Journal of Computers Vol. 31 No. 2, 2020, pp. 180-196
doi:10.3966/199115992020043102016

An Efficient Task Scheduling Algorithm Based on Particle Swarm Optimization with Self-Learning Strategy and Neighbor Heuristic Mechanism on the Cloud

Lili Fan[1], Minggang Dong[1,2*], Chao Jing[1,2]

[1] College of Information Science and Engineering, Guilin University of Technology, Guilin, 541004, China
Fanlily913@qq.com, d2015mg@qq.com, jingchao@glut.edu.cn

[2] Guangxi Key Laboratory of Embedded Technology and Intelligent System, Guilin University of Technology, Guilin, 541004, China

Received 22 July 2018; Revised 12 December 2018; Accepted 9 February 2019

Abstract. Task scheduling plays an important role for improving the efficiency of cloud. However, due to the features and complex scenarios of the cloud, traditional scheduling approaches face with three challenges: robust model, local optima and slow convergence. Therefore, in this paper, we firstly established a robust cloud task scheduling model, which takes heterogeneity, deadline, overheads of transmission and the cost into account. Then, we proposed a particle swarm optimization scheduling algorithm with the self-learning strategy and the neighbor heuristic mechanism. The self-learning strategy is adopted to improve the diversity of population and the neighbor heuristic mechanism can accelerate the convergence speed. In addition, a greedy policy was designed and applied to quickly improve the quality of the initial solutions. In this way, the proposed algorithm has a fast speed of convergence and can avoid trapping into the local optimum. Lastly, we conducted simulations on CloudSim platform. Both on small-scale and large-scale scheduling problems, the proposed scheduling approach outperforms other well-known representative scheduling algorithms in terms of makespan, users' expense and waiting time.

Keywords: cloud computing, neighbor heuristic, particle swarm optimization, self-learning, task scheduling

1 Introduction

Cloud computing is a recent computing paradigm that utilizes remote cloud resources to achieve a high-performance computation. It provides infrastructure platform and software as different on-demand services [1]. Since the cloud must process numerous tasks required by various users, and the cloud resources (computing nodes and bandwidth) are severely limited. It is crucial to properly utilize these resources to improve the efficiency of cloud systems. The task scheduling technique is one of the most effective approaches to address such problem. But due to the features and complex scenarios on the cloud, current works have limitations on optimizing the efficiency of the cloud [2-3]. Therefore, it is crucial to design and implement task scheduling with high efficiency, which is used for meeting the constraints based on the feature cloud systems.

There are plenty of works are focusing on using the techniques of task scheduling, which is applied to improve the performance and energy efficiency in the cloud. The work in [4] has been proposed to improve Quality of Service (QoS) to satisfy user's needs. To prevent huge energy consumption and overload, the authors in [5] have presented an efficient approach for resource allocation based on the skewness measure, which measures the irregularity in the practice of resources. The work in [6] has employed the distribution and scalability characteristics in the cloud to develop the high efficiency task

[*] Corresponding Author

scheduling. Meanwhile, due to the importance of resource allocation, the works in [7-8] have been proposed to improve the efficiency of resource allocation. And the work of [9] is dedicated to minimizing unreasonable task allocation during the stages of hosts to virtual machines and tasks to virtual machines. The work in [10-12] are paying considerable attention to devise the task scheduling algorithm to optimize the expense cost on the cloud. Also, the researchers in [13-14] have proposed to task scheduling algorithm to find a tradeoff between time and expense cost. Those methods have addressed the problem of improving efficiency in the cloud, however, as the increasing scale of tasks, the methods become inefficient with a longer time consumption.

In comparison, Evolutionary Algorithm (EA) can find a better solution within a reasonable time consumption. Due to the advantages of EA, it has been adopted into approaches of task scheduling that optimizes the efficiency of cloud systems. Genetic algorithm (GA) [15-16] and modified genetic algorithm [14, 17], ant colony optimization (ACO) algorithm [18], particle swarm optimization (PSO) algorithm [19], a simplified particle swarm optimization (SPSO) algorithm [20] and the improved linearly decreasing weight-particle swarm optimization (LDW-PSO) algorithm [21] are the well-known approaches of task scheduling in EA. Among these algorithms, the PSO [22] has a better performance than that of GA [23], [24-26]. Moreover, task scheduling problem is a typical combinatorial optimization problem, and PSO algorithm is widely used in this area. The improved PSO in [27-28] has been proposed to reduce the makespan and the expense cost by users. The works in [29] is to balance the system workload while shortening the task completion time. They have been used for optimizing tasks scheduling with the objective of shortening the total tasks' execution time in distributed systems. However, for the situation of the cloud, the resources are heterogeneous and tasks with various QoS, so we cannot directly use the EA algorithms to improve the task scheduling efficiency in the cloud.

Overall, above proposed algorithms have improved the performance of task scheduling to some extent. However, because some important factors have been neglected, the task scheduling model become infeasible. In the actual cloud system, task scheduling model is a significant factor to improve the system performance. There are several issues need to be considered in the design of the task scheduling model. First, the heterogeneity of cloud resources (e.g., servers) contain various capacity, so that generating different number of virtual machines (VMs). Then, according to the various users' demands, the task must be completed within the deadline constraints. Meanwhile, because of the limitation of the bandwidth, the high overheads of task transmission cannot be ignored. Last, when users are renting the virtual machines and transferring data, the total users' expense must be accounted without missing the tasks deadline constraint. In this paper, we proposed a complex task scheduling model to improve the performance. And we consider the heterogeneity of cloud resources and tasks, total task completion time and the user's expense in the design of the task scheduling model. Moreover, task scheduling is a NP-hard problem, general algorithms are not efficient in the cloud system. Many traditional approaches based on the cloud computing environment are easily fall into a local optimum and slow convergence. In this paper, we proposed a novel PSO algorithm with the self-learning strategy and the neighbor heuristic mechanism (SLNPSO) to reduce the task completion time and users' expense, simultaneously.

The main contributions of this paper can be summarized as follows:

- First, we setup the model of the task scheduling for complex scenarios on the cloud systems. In this model, heterogeneity of cloud resource, task scheduling with the deadline, overheads of the task transmission and the total users' expense are taken into account. Thus the proposed model is robust for application in real life.
- Second, we have proposed an efficient task scheduling algorithm based on the PSO with a self-learning strategy and neighbor heuristic mechanism. The self-learning strategy (SLS) is utilized to improve the diversity of population and the neighbor heuristic mechanism (NHM) can accelerate the convergence speed. Thus the proposed algorithm has advantages of ability of global search and fast convergence. In addition, a greedy policy (GP) has been introduced to the proposed algorithm that improves the quality of the initial solution.
- Last, we have conducted the experiment on Cloudsim [30] platform to verify the performance and feasibility of the proposed algorithm. The results have demonstrated that the proposed algorithm outperforms the state-of-the-art algorithms [14, 20, 22, 29] on both small-scale and large-scale tasksets.

The remainder of this paper is organized as follows. Section 2 presents the scheduling model and the problem formulation. The design details of our proposed SLNPSO algorithm in Section 3. Section 4 conducts experiments to evaluate the performance of our algorithm. Section 5 concludes the paper with the summary and future work.

2 Problem Formulation

In this section, we are going to give the problem formulation includes: the system model, cloud task scheduling and problem description.

