Journal of Southeast University (English Edition) Vol. 34, No. 4, pp. 430 – 438 Dec. 2018 ISSN 1003—7985

Dependent task assignment algorithm based on particle swarm optimization and simulated annealing in ad-hoc mobile cloud

Huang Bonan Xia Weiwei Zhang Yueyue Zhang Jing Zou Qian Yan Feng Shen Lianfeng

(National Mobile Communications Research Laboratory, Southeast University, Nanjing 210096, China)

Abstract: In order to solve the problem of efficiently assigning tasks in an ad-hoc mobile cloud (AMC), a task assignment algorithm based on the heuristic algorithm is proposed. The proposed task assignment algorithm based on particle swarm optimization and simulated annealing (PSO-SA) transforms the dependencies between tasks into a directed acyclic graph (DAG) model. The number in each node represents the computation workload of each task and the number on each edge represents the workload produced by the transmission. In order to simulate the environment of task assignment in AMC, mathematical models are developed to describe the dependencies between tasks and the costs of each task are defined. PSO-SA is used to make the decision for task assignment and for minimizing the cost of all devices, which includes the energy consumption and time delay of all devices. PSO-SA also takes the advantage of both particle swarm optimization and simulated annealing by selecting an optimal solution with a certain probability to avoid falling into local optimal solution and to guarantee the convergence speed. The simulation results show that compared with other existing algorithms, the PSO-SA has a smaller cost and the result of PSO-SA can be very close to the optimal solution.

Key words: ad-hoc mobile cloud; task assignment algorithm; directed acyclic graph; particle swarm optimization; simulated annealing

DOI: 10.3969/j.issn.1003 – 7985.2018.04.003

Mobile devices (MDs) have gained enormous popularity in our daily life. As a result, mobile applications running on the mobile devices require much more time and energy. In spite of the significant development in the capabilities of these MDs, compared with computer systems, MDs are still limited by storage, lifetime, and computing abilities, etc. Mobile cloud computing (MCC)[1], by enabling offloading part of the workload to a remote cloud, cloudlet or other MDs, opens up a new possibility to further enhance the MDs computing ability and to extend their storage and lifetime.

The ad-hoc mobile cloud (AMC) consists of a number of nodes with different performances, and each node has the capability to serve both as a server and, at the same time, a client. These nodes communicate over radio and operate without the benefit of any infrastructure, thereby creating an opportunistic resource sharing platform. Such characteristics make AMC an ideal solution for scenarios with weak or no internet connection.

AMC coordinates with MDs which are close to each other to process an application together, and offers an efficient solution for reducing the latency of the MDs applications[2]. Moreover, due to the rapid development of energy harvesting technology, MDs are expected to harvest energy from an ambient environment so as to facilitate self-sustainability and perpetual operation. With energy harvesting capability, MDs are more likely to perform task sharing to maximize the overall utility of the whole network.

Workflow can effectively describe the complex constraint relationships between activities. The directed acyclic graph (DAG) is a common way of describing it. In recent years, the research of DAG scheduling has made great progress. Gotoda et al. [3-4] first proposed the scheduling problem of multi-DAGs sharing a set of a distributed resource. Then, they put forward some solutions to the problems such as the minimization of the make span and the fairness of scheduling. Xu et al. [5-7] also made some improvements on this basis. However, only a few studies have been made on the DAG scheduling with a constraint. In this paper, a DAG model representing the dependencies between tasks under the constraint of time delay is proposed.

Task assignment is also an important part of cloud computing. Many intelligent optimization algorithms such as the genetic algorithm, particle swarm algorithm, ant colony optimization are introduced into the research of the task assignment. In Ref. [8], an improved particle swarm optimization (PSO) algorithm applied to workflow scheduling is studied. Rahman et al.[9] proposed the improved PSO algorithm to make the plan for virtual machine migration. In Ref. [10], the genetic algorithm (GA) is used to optimize the path length and the number

Received 2018-02-12, **Revised** 2018-06-01.

Biographies: Huang Bonan (1993—), male, graduate; Xia Weiwei (corresponding author), female, doctor, associate professor, wwxia@seu.edu.cn.

Foundation items: The National Natural Science Foundation of China (No. 61741102, 61471164, 61601122), the Fundamental Research Funds for the Central Universities (No. SJLX_160040).

Citation: Huang Bonan, Xia Weiwei, Zhang Yueyue, et al. Dependent task assignment algorithm based on particle swarm optimization and simulated annealing in ad-hoc mobile cloud [J]. Journal of Southeast University (English Edition), 2018, 34(4): 430 – 438. DOI: 10.3969/j.issn.1003 – 7985.2018.04.003.

of common hops jointly. In Ref. [10], the GA is used to maximize the throughput of the network. Although Truonghuu et al.[11-14] optimized the task assignment in cloud computing in different ways, neither the fairness of the resources' use nor the interference of the transmission in the wireless network is considered. To fill this gap, this paper proposes a model of wireless network ad-hoc mobile cloud communication, and the total energy consumption of all devices and fairness of resource usage of all devices are considered simultaneously.

The PSO outperforms other population-based optimization algorithms, such as the GA and other evolutionary algorithms, due to its simplicity[15-17]. The efficiency of solving various optimization problems was proved in Refs. [18-21]. However, the PSO has disadvantages. It is easy to fall into local optimal solutions, and it shows slow convergence in evolution and poor precision. The SA algorithm has the ability to make it possible to jump out of the trap of local optimal solutions and its convergence has been proved theoretically[22]. The algorithm in this paper combines the particle swarm optimization and simulated annealing with a strong ability to jump out of local optimal solutions, so as to improve convergence speed and precision.

