based task scheduling algorithm in cloud computing. *J Inform Comput Sci*. 2012;9:3821-3829.
15. Pandey S, Wu L, Guru SM, Buyya R. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: 2010 24th IEEE International Conference on Advanced Information Networking and Applications, 2010:400-407.
16. Zhang L, Chen Y, Sun R, Jing S, Yang B. A task scheduling algorithm based on PSO for grid computing. *International Journal of Computational Intelligence Research*. 2008;4:37-43.
17. Guo L, Zhao S, Shen S, Jiang C. Task scheduling optimization in cloud computing based on heuristic algorithm. *J Networks*. 2012;7:547-553.
18. Kaur G, Sharma ES. Optimized utilization of resources using improved particle swarm optimization based task scheduling algorithms in cloud computing. *Int J Emerging Technol Adv Eng*. June 2014;4:
19. Abdi S, Motamedi SA, Sharifian Saeed. Task scheduling using modified PSO algorithm in cloud computing environment, presented at the International Conference on Machine Learning, Electrical and Mechanical Engineering (ICMLEME'2014), Dubai (UAE), 2014.
20. Bilgaiyan S, Sagnika S, Das M. A multi-objective cat swarm optimization algorithm for workflow scheduling in cloud computing environment. In: Intelligent Computing, Communication and Devices: Proceedings of ICCD 2014, Volume 1, 2015, pp. 73-84.
21. Udomkasemsub O, Li X, Achalakul T. A multiple-objective workflow scheduling framework for cloud data analytics. In: 2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE), 2012:391-398.
22. Wu Z, Ni Z, Gu L, Liu X. A revised discrete particle swarm optimization for cloud workflow scheduling. In: 2010 International Conference on Computational Intelligence and Security, 2010:184-188.
23. Yassa S, Chelouah R, Kadima H, Granado B. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *Sci World J*. 2013;2013:13.
24. Khalili A, Babamir SM. Optimal scheduling workflows in cloud computing environment using Pareto-based Grey wolf optimizer. *Concurr Comp Pract E*, 2017;29(11):e4044-n/a, Art. no. e4044.
25. Nuaimi KA, Mohamed N, Nuaimi MA, Al-Jaroodi J. A survey of load balancing in cloud computing: challenges and algorithms. In: 2012 Second Symposium on Network Cloud Computing and Applications, 2012:137-142.
26. Nakai A, Madeira E, Buzato LE. On the use of resource reservation for web services load balancing. *J Network Syst Management*. 2015;23(3)502-538.
27. Daraghmi EY, Yuan S-M. A small world based overlay network for improving dynamic load-balancing. *J Syst Softw*. 2015;107:187-203.
28. Lu Y, Xie Q, Kliot G, Geller A, Larus JR, Greenberg A. Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. *Perform Eval*. 2011;68(11):1056-1071.
29. Ramezani F, Lu J, Hussain FK. Task-based system load balancing in cloud computing using particle swarm optimization. *Int J Parallel Program*. 2014;42(5):739-754.
30. Kokilavani T, George Amalarethinam DI. Load balanced min-min algorithm for static meta-task scheduling in grid computing. *Int J Comput Appl*. April 2011;20(2):0975-8887.
31. George Amalarethinam VK. Max-min average algorithm for SchedulingTasks in grid computing systems. *International Journal of Computer Science and Information Technologies*. 2012;3(2):3659-3663.
32. Elzeki LM, Reshad MZ, Elsoud MA. Improved max-min algorithm in cloud computing. *Int J Comput Appl*. July 2012;50(12):0975-8887.
33. Bhoi U, Ramanuj Purvi N. Enhanced max-min task scheduling algorithm in cloud computing. *International Journal of Application or Innovation in Engineering & Management*. April 2013;2(4). ISSN2319-4847.
34. Parsa S, Entezari-Maleki R. RASA: a new grid task scheduling algorithm. *International Journal of Digital Content Technology and its Applications*. December 2009;3(4):91-99.
35. Johnson BR, Nieh JC. Modeling the adaptive role of negative signaling in honey bee intraspecific competition. *J Insect Behav*. 23(6):459-471.
36. Nanduri R, Maheshwari N, Reddyraja A, Varma V. Job aware scheduling algorithm for MapReduce framework. In: 2011 IEEE Third International Conference on Cloud Computing Technology and Science, 2011:724-729.
37. Manupati VK, Thakkar JJ, Wong KY, Tiwari MK. Near optimal process plan selection for multiple jobs in networked based manufacturing using multi-objective evolutionary algorithms. *Comput Ind Eng*. 2013;66(1):63-76.

38. Zuo L, Shu L, Dong S, Zhu C, Hara T. A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. *IEEE Access*. 2015;3:2687-2699.

39. Mohamad MS, Omatu S, Deris S, Yoshioka, M. Particle swarm optimization with a modified sigmoid function for gene selection from gene expression data. *Artif Life Robot*. 2010;15(1):21-24.

40. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exp*. 2011;41:23-50.

41. Khalili A, Babamir SM. Makespan improvement of PSO-based dynamic scheduling in cloud environment, In: Proceedings of the 23rd Iranian Conference on Electrical Engineering (ICEE), IEEEComputer Society, 2015:613-618.

42. Fang Y, Wang F, Ge J. A task scheduling algorithm based on load balancing in cloud computing, presented at the Proceedings of the 2010 international conference on Web information systems and mining, Sanya, China, 2010.

43. Feitelson DG, Nitzberg B. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In: Job Scheduling Strategies for Parallel Processing, 1995:337-360.

How to cite this article: Ebadifard F, Babamir SM. A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment. *Concurrency Computat: Pract Exper*. 2017;e4368. https://doi.org/10.1002/cpe.4368