2.1 System Model

For the scheduler in the cloud system, it normally experiences two stages. Initially, the tasks are scheduled to the virtual machines. The second stage is from virtual machines to physical hosts. In this paper, we are focusing on the task scheduling algorithm from tasks to virtual resources. The cloud system extracts heterogeneous physical resources into resource pools through virtualization technology. Different users submit the tasks to the system, and tasks are buffered in the waiting queue for later assigning to the proper virtual machines visa the scheduler. The system model of task scheduling has been shown in Fig. 1.

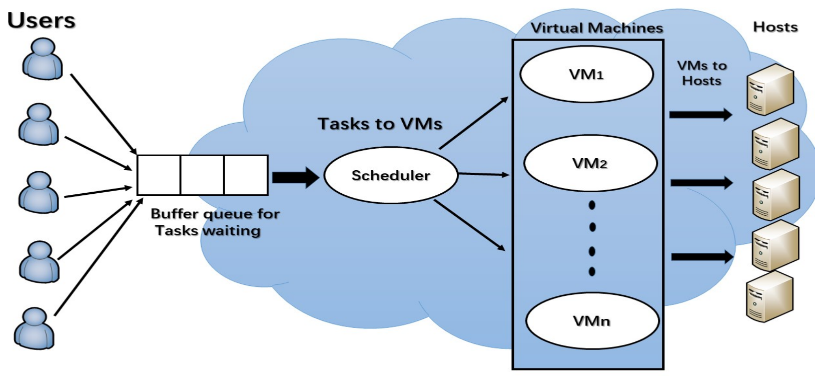

Fig. 1. Illustration of the system model for task scheduling on the Cloud

As it can be seen in Fig. 1, we can see that the basic mission of the scheduler is to assign *n* tasks to *m* heterogeneous virtual resources, and these tasks have to meet the objective of minimizing the total completion time. The objective of our work is not only reducing the completion time, but also account for users' expense, overhead of task transmission and different QoS submitted by users.

2.2 Task Scheduling Model in Cloud

In this subsection, we will give the detail formulation of the task scheduling model in basic and complex situations.

Basic scheduling model. A basic scheduling problem can be defined as follows: to find an optimal solution, a set of independent user tasks $T=\{T_1,T_2,\ldots,T_n\}$ is given, the goal is to minimize the completion time while mapping those tasks on a set of heterogeneous virtual machines $VMs=\{VM_1,VM_2,\ldots,VM_m\}$. The makespan is a maximum completion time *(CT)* on each virtual machine after all tasks finished. The task execution time (*ET*) on each virtual machine (VM) can be calculated by Eq (1), so the makspan (*MK*) can be derived from the maximum completion time of all tasks on each *VM*, which can be gained by Eq (2),

$$ET_{ij}=\frac{L\_Task_i}{VMips_j * Penum_j}. \tag{1}$$

$$MK=\max\{CT_j\},\quad j\in[1,m]. \tag{2}$$

where Eq (1) is *j* task execution time on VM_j, L_Task_i is the data size of *i* task, $VMips_j$ denotes the processing speed of virtual machine *j* and $Penum_j$ is the number of processors used by virtual machine *j*. Eq (2) is the makespan, CT_j represents the completion time on *j VM* after all tasks finished.

Complex scheduling model. There are n number of independent tasks submitted by users $T=\{T_1,T_2,...,T_n\}$. Each task contains L_Task_i size of data. And $VM=\{VM_1,VM_2,...,VM_m\}$ is given set of heterogeneous virtual machines. The process speed of VM is dependent on two factors: $VMips_j$ processing speed and $Penum_j$ assigned processors.

Different from the basic scheduling model, we take the average bandwidth (BW_{ij} for task i transfers to VM_j via link L_{ij}) into account because of severely limited bandwidth and large size of each task transferring in the cloud. Also, we suppose that the expense cost on each VM is proportional to the process speed, meanwhile, there is a transmission expense which is the transmission overhead for transferring task to VM. Therefore, the users' expense is the sum of task processing cost $Cproc_{ij}$ and transmission overhead $Ctrans_{ij}$ as Eq (3),

$$TUE=\sum_{1\le i\le n}^{n}\sum_{1\le j\le m}^{m} Cproc_{ij}+Ctrans_{ij}. \quad \textbf{(3)}$$

We assume that $Cproc_{ij}$ is proportional to the c_j and ET_{ij}, where c_j is the expense cost for processing cost per unit time by VM_j, it linearly increases with the processing speed on that VM_j. ET_{ij} is the execution time of task on VM_j. By doing so, the expense of task i processing cost on VM_j can be calculated as follows in Eq (4),

$$Cproc_{ij}=c_j*ET_{ij}. \quad \textbf{(4)}$$

In Eq (5), we suppose that $Ctrans_{ij}$ is associated with $\bar{c}_{ij}$ transfer cost per unit time and $trans_{ij}$ transferring time used on the link L_{ij}, where $\bar{c}_{ij}$ is proportional to the average bandwidth BW_{ij}, $trans_{ij}$ is the size of task L_Task_i divided by average bandwidth BW_{ij} in Eq (6),

$$Ctrans_{ij}=\bar{c}_{ij}*trans_{ij}. \quad \textbf{(5)}$$

$$trans_{ij}=\frac{L\_Task_i}{BW_{ij}} \quad \textbf{(6)}$$

Last, in some situation, there is a great number of tasks submitted to the cloud. The tasks have to be buffered in the waiting queue for later processing, so we suppose that the average waiting time follows Poisson distribution [31]. $Wait_{ij}$ denotes the average waiting time for task j that waits for execution on VM_j. Therefore, the task j completion time on VM_j. WET_{ij} is the sum of execution time ET_{ij} and $Wait_{ij}$, Eq (7),

$$WET_{ij}=\mathrm{W}ait_{ij}+ET_{ij}. \quad \textbf{(7)}$$

2.3 Problem Description

As we have given the system model and task scheduling model above, for the basic task scheduling model, there is n number of independent tasks and m number of virtual machines, the objective is to minimize the makespan while mapping the tasks to the virtual machines. Thus, the problem for the basic task scheduling (BTSC) on the scenario can be defined as follows,

$$objective: Minimize\, MK=\{\max(CT_1,CT_2,...,CT_m)\}. \quad \textbf{(8)}$$

$$s.t., CT_j=\sum_{i=1}^{n} ET_{ij}\cdot x_{ij}. \quad \textbf{(8.a)}$$

$$x_{ij}=0\, or\, 1. \quad \textbf{(8.b)}$$

$$\sum_{j=1}^{m} x_{ij}=1,\ i\in[1,n]. \quad \textbf{(8.c)}$$

where constraint (8.a) is the completion time on certain VM_j. (8.b) denotes the task i whether maps to the VM_j. (8.c) represents the task j must be assigned to one of the virtual machines VM_j.