Our main contributions can be summarized as follows: The energy consumption and time delay are regarded as costs at the same time. The dependencies between tasks are converted into a DAG model. The task assignment in the AMC is transformed into an optimization problem and a PSO-SA based algorithm is proposed to solve this complex optimization problem.

1 System Model

The considered ad-hoc mobile cloud is illustrated in Fig. 1. Ad-hoc mobile cloud consists of a set of mobile devices connected by a set of wireless links. The device

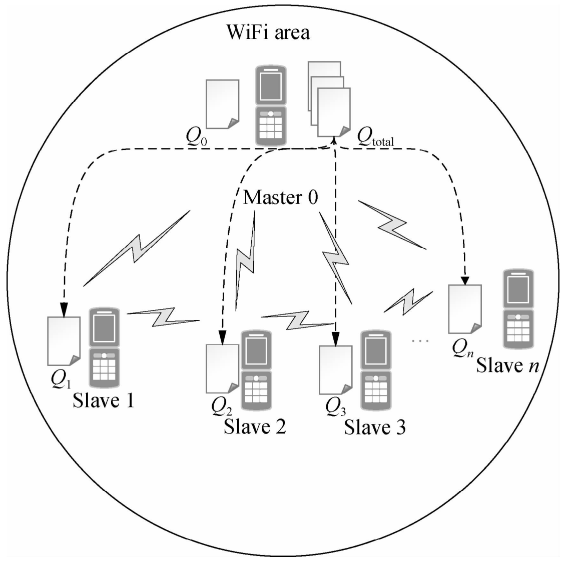

Fig. 1 Task assignment in ad-hoc mobile cloud

which asks for help from neighbors is denoted as a master; in contrast, the device which provides cooperation to the master for offloading is denoted as slave. As shown in Fig. 1, Master 0 is connected to each slave via a wireless link and it has a computing-intensive task Q_{total} to be calculated, but it does not have enough computing resources to execute them, so Master 0 split Q_{total} to $n+1$ parts (Q_0, Q_1, Q_2, $\cdots$, Q_n). Each part is transmitted to a specific slave to be processed or be calculated by local computing. As can be seen from Fig. 1, Q_0 represents the part of tasks calculated by local computing and other parts of tasks are offloaded to different slaves to be calculated. Each task is not independent during the execution, so it is necessary that the slave nodes are also able to collaborate with each other.

1.1 DAG model

Let M represent the total number of the tasks. The dependencies of the tasks can be represented by the DAG models. In this paper, a four-tuple $\boldsymbol{G}=\{T,E,W,C\}$ is used to represent the DAG model.

$T=\{e_{ij}\}$ is the collection of tasks in the DAG. t_i represents task i, where $0 \leqslant i < M$, M is the number of total tasks.

$E=\{e_{i,j}\}$ is the collection of directed edges, where $0 \leqslant i < M$, $0 \leqslant j < M$. If $e_{i,j}=1$, it means that task j cannot begin unless task i is completed.

$W=\{w(i)\}$ is the collection of the computing workload (i. e., the total number of CPU cycles). $w(i)$ denotes task i's computing workload, where $0 \leqslant i \leqslant M$.

$C=\{c_{i,j}\}$ is the collection of the size of computation input data (e. g., the program codes and input parameters). $c_{i,j}$ is the size of computation input data from task i to task j, where $0 \leqslant i < M$, $0 \leqslant j < M$.

In order to give a clearer description of the mathematical model, the following definitions are given:

$\text{pre}(t_i)$ represents the collection of previous tasks of task i.

$\text{suc}(t_i)$ represents the collection of succeeding tasks of task i.

$t_{\text{in}}=\{t \mid \text{pre}(t)=\varnothing\}$ represents the entry task.

$t_{\text{out}}=\{t \mid \text{suc}(t)=\varnothing\}$ represents the exit task.

T_{S}^{i} represents the start time of task i. This paper states defines that a task can be started only when all its previous tasks are completed.

T_{E}^{i} represents the end time of task i and the initial value is infinite.

Let N represent the total number of the slave nodes, $a_m \in [1,N]$ denote the offloading decision of task m. $a_m=n$ means that task m is offloaded to slave n. Furthermore, the decision profile is denoted as $\boldsymbol{A}=(a_1,a_2,\cdots,a_m)$.

For a given decision profile, the DAG model G can be further processed. If two tasks on one edge are assigned to the same node, then $c_{i,j}$ on this edge will be set to be zero, which means that communication energy consumption will not be generated.

Fig. 2 shows the dependencies between eight tasks. Let each circle in the figure represent a task; P_1, P_2, P_3 and P_4 represent four nodes; the edge between task i and task j represents the dependencies between tasks $e_{i,j}$; and the number above the edge between task i and task j represents the size of computation input data $c_{i,j}$. For example, as shown in the figure, $e_{2,5}=1$ and $e_{3,5}=1$, which means that task 5 relies on tasks 2 and 3. These two tasks are assigned to the same node, so the energy consumption generated by transferring input data can be ignored, which means that $c_{2,5}=0$.