Moreover, we are going to describe the problem of complex task scheduling (CTSC) on the scenario. Since the users' expense is a key factor on the cloud, the objective on CTSC becomes minimizing the user's expense and makespan under the task deadline constraint D and users' expense budget B, while mapping n tasks onto m virtual machines. CTSC can be defined as,

$$\begin{aligned} &objective: \\ &Minimize\ \Theta = w_1 \cdot \overline{MK} + w_2 \cdot TUE \quad . \end{aligned} \tag{9}$$

$$s.t.,$$

$$\overline{MK} = \{\max(\overline{CT}_1, \overline{CT}_2, ..., \overline{CT}_m)\} \tag{9.a}$$

$$\overline{CT}_j = \sum_{i=1}^{n} WET_{ij} \cdot y_{ij}. \tag{9.b}$$

$$y_{ij} = 0\, or\, 1. \tag{9.c}$$

$$\sum_{j=1}^{m} y_{ij} = 1,\ i \in [1, n]. \tag{9.d}$$

$$\overline{MK} \le D. \tag{9.e}$$

$$TUE \le B. \tag{9.f}$$

$$w_1 + w_2 = 1,\ w_1, w_2 \in [0,1]. \tag{9.g}$$

where the makspan $\overline{MK}$ is gained on each VM after all tasks completed. $\overline{CT}_j$ is the VM_j completion time while mapping a set of tasks. (9.c) denotes the task i whether maps to the VM_j. (9.d) represents the task j must be assigned to one of the virtual machines VM_j. Constraint (9.e) and (9.f) stand for the deadline constraint and expense budget, respectively. Constraint (9.g) is the weight parameters for the relative importance of makespan and users' expense.

3 The Proposed SLNPSO Algorithm

3.1 Overview

For PSO algorithm in the past literatures [29, 32], it has shown great improvement in efficiency for task scheduling in the cloud. Our algorithm is an improved version for the task scheduling based on PSO algorithm. In this paper, we proposed a particle swarm optimization scheduling algorithm with the SLS and the NHM. The proposed algorithm has advantages of fastening the speed of convergence and search. The SLS is adopted to improve the diversity of population, and the NHM can accelerate the convergence speed. Moreover, a GP was designed and applied to quickly improve the quality of the initial solutions.

In this paper, our proposed algorithm is compared with four state-of-the-art algorithms. Therefore, we have compared various scheduling methods to highlight our algorithm difference in Table 1. In Table 1, simple particle position (SPP) represents the policy of simplifying particle position update, fastening position updating (FPU) presents the strategy of fastening position updating, space shared (SS) is the policy of space shared. The SLS and NHM and are applied to SLNPSO algorithm. The comparison includes initialization methods, the strategy of jumping out of local optimum (JLO), the scheduling objectives, the strategy of optimizing global optimum (OGO) and scheduling models. References are the traditional PSO algorithm [22], SPSO algorithm [20], the greedy particle swarm optimization (G&PSO)

algorithm [29], the space-shared genetic algorithm (SSGA) [14] and SLNPSO algorithm.

Table 1. Comparison of various scheduling methods

Algorithms	Initialization	JLO	Scheduling Objectives			OGO	Scheduling Model	
			Expense	Makespan	System Load Balance		Basic	Complex
PSO	random	—	—	√	—	—	√	—
SPSO	random	SPP	—	√	—	—	√	—
G&PSO	GP	FPU	—	√	√	—	√	—
SSGA	random	—	√	√	—	SS	—	√
SLNPSO	GP	NHM	√	√	√	SLS	√	√

3.2 Greedy Policy

Population initialization based the GP is introduced to improve the quality of the initial solution. For the traditional PSO algorithm applied to the task scheduling, the initialization of particles is random. In this paper, a GP [29] is introduced to improve the quality of the initial solution. It quickly finds the initial solution G_{ov} and the expected total completion time G_{ct}. Also, the global optimum value (*gbest*) is initialize by G_{ct}.

The GP is effective in the basic scheduling model. However, due to heterogeneity and uncertainty of resources bring challenges to resource allocation in the cloud, which requires an efficient task scheduling method to adapt complex scheduling environment. So, the effectiveness of the GP is also to be verified in the complex scheduling model. Moreover, we are going to test the efficiency in the complex scheduling scenario.

In this paper, the fitness function of basic model and complex model are described as follow Eq (10) and Eq. (11), respectively. Where MK is from Eq. (2), makespan and expense (ME) is the sum of the weighted values of the makespan from Eq (3) and the expense from Eq (7).

$$FitMK = 1/_{MK}. \tag{10}$$

$$FitME = 1/_{ME}. \tag{11}$$

3.3 Neighbor Heuristic Mechanism

The standard PSO algorithm is slow convergence speed and easily trapped into the local optima because limitations of itself. A new neighbor heuristic mechanism in [33] is introduced to overcome these defects. The new velocity updating equation is not only influenced by the local optimum value (*pbest*) of particle and *gbest* of particle, but also directed by the better position in the neighborhood. The *i* particle's neighbor heuristic particle is denoted by m_i, shorten for *mparticle* in Eq. (12). The neighbor information of the current particle feeds back into the velocity updating equation, which is given by Eq. (13), the particle's position updating is calculated by Eq. (14). And the linearly varying inertia weight *w* was introduced to control the search speed [34] in Eq. (15).

$$m_{id}^{t} = \frac{\sum_{j=1}^{k} n_{jd}^{t}}{k}. \tag{12}$$

$$v_{id}^{t+1} = wv_{id}^{t} + c(r_1(p_{gd}^{t} - x_{id}^{t}) + r_2u_1(p_{id}^{t} - x_{id}^{t}) + r_3u_2(m_{id}^{t} - x_{id}^{t})). \tag{13}$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}. \tag{14}$$

$$w = w_{max} - (w_{max} - w_{min})\frac{t}{T}. \tag{15}$$

where w is inertia weight, T is the maximum number of iterations, t is the current iteration, w_{max} and w_{min} are the maximal weight and minimal weight. The number of *mparticles* is denoted by *k*. *c* is acceleration coefficient, other specific parameters refer the document [33]. The process of *mparticles* can be described by Function 1.

Function 1. process of *mparticles*

```
Input: position of the particle $P_i$, the size of population $N$
Output: new position $NP_i$
1.  function Mparticle()
2.  Calculate Fitness of particles according to Eq.(10) or Eq.(11);
3.    for $i$ =1 to $N$ do
4.      for $j$ =1 to $N$ do
5.         If (Fitness ($j$) > = Fitness ($i$)) then
6.           $P_{sum}$ + = $P_j$;
7.           sum++;
8.         end if
9.      end for
10.    $NP_i$ = $P_{sum}$ /sum;
11.   end for
12. end function
13. Return new position $NP_i$;
```

In Function 1. The input is the position of *i* particle denoted P_i, the size of population denoted *N*. The output is the new position of *i* particle denoted NP_i. The process is aimed to find the *i* particle's neighbor heuristic particles (lines 3-11), *i*, *j* represents *i* and *j* particle, P_{sum} is the sum of particle position and *sum* is the number of particles, Sum will be cumulated when the fitness value better than the *i* particle in the whole population.

3.4 Self-Learning Strategy

The GP is used to quickly improve the quality of the initial solution and the NHM is used to accelerate the convergence speed of the particle in this above section. However, the diversity of the population will be decreased in the process simultaneously. Therefore, to avoid particles attracting too fast toward the best particles, the SLS was proposed in this section.

The SLS is used to improve the diversity of population by self-learning strategy of *gbest* particle. Differ from other particles, the *gbest* has no exemplars to learn as well as is easily fall into local optimal. In this paper, we design the SLS considering the characteristics of the cloud computing task scheduling. It is proposed to help *gbest* push itself out to a potential better area and obtain the global optimum. If such area is found, the rest of the population will follow the leader to jump out of local optima and converge to the new area. The processing of *gbest* are expressed by Eq. (16), Eq. (17).

$$d = random(1, D). \tag{16}$$

$$p^d = p^d_{new}. \tag{17}$$

D is the dimension of particals, P^d is transformed randomly from [1, VM_{num}], VM_{num} is the number of VMs. The SLS randomly chooses one dimension from *gbest*'s historic best position by Eq. (16), which is denoted by P^d for the d dimension. As every dimension has the same probability to be chosen, the SLS operation can be regarded as the same probability to perform on every dimension. And SLS's pseudo code is shown in Function 2.

Function 2. self-learning process SLS

```
Input: global best position set gP
Output: new global best position set newP
1. function learnProcess()
2.    calculate $fit_1$ to Eq.(10) or Eq.(11)
3.      $p^d$ ,$d$ ← according to Eq.(16), Eq.(17);
4.      for $i$ = 1 to $K$ do
5.         for $j$ =1 to $VM_{num}$ do
6.            if ($j$! =$p^d$) then
7.            calculate $fit_2$ to Eq.(10) or Eq.(11)← new set;
8.               if ($fit_2$ > $fit_1$) then
9.               $p^d$ = $j$;
10.                calculate newP ← new set;
11.              end if
12.           end if
13.        end for
14.    end for
15. end function
16. Return new global best position set newP
```

In function 2, the input is a global best position set denoted *gP*, which is the set of virtual machines that tasks are mapped, the fitness function value is fit_1. The output is a new global best position collection denoted *newP*, which is the new set of virtual machines that tasks are mapped, the new fitness function value is fit_2. Where p^d is the virtual machine of d dimension. K is the maximum number of learning. The new global best set is generated (lines 4-14). Then, a new position set is obtained, and the new position will be accepted only when its fitness is better than the current *gP*.

3.5 SLNPSO

We have proposed an efficient task scheduling algorithm based on the PSO algorithm with a self-learning strategy and neighbor heuristic mechanism. The SLNPSO algorithm employs three strategies, namely SLS, NHM and GP. The self-learning strategy can improve the diversity of population, and the neighbor heuristic mechanism is applied to accelerate the convergence speed. The proposed algorithm has advantages of fastening the speed of convergence and search. Meanwhile, a greedy policy has been introduced to the proposed algorithm that improves the quality of the initial solution. The main steps of SLNPSO are described in Algorithm 1.

Algorithm 1. SLNPSO

```
Input: population size $N$ , max number of iteration $M$
Output: a best assign set of tasks $S$;
1.  Uniformly randomly initialize each particle position $X_i$ and
    velocity of particle $V_i$;
2.  Use greedy strategy initialize $pbest_i$ and $gbest$;
3.  While iterations < $M$ do
4.     for $i$ = 1 to $N$ do
5.       Calculate $m_i$ according to Eq.(12);
6.       Caculate the velocity of particle $V_i$ according to Eq.(13);
7.       Update the position of particle $X_i$ according to Eq.(14);
8.       Update $pbest_i$ and $gbest$;
9.     end for
.       /* globally best particle learning */
10.     for $k$ = 1 to $K$ do
11.     Calculate new globally best particle  position to
        Eq.(16),Eq.(17);
12.         if  Fitness(newP) > Fitness(gbest) then
```

```
13.          S = newP;
14.         end if
15.        k++;
16.     end for
17.   Iterations++;
18.  end while
19. Return a best assign set of tasks S;
```

In algorithm 1, the input is the population size denoted *N*, the maximum number of iteration denoted *M*. The output is a best assign set of tasks denoted *S*. Where X_i denotes the *i* particle position, V_i denotes the *i* particle velocity, *newP* is the set of new positions, *K* represents the maximum number of learning. The particle position and velocity are initialized by randomly, and $pbest_i$ and *gbest* are initialized by a greedy policy (lines1-2). The objective is obtaining a set of approximate solution in iterative process of particle swarm (lines 3-18). The update process of particles is calculated in (lines 4-9). The learning process of global best particle learning is showed in (lines 10-16). SLS improves the *gbest* in each iteration so that we can get the optimal combination.

3.6 Analysis of Complexity

The time complexity is often used to evaluate the effectiveness of the algorithm. So, we evaluate the time complexity of the algorithm 1. The algorithm is divided into two parts. In first part, particles are initialized (lines 1-2), the time complexity is the population size denoted $O(N)$. The second part is the iterative process of particles (lines 3-18), which include the particle update process and the particle learning process in each iterative. In the particle update process (lines 4-9), the time complexity is related to the size of population denoted $O(N)$. After all the particles are updated, the learning process begins (lines 10-16). The time complexity of the learning process is related to particle's learning opportunities denoted *K*. In the process, we need calculate the new global best particle *newP,* the number of virtual machines is denoted VM_{num}, time complexity of *newP* has to do with VM_{num}. So, the time complexity of learning process is $O(K*VM_{num})$. In entire iteration process, the time complexity is $O(M*K*VM_{num}) + O(M*N)$.

4 Performance Evaluation

In this section, we are going to detail the experimental setup, parameters and results for two task scheduling scenarios.

4.1 Experimental Setup

To validate the feasibility and performance of the SLNPSO algorithm in terms of scheduling ability in the cloud, we used the cloud computing simulation platform Cloudsim [30]. It is a framework developed by the GRIDS laboratory of university of Melbourne which enables seamless modelling, simulation and experimenting on designing cloud computing. It supports the modeling and simulations of large cloud computing node, and includes service brokers, resource provisioning, datacenters and allocation policies. In this paper, we extended Cloudsim-3.0.3 to implement cloud scheduling algorithm, by modifying the Datacenter Broker in a class method, implement SLNPSO algorithm in the application of cloud task scheduling. Eclipse 4.4.0 IDE was used for the implementation. The proposed experiment is performed on two different scheduling scales: (1) small-scale and large-scale in the basic and complex task scheduling scenarios, simultaneously.

In the basic task scheduling scenario, we set only one CPU for each virtual machine (each virtual machine can only handle one task at one time and each has different properties). The number of virtual machines is set 5. Task number is set {10, 20, 30, 40, 50} on small-scale taskset, and task number is set {100, 200, 300, 400, 500} on large-scale taskset. The other parameters setting refer the document [29] in the basic task scheduling scenario. We evaluate makespan and the system load balance in the basic task scheduling of the cloud.

In the complex task scheduling scenario, the number of virtual machines is set 5, task number is set {20, 40, 60, 80, 100} on small-scale taskset. The task number is set {200, 300, 500, 600, 700}, and the number of VMs is set 50 on large-scale taskset. We set the computation price ranging from [0.001, 0.01] per second and the transfer overhead price ranging from [0.1, 0.7] per Gigabyte (GB). The main parameter settings of Cloudsim is shown in Table 2, and main parameters of the algorithm is shown in Table 3. The other parameters setting refer the document [35]. In table 2, Million Instructions Per Second (MIPS) denotes the capacity of the virtual machine. In our experiment, the population size is set 100, the maximum number of iterations is 200, the maximum number of learning times(k) is 5, the inertial factor (w) is ranging from [0.4, 0.9].

Table 2. Parameters setting of Cloudsim

Entity Type	Parameters	Values
	The number of cloud tasks	[20,100], [200, 600]
Task	The length of tasks	[1000, 20,000]
	File size of tasks	[200, 400] MB
	Output file size	300MB
	The number of VMs	5, 50
Virtual Machine	vCPU capacity	[500, 2000] MIPS
	vRAM	[256, 2048] MB
	Bandwidth	[500, 1000] MB/s
	The number of hosts	6
Host	memory	4096 MB
	bandwidth	2660 MB/s

In this paper, we have done many experiments, we have seen that the makespan and the users' expense can be reduced simultaneously, when w_t is set 0.6 and w_c is set 0.4. We evaluate the fitness value, average makespan, users' expense and the average waiting time in the complex task scheduling scenario of the cloud.