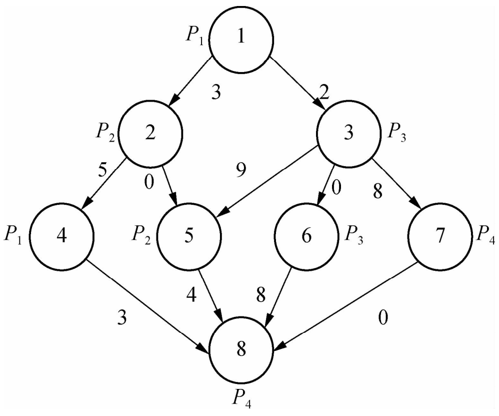

Fig. 2 The dependencies between tasks

1.2 Communication model and calculation model

According to the decision profile $\boldsymbol{A}$, the number of the slave node n which task m is assigned to can be obtained. Then, the slave node n will execute the computation task m and return the results to the master. The execution of task m in the cloud includes three phases: The transmitting phase, the computing phase, and the receiving phase.

In this paper, the energy consumption generated by the receiving phase is not taken into account. This is because the size of the returned result is much smaller than that of computation input data. Thus, the consumption of this phase is low enough to be ignored.

The quality of a wireless link depends on the signal to interference and noise ratio (SINR). The SINR of the link which connects slave node x and slave node y is defined as

$$\mathrm{SINR}_{x,y}(\boldsymbol{A})=\frac{P^{\mathrm{t}}_{x,y}H_{x,y}}{\sigma_{\mathrm{n}}^2 W} \tag{1}$$

where $P^{\mathrm{t}}_{x,y}$ is the transmission power of slave node x offloading task to slave node y; $H_{x,y}$ denotes the channel gain from slave node x and slave node y due to the path loss and shadowing attenuation. The channel gain $H_{x,y}=d_{x,y}^{-\varepsilon}$, where is the distance between slave x and slave y, is the distance attenuation index. σ_{n}^2 denotes the thermal noise power and W is the channel bandwidth.

Using the Shannon capacity theorem, the transmission rate for node x and slave node y can be denoted as

$$R_{x,y}(\boldsymbol{A})=W\log_2(1+\mathrm{SINR}_{x,y}(\boldsymbol{A})) \tag{2}$$

Let $t_j\in\mathrm{pre}(t_k)$ and $\boldsymbol{G}$ represents the DAG model of dependencies between tasks. The transmission time between task j and task k is defined as

$$T_{\mathrm{trs}}^{j,k}(\boldsymbol{A},\boldsymbol{G})=\begin{cases}\dfrac{c_{j,k}}{R_{a_j,a_k}} & a_j\neq a_k\\ 0 & a_j=a_k\end{cases} \tag{3}$$

and

$$E_{\mathrm{trs}}^{j,k}(\boldsymbol{A},\boldsymbol{G})=P_{a_j,a_k}T_{\mathrm{trs}}^{j,k} \tag{4}$$

According to the decision profile $\boldsymbol{A}$, a_j and a_k are the slave node numbers that tasks j and k are offloaded to. P_{a_j,a_k} is the transmission power of slave node a_j to slave node a_k, and R_{a_j,a_k} is the transmission rate of slave node a_j to slave node a_k. As can be seen from Eq. (3) and Eq. (4), the low data transmission rate will result in high energy consumption in the long transmission time.

Let $f_{m,n}$ denote the computation capability of slave n on task m. This paper allows different slave nodes to have different computation capabilities and different tasks to be executed for a slave at different clock frequencies. The computation execution time of task m processed by the node is given by

$$T_{\mathrm{exe}}^{m}=w(m)f_{m,a_m}^{-1} \tag{5}$$

and the energy consumption generated by handling computation task is given by

$$E_m^{\mathrm{e}}=kw(m)f_{m,a_m}^{2} \tag{6}$$

The value of the constant k depends on the specific chip architecture. Combined with the above analysis, energy consumption can be defined for task m as

$$E_m(\boldsymbol{A},\boldsymbol{G})=\sum_{t_j\in\mathrm{pre}(t_m)}E_{\mathrm{trs}}^{j,m}+E_m^{\mathrm{e}} \tag{7}$$

$T_{\mathrm{d}}(\boldsymbol{A},\boldsymbol{G})$ represents the time delay of this task assignment. The way to calculate the start time and the end time of each task is introduced as follows: $U=\{t_i\}$ is a collection of uncalculated tasks where $0\leqslant i<M$. Before the task assignment begins, U contains all the tasks that need to be processed.

Time calculation is described as follows: First, the execution time of the entry task $T_{\mathrm{E}}^{t_{\mathrm{n}}}$ is calculated according to Eq. (5) and the entry task will be removed from the collection U. Then each task t in the collection U will be checked whether all its previous tasks pre(t) have been completed, and if so, task m can be started. The communication time and processing time of task m can be calculated according to Eqs.(3) and (5).

The end time of task m is denoted as

$$T_{\mathrm{S}}^{m}(\boldsymbol{A},\boldsymbol{G}) = \max_{t_j \in \mathrm{pre}(t_m)} (T_{\mathrm{E}}^{j} + T_{\mathrm{trs}}^{j,m}) \tag{8}$$

$$T_{\mathrm{E}}^{m}(\boldsymbol{A},\boldsymbol{G}) = T_{\mathrm{S}}^{m}(\boldsymbol{A},\boldsymbol{G}) + T_{\mathrm{exe}}^{m} \tag{9}$$

Eq. (8) indicates that the start time of the task depends on the longest completion time and the communication time of all the previous tasks. T_{E}^{j} represents the end time of task j. Eq. (9) shows that the end time of a task depends on the start time of the task and the execution time on the node.

The task which has been computed is removed from collection U. Repeat the time calculation until U is empty.