4.2 Comparative Algorithm

The values in these experiments represent the average result with twenty independent runs on each taskset. The proposed algorithm is compared with the state-of-the-art algorithms, traditional PSO algorithm [22], it is one of the most important algorithms in evolutionary algorithms and often applied to task scheduling in the cloud. G&PSO algorithm [29] is proposed to reduce the makespan as well as improve the system balance by combining PSO algorithm and a greedy policy in the cloud. SPSO algorithm [20] is simplifying the processing of particle position update to minimize the makespan and improve the convergence rate. SSGA [14] is proposed to reduce the makespan and users' expense by combining GA and the space-share strategy. Above algorithms, they are compared with the proposed algorithm in the experiment.

Table 3. Parameters of the algorithm

Parameter	Values
Population size NP	100
Maximum number of iterations	200
Maximum number of learning times(k)	5
Inertial factor (w)	[0.4,0.9]
w_t, w_c	0.6, 0.4

4.3 Experimental Results

Comparision Results of the Basic Task Scheduling in Cloud. In this section, we are going to demonstration and analysis of experimental results in the basic task scheduling scenario. In the scenario, we only consider the makespan as the optimization goal. Since the load balance is a crucial metric for the cloud system, so we also observe the load balance degree while minimizing the makespan. The load balance degree can be defined as follows Eq (18),

$$Load_{level} = \frac{min_{1 \le i \le n} Load_{VMT_i}}{max_{1 \le j \le n} Load_{VMT_j}}. \quad \textbf{(18)}$$

where n denotes the number of tasks, $min_{1 \le i \le n} Load_{VMT_i}$ is the minimum time for all the virtual machines to complete all the tasks above, and $max_{1 \le j \le n} Load_{VMT_j}$ is the maximum time for all the virtual machines to complete all the tasks above. And $Load_{level}$ is greater than 0 and less than 1, the closer to 1, the better. When $Load_{level} = 0$ means that there are idle virtual machines, and $Load_{level} =1$ means that the system load balance is the best.

Small-scale. The result of the average makespan is depicted in Fig. 2. They are admitted to task scheduler, which utilizes the CPU gap to minimize idle time and improve throughput. The lowest makespan improvement is obtained from SLNPSO. On average, the average makespan of the SLNPSO algorithm is 9%, 8%, 3% lower than PSO, SPSO and G&PSO, respectively. The proposed SLNPSO algorithm can efficiently utilize the resources, which enables more tasks to complete in shorter time.

The load balance in systems is measured by the load balance degree, which is calculated by Eq (18). The load balance degree is depicted in Fig. 3. In this paper, the greater the load balance degree, more balanceable the system. We can see that the load becomes more balanceable with the increasing number of tasks. The most imp provement for the load balance is SLNPSO. To summarize, the load balance degree of the SLNPSO algorithm is 31%, 29%, 9% higher than SPSO, PSO and G&PSO, respectively. The proposed SLNPSO algorithm can efficiently improve the system load balance.

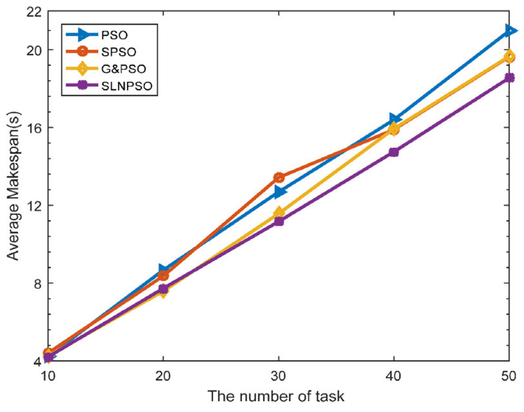

Fig. 2. Average Makespan with different tasks

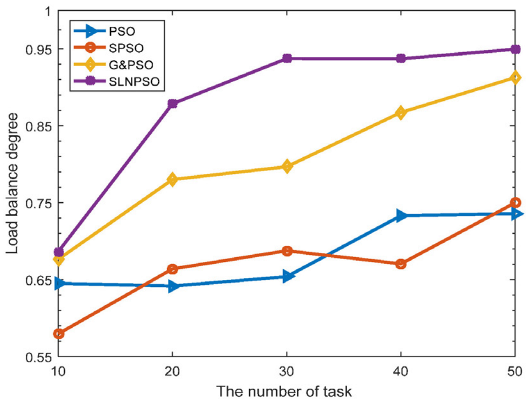

Fig. 3. System load balancing degree

Large-scale. Fig. 4 depicts the average makespan for the PSO, SPSO, G&PSO and SLNPSO algorithm. As we can see, the trend of the figure indicated that the average makesapn is increasing with a growing number of tasks. The minimal average makespan is obtained from SLNPSO algorithm and the maximal average makespan is obtained from PSO algorithm. In sum up, the average makepsan of the SLNPSO algorithm is 12%, 10%, 2% lower than PSO, SPSO, G&PSO, respectively. The SLNPSO algorithm outperforms other algorithms in terms of the average makespan, and it represents the most reliable scheduling algorithm.

For large-scale, the load balance degree is depicted in Fig. 5. We observe that the system balance degree is increasing with a growing number of tasks. On average, the load balance degree of the SLNPSO algorithm is 33%, 31%, 2% higher than SPSO, PSO, G&PSO, respectively. Our proposed SLNPSO algorithm can efficiently improve the system load balance and void the workload overload on virtual machine.

Compared with other algorithms, when scheduling large-or-small scale tasks, the proposed algorithm shows a stronger ability within the optimization process, and it has a better scheduling efficiency on reducing the makespan and improving the system load balance degree in the basic scheduling scenario.

Comparision results of the complex task scheduling in cloud. In this section, we are going to demonstration and analysis of experimental results in complex task scenario.

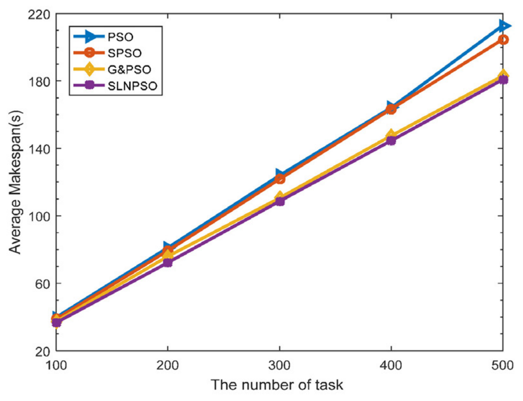

Fig. 4. Average Makespan with different tasks

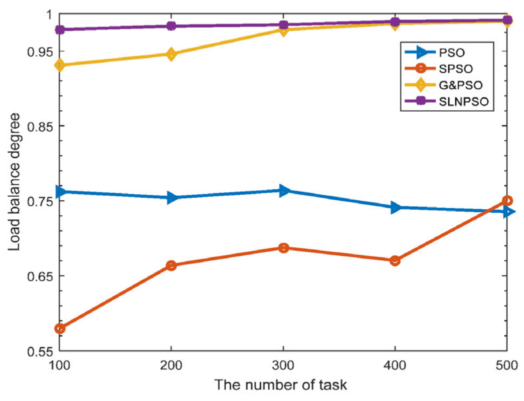

Fig. 5. System load balancing degree

Small-scale. In this paper, the fitness value is calculated by Eq (11), and the greater the fitness value, the result the better. The fitness value is shown in Fig. 6. We can see that the fitness value is decreasing with the number of tasks for all algorithms. The fitness value is determined by the makespan and users' expense, we observe that the proposed algorithm has the maximum average fitness in each taskset, so the proposed algorithm can reduce the makespan and users' expense, simultaneously. Results for large-scale taskset are shown from Fig. 6 to Fig. 9.