Through this process, the start time and the end time of each task under the given decision profile $\boldsymbol{A}$ can be calculated, and the end time of exit task t_{out} can be defined as the time delay $T_{\mathrm{d}}(\boldsymbol{A},\boldsymbol{G})$ of this assignment.

$$T_{\mathrm{d}}^{i}(\boldsymbol{A},\boldsymbol{G}) = T_{\mathrm{E}}^{t_{\mathrm{out}}}(\boldsymbol{A},\boldsymbol{G}) \tag{10}$$

Finally the evaluation function is defined as

$$\min_{a_m} K(\boldsymbol{A},\boldsymbol{G}) = \delta \sum_{m} E_m(\boldsymbol{A},\boldsymbol{G}) + \mu T_{\mathrm{d}}(\boldsymbol{A},\boldsymbol{G}) \tag{11}$$

s. t.

$$T_{\mathrm{d}} < T_{\mathrm{in}}$$
$$a_m \in [1, N]$$

where δ and μ are weights for energy consumption and time delay, respectively. T_{in} represents the maximum delay of this assignment. Constraint $T_{\mathrm{d}} < T_{\mathrm{in}}$ ensures that the total time delay is smaller than the deadline, and constraint $a_m \in [1, N]$ gives the definition of the decision variables.

As can be seen from the formulation, this task assignment is an NP-hard problem. Solving such large size problems using a mathematical programming approach will take a large amount of computational time. However, the heuristic algorithm can obtain a result which is close to the optimal solution in a relatively small amount of time. Therefore, this paper adopts the heuristic algorithm to solve the optimization problem.

2 PSO-SA Task Assignment Algorithm

In this section, an algorithm based on the PSO and SA is proposed to solve the optimization problem Eq. (11).

2.1 Particle swarm optimization

Let $x_i^{\mathrm{id},m} \in [1, N]$ represent the decision of task m of particle I, $x_i^{\mathrm{id},m} = n$ means that task m is offloaded to slave n. The position of particle i is defined as $X_i^{\mathrm{id}} = (x_i^{\mathrm{id},1}, x_i^{\mathrm{id},1}, \cdots, x_i^{\mathrm{id},m})$.

When the PSO-SA is used, an evaluation function is necessary to calculate the fitness value of each particle in the swarm and the evaluation function. Let $\boldsymbol{G}$ represent the DAG model about dependencies between tasks, the fitness value is defined as

$$F_i(X_i^{\mathrm{id}}, G) = -K(X_i^{\mathrm{id}}, G) \tag{12}$$

The velocity of particle i is denoted as $V_i = (v_i^1, v_i^2, \cdots, v_i^m)$ and this paper uses the traditional PSO method,

$$v_i^w = w' v_i^w + c_1 r_1 (p_i^w - x_i^{\mathrm{id},w}) + c_2 r_2 (g^w - x_i^{\mathrm{id},w}) \tag{13}$$

where v_i^w is the velocity of the particle i for task w; w' is the inertial weight, which affects the search accuracy and convergence speed. $x_i^{\mathrm{id},w}$ is the current position of the particle i for task w; p_i^w and g^w are the personal best position and group best position of the particle i for task w; r_1 and r_2 are the random numbers between (0,1); and c_1 and c_2 are the learning factors. Eq. (13) allows the velocity to be updated according to the personal best solution $P_i = (p_i^1, p_i^2, \cdots, p_i^m)$ and group best solution $G = (g_i^1, g_i^2, \cdots, g_i^m)$, which record the optimal decision of the individual and the swarms, respectively.

In population-based search optimization methods, during the early part of the search, considerably high diversity is necessary to allow the use of the full range of the search space. On the other hand, during the latter part of the search, when the algorithm is close to the optimal solution, micro-tuning of the current solutions is an important method to help us find the global optimal solution efficiently. Based on the above discussion, w is denoted as

$$w' = (w_1 - w_2) \frac{I_{\max} - I}{I_{\max}} + w_2 \tag{14}$$

where w_1 and w_2 are the initial and final values of the inertia weight, respectively; I is the current iteration number; and $I_{\max}$ is the maximum number of allowable iterations.

As mentioned in Section 1, a major disadvantage of the traditional PSO is that it is easy to obtain local optimal solutions. In order to avoid this shortcoming, when selecting the optimal solution of the population, the minimum fitness value is not directly selected, but the probability of particle i being selected as the population is calculated as

$$p_i^{\mathrm{b}} = \frac{-F_i(X_i^{\mathrm{id}}, \boldsymbol{G})}{\sum_i \exp(-F_i(X_i^{\mathrm{id}}, \boldsymbol{G}))} \tag{15}$$

A position is updated as follows: First, by

$$x_{i,k+1}^{\mathrm{id}} = x_{i,k}^{\mathrm{id}} + v_{k+1}^{i} \tag{16}$$

a new coding sequence containing illegal coding can be obtained. Then, legalize the illegal coding. The main processing methods include taking absolute values, taking integers upwardly, taking remainders, etc. Specific methods are described as

$$x_i^{\mathrm{id}} = \begin{cases} [\,|x_i^{\mathrm{id}}|\,] & x_i^{\mathrm{id}} \in [0, N] \\ \mathrm{mod}([\,|x_i^{\mathrm{id}}|\,], N) & \text{else} \end{cases} \tag{17}$$

By the legalization processing method, legal and valid position vectors can be obtained.

2.2 Simulated annealing

Simulated annealing is used to do further processing of the result of PSO to avoid falling into local optimal solutions. When the particle's current fitness is better than before, the update of the particle's position will be accepted with probability 1. When the particle's current fitness is worse than before, the update will be accepted with a certain probability p_i calculated by

$$p_i = \begin{cases} 1 & F_i^k \leqslant F_i^{k+1} \\ \exp\left(-\dfrac{F_i^k - F_i^{k+1}}{T} \right) & F_i^k > F_i^{k+1} \end{cases} \tag{18}$$

where F_i^k represents the previous fitness of particle i; F_i^{k+1} represents the current fitness of particle i; T represents the annealing temperature which controls the process to optimize the direction of searching for the optimal value.

The cooling process of T will ensure the convergence to the optimal solution. In this paper, the cooling process is denoted as

$$T = \frac{T_0}{\lg(1+I)} \tag{19}$$

where T_0 represents the initial temperature. As can be seen from Eq. (19), when the number of iterations increases, the value of T decreases gradually.

Updated selection for the particle's position based on simulated annealing is given as follows:

Step 1 Calculate the fitness value of particle i according to Eq. (12).

Step 2 If the new fitness value is better than the previous fitness value, then the position of particle i will be updated, otherwise jump to Step 3.

Step 3 Generate a random number between 0 and 1, which is denoted as r.

Step 4 Calculate the probability p_i of particle i by Eq. (18).

Step 5 If p_i is larger than r, then the position of the particle i will be updated, otherwise it will be rejected.

Step 6 Repeat Steps 1 to 5 until all the particles have been calculated once.

Step 7 Update the annealing temperature by Eq. (19).

2.3 PSO-SA

The specific process of the proposed PSO-SA algorithm is described as follows:

1) Before the PSO-SA starts, the maximum iteration number $I_{\max}$, the population size inertia weight, learning factors c_1, c_2 and other parameters will be set. Then, the initial positions and velocities of the population are set randomly. In each iteration, PSO is adopted before SA. In the PSO, each particle will be updated based on the individual optimal solution and swarm optimal solution.

2) After the PSO is finished, the update selection is used based on SA for all particles in the swarm to do further processing to avoid falling into local optimal solutions.

3) Finally, judge whether the iteration number reaches the maximum value. If not, update the inertia weight and annealing temperature by Eq.(14) and Eq.(19), respectively, then return to PSO for the next iteration. Otherwise, output the optimal result, and end the algorithm.

Algorithm 1 PSO-SA

```
  Input: $T_{\text{in}}$, $N$, $M$, $F$, $w_1$, $w_2$, $c_1$, $c_2$, $I_{\max}$, $T_0$, $P$, $H$, $G$;
  Output: $K$.
  Set iteration number index $I = 0$
  Get the total number of task $m$
  Get the total number of node $n$
  for each particle $k = 1, 2, \cdots, p$
    Initialize the particle with $X_k$ and $V_k$
  end for
  while $I < I_{\max}$
    $I = I + 1$
    for each particle $k = 1, 2, \cdots, p$
  Calculate $T_{\text{d}}^k$ by time calculation and Eq. (10)
      if $T_{\text{d}}^k > T_{\text{in}}$
        rejected this update
      Calculate fitness value $F_k$ by Eq. (12)
      if the fitness value is better than $P_i$
        $P_k \leftarrow X_k^{\text{id}}$
        $P_k^{\text{F}} \leftarrow F_k$
      end if
  end for
  for each particle $k = 1, 2, \cdots, p$
      Calculate $p_i^{\text{b}}$ by Eq. (15)
  end for
Select the optimal solution of population $\boldsymbol{G}$
  for each particle $k = 1, 2, \cdots, p$
    Calculate particle velocity by Eq. (13)
  Update particle position by Eqs. (16) and (17)
    if $F_i^k > F_i^{k+1}$
Calculate acceptance probability $p_i$ by Eq. (18)
  Generate a random number $r$ between 0 and 1
        if $r < p_i$
          Accept the update
        end if
      end if
    end for
    Update inertia weight by Eq. (14)
    Update annealing temperature by Eq. (19)
    end while
```

3 Simulation Results

In this section, the performance of the proposed PSO-SA algorithm will be evaluated. The AMC scenario considers that all mobile devices are randomly scattered over a 100 m × 100 m region, and the specific location is shown in Tab. 1. The data size of each task is in the intervals [5, 10] MB and the computing load in [100, 200] instructions, and the speed of each task being processed by each slave is in [5 000, 7 000] instruction/s. The dependencies between tasks are shown in Fig. 3.

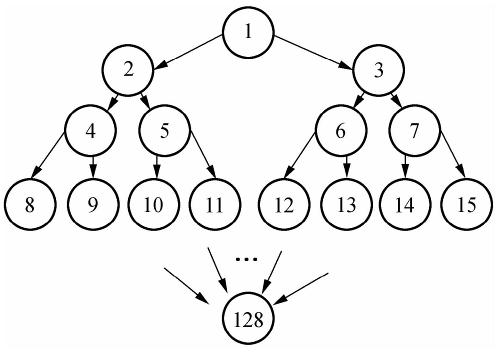

Fig. 3 The dependencies between tasks

For simplicity, task 1 is denoted as the entry task and task 128 is denoted as the exit task. Except for the exit task, each task has two succeeding tasks, and the total number of tasks is 128.

The parameters for the PSO-SA are given as follows: The number of particles is $Z = 20$; initial weights are set to be $\delta = \mu = 0.4$; learning factors are denoted as $c_1 = 1.8$, $c_2 = 1.3$; the maximum number of iterations $I_{\max} = 100$; the initial inertia weight and final inertia weight are $w_1 = 0.7$, $w_2 = 0.3$; the initial temperature is set to be $T_0 = 150$; the maximum delay $T_{\text{in}} = 80$ s. In order to reduce the impact of the randomness of the heuristic algorithm on the simulation results, all data in this paper is run 50 times under the same parameters, and the average results are obtained.