The average makespan has been shown in Fig. 7. The proposed algorithm has gained the minimum average makespan compared other algorithms. In sum up, the average makespan of the SLNPSO algorithm shows 66%, 45%, 29% and 15% over the SPSO, SSGA, PSO and G&PSO, respectively. We observe that SLNPSO algorithm is more efficient to reduce users' expense, and it shows the better scalability with the increasing number of tasks.

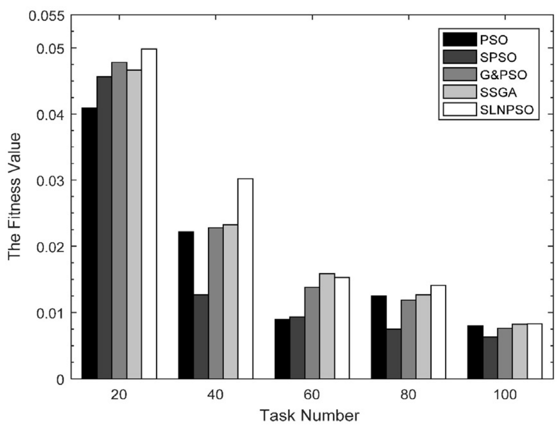

Fig. 6. Fitness value with different tasks

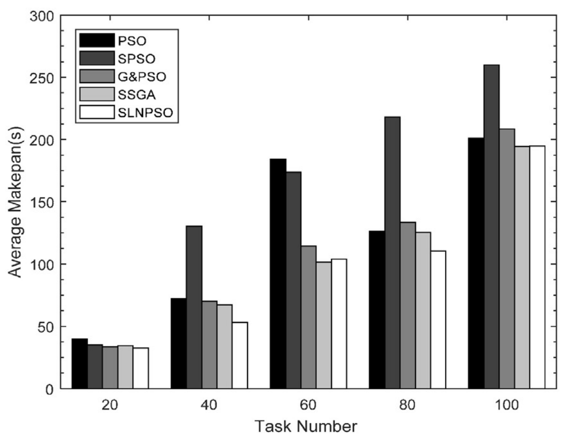

Fig. 7. Average Makespan with different tasks

The users' expense is evaluated with the different tasks in Fig. 8. As we can see, the minimal users' expense is obtained from SLNPSO. To summarize, the users' expense of the SLNPSO algorithm is 26%, 17%, 12%, 8% lower than SPSO, G&PSO, PSO and SSGA, respectively. SLNPSO algorithm outperforms other algorithms in terms of the users' expense.

The task average waiting time is depicted in Fig. 9. We observe that the SLNPSO algorithm outperforms other algorithms in terms of the average waiting time. In a word, SLNPSO is 24% over both the SPSO and SSGA, 14 %, 4% over the PSO, G&PSO, respectively.

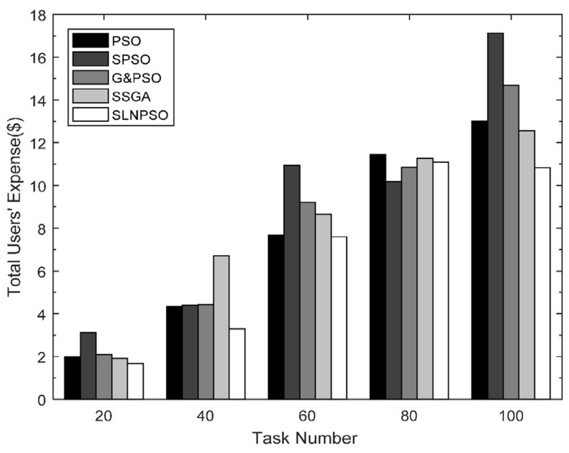

Fig. 8. Total users' expense with different tasks

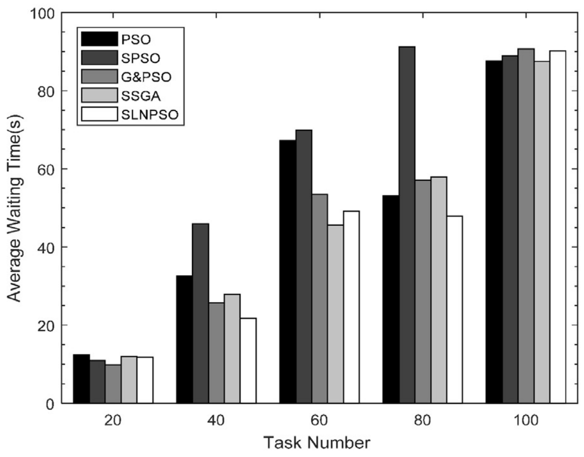

Fig. 9. Average waiting time with different tasks

Large-scale. In Table 4, we list the evaluation result of all the algorithms in large-scale experiment, the best result in bold face. Results for large-scale taskset are shown from Fig. 10 to Fig. 13.

Table 4. Evaluation result of algorithms

Evaluation metrics	Tasks	PSO	SPSO	G&PSO	SSGA	SLNPSO
Fitness	200	0.0184	0.0150	0.0275	0.0231	**0.0291**
	300	0.0126	0.0107	0.0192	0.0142	**0.0224**
	400	0.0098	0.0098	0.0142	0.0100	**0.0161**
	500	0.0074	0.0068	0.0132	0.0075	**0.0136**
	600	0.0062	0.0071	0.0108	0.0072	**0.0113**
Makespan	200	76.7486	94.6411	43.5127	57.0668	**41.7154**
	300	109.3291	133.1240	60.5572	96.0385	**53.0628**
	400	134.9671	139.1253	84.7094	133.9124	**73.6540**
	500	185.1845	203.4213	88.8206	178.9497	**82.2871**
	600	220.9361	189.0902	107.1317	184.1522	**100.8606**
Expense	200	24.2764	26.2023	25.7580	**22.9842**	23.2928
	300	36.8867	36.2321	39.6317	32.9008	**32.1015**
	400	54.3589	48.7912	49.2638	50.5143	**44.6342**
	500	64.5431	65.4863	56.2947	63.7998	**54.6566**
	600	76.3660	71.3066	71.4317	71.9718	**69.3025**
Waiting Time	200	16.8450	19.3073	**11.7958**	15.6398	13.9423
	300	26.7738	27.9749	20.3806	26.8268	**18.8224**
	400	33.5578	32.5457	30.8218	37.0671	**29.0612**
	500	50.0237	48.6857	35.1102	47.3558	**34.5330**
	600	56.4754	46.9777	43.9637	55.6709	**42.5856**

The fitness value is shown in Fig. 10. The greater the fitness value, the better the result. We observe that the fitness value is decreasing with the number of tasks. the maximum average fitness is obtained from SLNPSO, and it has the maximum average in each taskset.

The average makespan is shown in Fig. 11. The minimum average makespan is obtained from SLNPSO. As we can see from Fig. 11, the average makespan of the SLNPSO algorithm is less than compared other algorithms with the increasing of number of tasks. To summarize, the average makespan of the SLNPSO algorithm shows 54% over both the SPSO and SSGA, 50%, 9% over the PSO, G&PSO, respectively.