Tab. 1 The location of all devices

Device	Location/(m,m)	Device	Location/(m,m)
1	38,17	6	19,75
2	24,67	7	45,85
3	58,12	8	14,82
4	74,26	9	18,49
5	25,61	10	62,39

3.1 Performance of PSO-SA algorithm

In this part, the simulation results of the PSO-SA will be compared with those of the optimal solution to prove the performance of the PSO-SA. Device 10 is denoted as the master, devices 1-9 are denoted as slave nodes 1-9, respectively. The number of tasks is 128. To be specific, the total size of computation input data is 12 MB, and the total number of CPU cycles is 1.2×10^4.

The result of the optimal decision is generated by the enumeration method of which the time complexity is $O(M^N)$, and the time complexity of the PSO-SA is $O(M \times N \times I_{\max} \times Z)$.

As shown in Tab. 2 and Tab. 3, the PSO-SA produces a little more energy consumption and results in a longer time delay, but the results of the PSO-SA can be very close to that of the enumeration method, which proves the accuracy of the PSO-SA.

Tab. 2 The result of time delay

Device	Time delay/s		Percentage of differences/%
	PSO-SA	EM	
Slave 1	10.22	9.96	2.61
Slave 2	2.78	2.62	6.11
Slave 3	5.01	4.74	5.70
Slave 4	5.79	6.32	8.39
Slave 5	5.12	4.71	8.70
Slave 6	8.52	8.14	4.67
Slave 7	10.07	9.82	2.55
Slave 8	9.44	9.13	3.40
Slave 9	8.64	9.28	6.90
Master	8.12	7.55	7.55
Total	73.71	72.27	1.99

Tab. 3 The result of energy consumption

Device	Energy consumption/J		Percentage of differences/%
	PSO-SA	EM	
Slave 1	8.99	8.76	2.63
Slave 2	5.61	5.12	9.57
Slave 3	3.28	3.29	0.30
Slave 4	1.24	1.30	4.62
Slave 5	4.01	3.79	5.80
Slave 6	9.08	8.58	5.83
Slave 7	0.23	0.21	9.52
Slave 8	8.90	8.60	3.49
Slave 9	5.51	5.80	5.00
Master	9.37	8.54	9.72
Total	56.22	53.99	4.13

The enumeration method will take much more time than the PSO-SA when it comes to a great number of tasks or the number of nodes. For example, when $N = 10$ and $M = 128$, the enumeration method will take 10^{128} times operations to generate a result. While the PSO-SA takes only 4×10^3 times operations, the time complexity of which is much lower than that of the enumeration method.

3.2 Comparison with heuristic algorithms

In this part, the performance of the PSO-SA and other heuristic algorithms are compared. Binary particle swarm optimization (BPSO)[23] and the genetic algorithm (GA)[24], which are frequently used in task assignments, are chosen as a comparison. In a GA, a population of candidate solutions to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties which can be mutated and altered. BPSO is a computational method that optimizes the objective function by iteratively trying to improve a candidate solution with regard to a given degree of quality and the position of each particle is binary. In order to guarantee the same time complexity, the number of particles and the maximum number of iterations for three algorithms are the same.

The parameters of the scene and the ones of the PSO-SA are consistent with the above.

The parameters for BPSO are given as follows: The number of particles is 20; initial weights are set to be $\delta = \mu = 0.4$; learning factors are denoted as $c_1 = 1.8$, $c_2 = 1.3$; the maximum number of iterations $I_{\max} = 100$; the initial inertia weight and final inertia weight are $w_1 = 0.7$, $w_2 = 0.3$; the maximum delay $T_{\text{in}} = 80$ s.

The parameters for the GA are given as follows: The number of particles is 20; initial weights are set to be $\delta = \mu = 0.4$; the maximum number of iterations $I_{\max} = 100$; the maximum delay $T_{\text{in}} = 80$ s. The crossover probability and mutation probability are set to be 0.7 and 0.005, respectively.

In this part of the simulation, device 1 is denoted as master, device 2-10 are denoted as slaves 1-9, respectively. The number of tasks is 128. To be specific, the total size of computation input data is 20 MB, and the total number of CPU cycles is 2.0×10^4.

Cost in this simulation is the optimization goal K in Eq. (11). As can be seen from Fig. 4, all these three algorithms in the simulation can effectively decrease the cost. In addition, in the case of the same number of iterations, the PSO-SA can reduce more costs than the other two. Fig. 5 shows that when the number of tasks is small,

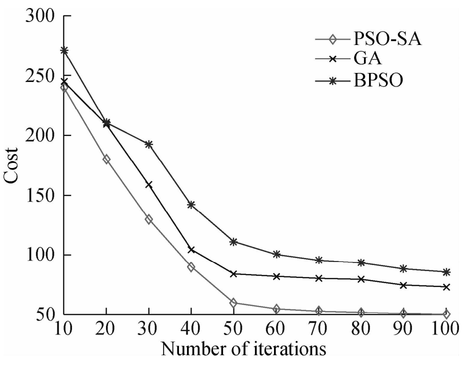

Fig. 4 A comparison of cost as the number of iterations changes

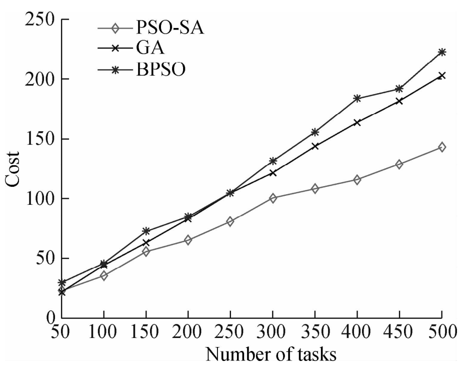

Fig. 5 A comparison of cost as the number of tasks changes

the performance of the three algorithms is almost the same. However with the increase of the tasks, the performance of the PSO-SA will be better than that of the other two heuristic algorithms.