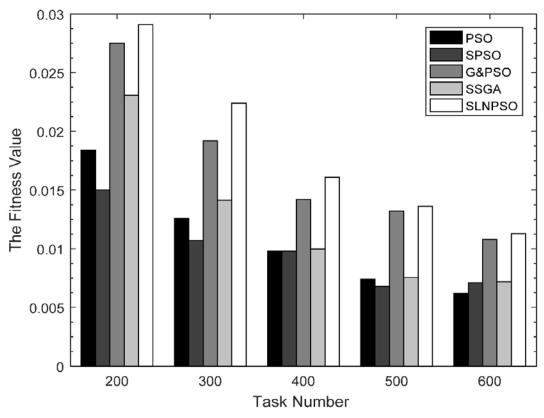

Fig. 10. Fitness value with different tasks

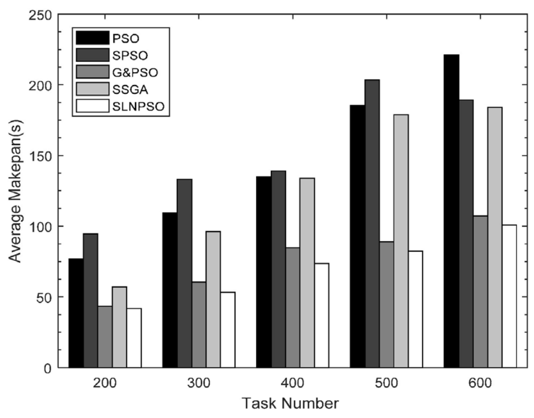

Fig. 11. Average Makespan with different task

The users' expense with the different tasks is shown in Fig. 12. In comparison with other algorithms, the minimum average users' expense is gained by SLNPSO algorithm. On average, the users' expense of the SLNPSO algorithm is 16%, 13%, 10%, 9% lower than SSGA, PSO, SPSO and G&PSO, respectively. The proposed algorithm outperforms other algorithms to reduce the users' expense.

The task average waiting time is depicted in Fig. 13. As we can see, the SLNPSO algorithm exhibits a better waiting time in most tasksets. In sum up, the SLNPSO algorithm shows 44%, 23% and 22% over SSGA, PSO, SPSO, respectively.

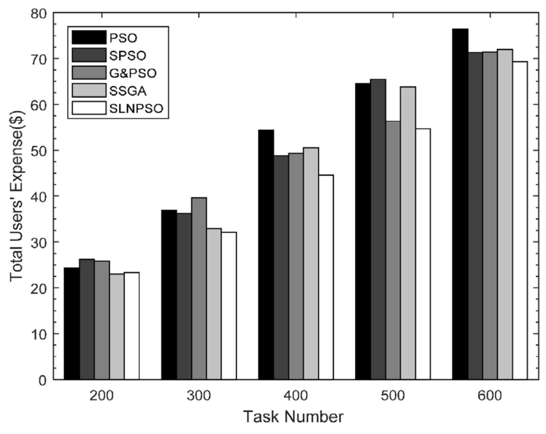

Fig. 12. Total users' expense with different tasks

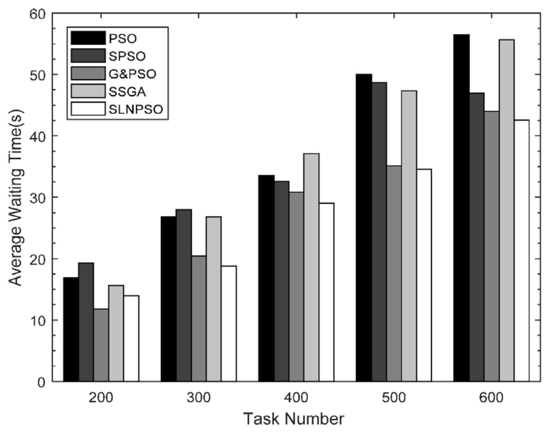

Fig. 13. Average waiting time with different tasks

4.4 Discussion

As we can find from Fig.2 to Fig.5 in the basic task scheduling scenario, the proposed algorithm is efficient to reduce the makespan, as well as improve the system load balance. With the increasing of task number, we also find that the proposed algorithm can make the system load balanced while reducing the makespan. It can be shown from Fig.6 to Fig.13 for the complex task scheduling scenario, the proposed algorithm has the best performance compared with other alternative algorithms: traditional PSO algorithm, SPSO, G&PSO and SSGA algorithm in minimizing makespan and reducing the total users' expense.

Because the greedy policy is used in population initialization, the value of global optimum is shrunken to a reasonable range. By using the neighbor heuristic mechanism and the self-learning strategy, those fasten the proposed algorithm speed for search and convergence to eventually gain the global optimum. Furthermore, compared with other algorithms, our algorithm has been shown fastening convergence speed and strong global search performance. The proposed algorithm can jump the local optimum and obtain the global optimum within a reasonable time. Therefore, we can conclude that the proposed algorithm has the better performance for global search that gains the optimal results on small-scale and large-scale tasksets.

For the proposed algorithm, the task scheduling model is more complex than other compared algorithms. In fact, the cloud environment is complex, and it is important to design a more suitable model in the cloud environment. So, the proposed algorithm is feasibility and availability for the task scheduling on the cloud.

To sum up, the proposed algorithm has demonstrated the best performance for the task scheduling in the basic and complex scenario. The self-learning strategy and neighbor heuristic mechanism facilitates the algorithm to avoid trapping into the local optimal solution and fastens the speed of convergence. Meanwhile, by integrating with the greedy policy, we gain a group of initial solutions with better quality, so that making a further improvement for the efficiency of the proposed algorithm.

5 Conclusions and Future Work

In this paper, we have made a deep insight to the issue of tasks scheduling in the cloud. Firstly, we established a robust cloud task scheduling model, which takes heterogeneity, deadline, overheads of transmission and the cost into account. Then, we proposed a particle swarm optimization scheduling algorithm with the self-learning strategy and the neighbor heuristic mechanism. In our proposed algorithm, the self-learning strategy can improve the diversity of population, and the neighbor heuristic mechanism is adopted to accelerate the convergence speed. Meanwhile, a greedy policy was designed and applied to quickly improve the quality of the initial solutions. Lastly, we have conducted the experiment on Cloudsim to verify the feasibility of the proposed algorithm. Experiment is implemented in the basic and complex task scheduling scenario on both small-scale and large-scale tasksets. The goal in the basic scenario is to minimize the makespan, the latter is to optimize the task scheduling to meet the minimum requirements of deadline and total users’ expense. The results have demonstrated that the proposed algorithm outperforms than that of the state-of-the-art algorithms traditional PSO algorithm, SPSO, G&PSO and SSGA algorithm in both small-scale and large-scale tasksets. And in the complex task scheduling scenario, the average makespan of the proposed algorithm shows 60%, 50%, 40%, 12% over the SPSO, SSGA, PSO and G&PSO algorithm, respectively. The users’ expense of the proposed algorithm is 18%,13%, 13% and 12% better than the SPSO, PSO, G&PSO and SSGA algorithm, respectively.