Fig. 6 and Fig. 7 show different performances of different heuristic algorithms in energy consumption and time delay, respectively. In this part of the simulation, device 1 is denoted as the master, device 2-10 are denoted as slave nodes. The relevant parameters are consistent with the previous simulation.

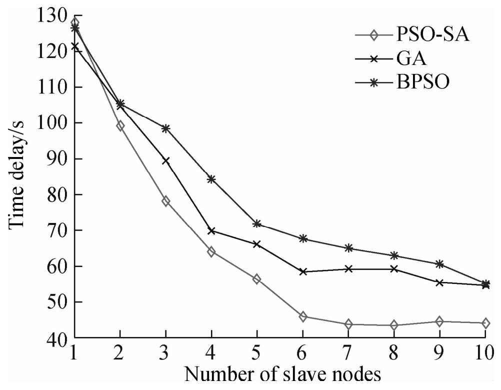

Fig. 6 Comparison of the time delay

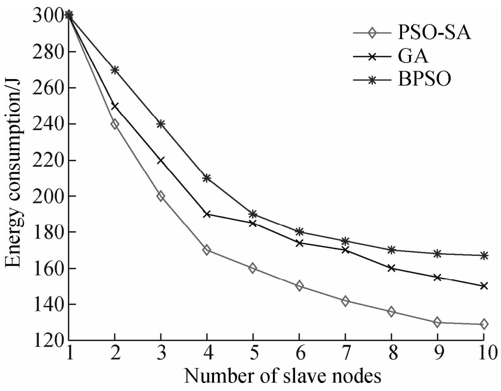

Fig. 7 Comparison of the energy consumption

As can be seen from Fig. 6 and Fig. 7, when the number of slave nodes is small, the performance of the three algorithms is almost the same. But, with the increase in the number of slave nodes, the performance of the PSO-SA is better than that of the other two heuristic algorithms. When the number of slave nodes continues to increase, the results of the three algorithms tend to be stable. This is because in the case of the small number of salve nodes, the probability of the new candidate solution result being better than that of previous optimal solution is great. However, as the slave node continues to increase, this probability will drop, which leads to the gradual stabilization of the value. In order to compare the performances of the PSO-SA with other heuristics algorithms more clearly, Tab. 4 shows the improved ratio of the PSO-SA in terms of average energy consumption and time delay. In Tab. 4, the average energy consumption of BP-

SO, GA, PSO-SA algorithms is the average energy consumption of all devices in AMC when the number of slave nodes changes from 1 to 10, and the average time delay is the average energy time delay produced by different algorithms when the number of slave nodes changes from 1 to 10. Energy consumption improved by PSO-SA represents the ratio of the reduced average energy consumption of PSO-SA to that of BPSO or GA (R_1). The time delay improved by PSO-SA represents the ratio of the reduced average time delay of PSO-SA to that of BPSO or GA (R_2). It can be seen from Tab. 4 that compared to the BPSO and GA, the performance of the PSO-SA has been greatly improved in terms of energy consumption and time delay.

Tab. 4 The improved ratio of PSO-SA

Parameter	Algorithm		
	BPSO	GA	PSO-SA
Average energy consumption/J	207.82	195.47	175.72
Average time delay/s	79.93	73.85	64.78
R_1	15.41	10.10	0
R_2	17.93	12.28	0

4 Conclusion

In this paper, for the task assignment problem, time delay and total energy consumption of all devices under the constraint of certain time delay are considered as costs at the same time. The dependencies of tasks are converted into a DAG model, and PSO and SA are combined to make the decision for the task assignment problem. The simulation results indicate that compared with other population-based optimization algorithms, the PSO-SA has a small cost and the result of the PSO-SA is close to the optimal solution. In our future work, the mobility of MDs is going to be considered. This will make our approach more practical and effective.

References

[1] Chen M, Hao Y X, Li Y, et al. On the computation offloading at ad hoc cloudlet: Architecture and service modes[J]. *IEEE Communications Magazine*, 2015, **53**(6):18 – 24. DOI:10.1109/mcom.2015.7120041.

[2] Wang S, Chan K, Urgaonkar R, et al. Emulation-based study of dynamic service placement in mobile micro-clouds[C]//*Military Communications Conference*. Tampa, FL, USA, 2015:1046 – 1051.

[3] Gotoda S, Ito M, Shibata N. Task scheduling algorithm for multicore processor system for minimizing recovery time in case of single node fault[C]//*International Symposium on Cluster, Cloud and Grid Computing*. Ottawa, Canada, 2012:260 – 267.

[4] Darbha S, Agrawal D P. Optimal scheduling algorithm for distributed-memory machines[J]. *IEEE Transactions on Parallel and Distributed Systems*, 1998, **9**(1): 87 – 95. DOI:10.1109/71.655248.

[5] Xu Y, Li K, Khac T T, et al. A multiple priority queueing genetic algorithm for task scheduling on heterogeneous computing systems [C]//*International Conference on High Performance Computing and Communication*. Exeter, UK, 2012:639 – 646.