There are serval avenues in our future work. In the complex scenario of the task scheduling, we consider the resource heterogeneity for task scheduling with the aim of minimizing makespan to meet the deadline and reducing the total users’ expense. However, this complex scenario for task scheduling can be further extended. Since the energy consumption and reliability are two important factors for task scheduling on cloud, these can be accounted for the extension work. Also, this paper is an initial work of our group to do the task scheduling on the cloud. The experiment has been implemented on the Simulation tool Cloudsim. We will extend the experiment to the real cloud system.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No. 61563012, 61802085, 61203109), Guangxi Natural Science Foundation (2014GXNSFAA118371, 2015GXNSFBA139260), Guangxi Key Laboratory of Embedded Technology and Intelligent System Foundation (Grant No. 2018A-04), Guangxi Key Laboratory of Trusted Software Foundation (Grant No. kx201926).

References

[1] Y. Cui, R. Buyya, J. Liu, Guest editorial: cloud computing, China Communications 11(4)(2014) i-ii.

[2] H. Yuan, J. Bi, W. Tan, B.-H. Li, Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds, IEEE Trans. Automation Science and Engineering 14(1)(2017) 337-348.

[3] A.-N. Toosi, R.-O. Sinnott, R. Buyya, Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using Aneka, Future Generation Computer Systems 79(2018) 765-775.

[4] H.G.E.D.H. Ali, I.-A. Saroit, A.-M. Kotb, Grouped tasks scheduling algorithm based on QoS in cloud computing network, Egyptian Informatics Journal 18(1)(2017) 11-19.

[5] A. Bamini, S. Enoch, Dynamic scheduling and resource allocation in cloud, International Journal of Control Theory and Applications 10(3)(2017) 63-72.

[6] S. Mittal, A. Katal, An optimized task scheduling algorithm in cloud computing, in: Proc. 2016 IEEE 6th International Conference on Advanced Computing (IACC), 2016.

[7] C.-C Kao, Mapping virtual tasks onto physical devices for cloud computing, Journal of Computers 29(1)(2018) 40-46.

[8] J. Chase, D. Niyato, Joint optimization of resource provisioning in cloud computing, IEEE Transactions on Services Computing 10(3)(2017) 396-409.

[9] P.-Y. Zhang, M.-C. Zhou, Dynamic cloud task scheduling based on a two-stage strategy, IEEE Transactions on Automation Science and Engineering 15(2)(2018) 772-783.

[10] M. Kumar, K. Dubey, S.-C. Sharma, Job Scheduling algorithm in cloud environment considering the priority and cost of job, in: Proc. Sixth International Conference on Soft Computing for Problem Solving, 2017.

[11] X. Tang, X. Li, Z. Fu, Budget-constraint stochastic task scheduling on heterogeneous cloud systems, Concurrency and Computation: Practice and Experience 29(19)(2017), e4210.

[12] S.-G. Domanal, G.R.M. Reddy, An efficient cost optimized scheduling for spot instances in heterogeneous cloud environment, Future Generation Computer Systems 84(2018) 11-21.

[13] X.-Q. Pham, N.-D. Man, N.D.T. Tri, N.Q. Thai, E.-N. Huh, A cost-and performance-effective approach for task scheduling based on collabo-ration between cloud and fog computing, International Journal of Distributed Sensor Networks 13(11)(2017) 1550147717742073.

[14] H. Aziza, S. Krichen, Bi-objective decision support system for task-scheduling based on genetic algorithm in cloud computing, Computing 100(2)(2018) 65-91.

[15] Z. Zheng, R. Wang, H. Zhong, X. Zhong, An approach for cloud resource scheduling based on parallel genetic algorithm, in: Proc. 3rd International Conference on Computer Research and Development, 2011.

[16] Y.-K. Lin, C.-S. Chong, Fast GA-based project scheduling for computing resources allocation in a cloud manufacturing system, Journal of Intelligent Manufacturing 28(5)(2017) 1189-1201.

[17] K. Duan, S. Fong, S. Siu, W. Song, S. Guan, Adaptive incremental genetic algorithm for task scheduling in cloud environments, Symmetry 10(5)(2018) 168.

[18] M.-A. Tawfeek, A. El-Sisi, A.-E. Keshk, F.-A. Torkey, Cloud task scheduling based on ant colony optimization, in: Proc. 8th International Conference on Computer Engineering & Systems (ICCES), 2013.

[19] A. Khalili, S.-M. Babamir, Makespan improvement of PSO-based dynamic scheduling in cloud environment, in: Proc. 23rd Iranian Conference on Electrical Engineering, 2015.

[20] I. Attiya, X. Zhang, A simplified particle swarm optimization for job scheduling in cloud computing, International Journal of Computer Applications 163(9)(2017) 0975-8887.

[21] J. Ge, S. Sheng, Y. Fang, Cloud resource scheduling algorithm based on improved LDW particle swarm optimization algorithm, in: Proc. Information Technology and Mechatronics Engineering Conference (ITOEC), 2017.

[22] R. Eberhart, J. Kennedy, Particle swarm optimization, in: Proc. the IEEE International Conference on Neural Networks, 1995.

[23] S. Volke, S. Bin, D. Zeckzer, M. Middendorf, G. Scheuermann, Visual analysis of discrete particle swarm optimization using fitness landscape, in: H. Richter, A. Engelbrecht (Eds.), Recent Advances in the Theory and Application of Fitness Landscapes, Vol. 6, Springer, Berlin, Heidelberg, 2014, pp. 487-507.

[24] V. Priyatharsini, S. Grahalakshmi, Load balancing with multiple cloud services using PSO Techniques, in: Proc. 2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE), 2017.

[25] S. Asghari, N.-J. Navimipour, Review and comparison of meta-heuristic algorithms for service composition in cloud computing, Majlesi Journal of Multimedia Processing 4(4)(2016) 28-34.

[26] A. Salman, I. Ahmad, S. Al-Madani, Particle swarm optimization for task assignment problem, Microprocessors and Microsystems 26(8)(2002) 363-371.

[27] S. Xue, W. Shi, X. Xu, A heuristic scheduling algorithm based on PSO in the cloud computing environment, International Journal of u-and e-Service, Science and Technology 9(1)(2016) 349-62.

[28] S.-S. Gill, R. Buyya, I. Chana, M. Singh, A. Abraham, BULLET: particle swarm optimization based scheduling technique for provisioned cloud resources, Journal of Network and Systems Management 26(2)(2018) 361-400.

[29] Z. Zhong, K. Chen, X. Zhai, S. Zhou, Virtual machine-based task scheduling algorithm in a cloud computing environment, Tsinghua Science and Technology 21(6)(2016) 660-667.

[30] R. Buyya, R. Ranjan, R.-N. Calheiros, Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities, in: Proc. International Conference on High Performance Computing & Simulation, 2009.

[31] J. Medhi, Waiting time distribution in a Poisson queue with a general bulk service rule, Management Science 21(7)(1975) 777-782.

[32] Z. Zhou, J. Chang, Z. Hu, J. Yu, F. Li, A modified PSO algorithm for task scheduling optimization in cloud computing, Concurrency and Computation: Practice and Experience 30(24)(2018) e4970.

[33] D. Zhan, H. Lu, W. Hao, D. Jin, Improving particle swarm optimization: using neighbor heuristic and Gaussian cloud learning, Intelligent Data Analysis 20(1)(2016) 167-182.

[34] Y. Shi, R.-C. Eberhart, Empirical study of particle swarm optimization, in: Proc. Proceedings of the 1999 Congress on Evolutionary computation-CEC99 (Cat. No. 99TH8406), 1999.