[6] Li L, Li D, Song Y, et al. An effective list scheduling algorithm for homogeneous multi-core processor [C]//*IEEE International Conference on Anti-Counterfeiting*. Xiamen, China, 2014:1 – 5.

[7] Geng X, Xu G, Wang D, et al. A task scheduling algorithm based on multi-core processors[C]//*International Conference on Mechatronic Science*. Amsterdam, the Netherlands, 2011:942 – 945.

[8] Cheng H. A High efficient task scheduling algorithm based on heterogeneous multi-core processor[C]//*International Workshop on Database Technology and Applications*. Xiamen, China, 2010:1 – 4.

[9] Rahman A, Jin J, Cricenti A, et al. A cloud robotics framework of optimal task offloading for smart city applications[C]//*Global Communications Conference*. Singapore, 2017:1 – 7.

[10] Xu A, Yang Y, Mi Z, et al. Task scheduling algorithm based on PSO in cloud environment[C]//*International Conference on Scalable Computing and Communications and ITS Associated Workshops*. Beijing, China, 2016: 1055 – 1061.

[11] Truonghuu T, Tham C K, Niyato D. A stochastic workload distribution approach for an ad hoc mobile cloud [C]//*International Conference on Cloud Computing Technology and Science*. Huangshan, China, 2015:174 – 181.

[12] Monoyios D, Vlachos K. Multiobjective genetic algorithms for solving the impairment-aware routing and wavelength assignment problem[J]. *Journal of Optical Communications and Networking*, 2010, **3**(1):40 – 47. DOI:10.1364/jocn.3.000040.

[13] Hajjem L, Benabdallah S. An MMAS-GA for resource allocation in multi-cloud systems[C]// *Internet Technology and Secured Transactions*. Cambridge, UK, 2017: 421 – 426.

[14] Mandal S, Bhattacharyya S. Secret data sharing in cloud environment using steganography and encryption using GA[C]//*International Conference on Green Computing and Internet of Things*. Xi'an, China, 2016: 1469 – 1474.

[15] Angeline P J. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences [C]//*International Conference on Evolutionary Programming*. Berlin, Heidelberg: Springer, 1998:601 – 610.

[16] Robinson J, Rahmat-Samii Y. Particle swarm optimization in electromagnetics[J]. *IEEE Transactions on Antennas and Propagation*, 2004, **52**(2): 397 – 407. DOI: 10.1109/tap.2004.823969.

[17] Poli R. *Analysis of the publications on the applications of particle swarm optimisation* [M]. Hindawi Publishing Corp., 2008.

[18] Omran M, Engelbrecht A P, Salman A. Particle swarm optimization method for image clustering [J]. *International Journal of Pattern Recognition and Artificial Intelli-*

gence, 2005, **19** (3): 297 - 321. DOI: 10. 1142/ s0218001405004083.

[19] Hettenhausen J, Lewis A, Mostaghim S. Interactive multi-objective particle swarm optimization with heatmap-visualization-based user interface[J]. *Engineering Optimization*, 2010, **42** (2): 119 - 139. DOI: 10. 1080/ 03052150903042632.

[20] Mudjihartono P, Jiamthapthaksin R, Tanprasert T. Parallelized GA-PSO algorithm for solving job shop scheduling problem[C]// *International Conference on Science in Information Technology*. Zhuhai, China, 2017:103 - 108.

[21] Sujan S, Devi R K. A batchmode dynamic scheduling scheme for cloud computing[C]//*IEEE Communication Technologies*. Thuckalay, India, 2015:297 - 302.

[22] Gabi D, Ismail A S, Zainal A, et al. Cloud scalable multi-objective task scheduling algorithm for cloud computing using cat swarm optimization and simulated annealing[C]// *IEEE International Conference on Information Technology*. Thuckalay, India, 2017:1007 - 1012.

[23] Guo W, Li J, Chen G, et al. A PSO-optimized real-time fault-tolerant task allocation algorithm in wireless sensor networks[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2015, **26**(12):3236 - 3249.

[24] Lu H, Cao J, Lv S, et al. A case study of task priority effects in GA for cluster based DAG scheduling[C]//*International Conference on Information Society*. London, UK, 2016:157 - 162.

Ad-hoc 移动朵云中基于粒子群优化和模拟退火优化的任务分配算法

黄博南　夏玮玮　章跃跃　张　静　邹　倩　燕　锋　沈连丰

(东南大学移动通信国家重点实验室, 南京 210096)

摘要:为了在 ad-hoc 移动朵云中高效率地解决任务分配这一核心问题,提出了一种基于启发式算法的任务分配算法. 粒子群优化和模拟退火优化的任务分配算法(PSO-SA)将任务之间的依赖关系转化为有向无环图(DAG)模型,其中各个节点上的数值表示任务产生的负载,DAG 的各个边的数值表示传输产生的负载. 为了模拟 ad-hoc 移动朵云的任务分配环境,建立了数学模型来描述各个子任务之间的依赖关系并定义各个子任务的卸载成本. PSO-SA 用于任务分配决策并最小化所有移动设备的成本,能耗和时间延迟同时作为卸载成本. PSO-SA 结合了粒子群优化和模拟退火优化的优势,通过以一定概率选取最优解的方式,避免算法过早落入局部最优解,同时保证算法收敛速度. 仿真结果表明,与其他现有算法相比,PSO-SA 算法产生的卸载成本较低并且其结果可以非常接近最优解.

关键词:ad-hoc 移动朵云;任务分配算法;有向无环图;粒子群优化;模拟退火优化

中图分类号:TN929.5