Task scheduling based on ant colony optimization in cloud environment

Qiang Guo

Citation: AIP Conference Proceedings **1834**, 040039 (2017); doi: 10.1063/1.4981635
View online: http://dx.doi.org/10.1063/1.4981635
View Table of Contents: http://aip.scitation.org/toc/apc/1834/1
Published by the American Institute of Physics

Task Scheduling Based on Ant Colony Optimization in Cloud Environment

Qiang Guo[a)]

College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065 China.

[a)]helloworldwelcom@163.com

Abstract. In order to optimize the task scheduling strategy in cloud environment, we propose a cloud computing task scheduling algorithm based on ant colony algorithm. The main goal of this algorithm is to minimize the makespan and the total cost of the tasks, while making the system load more balanced. In this paper, we establish the objective function of the makespan and costs of the tasks, define the load balance function. Meanwhile, we also improve the initialization of the pheromone, the heuristic function and the pheromone update method in the ant colony algorithm. Then, some experiments were carried out on the Cloudsim platform, and the results were compared with algorithms of ACO and Min-Min. The results shows that the algorithm is more efficient than the other two algorithms in makespan, costs and system load balancing.

Key words: task scheduling; cloud computing; makespan; load balance.

INTRODUCTION

Cloud computing is the latest computing model for a variety of applications, data and IT services over the web [1]. At present, cloud computing is applied to all walks of life in society. In the specific implementation process of cloud computing, task scheduling or resource scheduling is an unavoidable link, which directly determines the efficiency of the whole system. The cloud computing system has a large scale of resources, heterogeneous resources, wide user base, different types of application tasks, QoS target constraints are different, cloud computing systems have to deal with a large number of user tasks and massive data [2]. Therefore the cloud computing task scheduling strategy has been the hot spot that is difficult to study, and we need an efficient algorithm for task scheduling in the cloud environment [3].

A good task scheduler should adapt its scheduling strategy to the changing environment and the types of tasks [4]. And task scheduling problems are a typical NP-hard problem. Therefore, a dynamic task scheduling algorithm, such as ant colony optimization (ACO), is appropriate for clouds. ACO can be used to solve many NP hard problems such as traveling salesman problem [5], graph coloring problem [6], vehicle Routing and scheduling problems [7].In this paper we proposed a Multi-objective Optimization Algorithm for Cloud Computing Task Scheduling Based on Improved Ant Colony Algorithm (MO-ACO) to find the optimal resource allocation for each task in the dynamic cloud System which minimizes the makespan and costs of tasks on the entire system, and balance the entire system load. Then, this scheduling strategy was simulated using the Cloudsim toolkit package. Experimental results compared to Ant Colony Optimization (ACO) and Min-Min showed the MO-ACO algorithm satisfies expectation.

The organization of paper is as following. Section II introduces the related work. Section III introduces a cloud model and presents the problem statement of the multi-objective task scheduling. Section IV details the proposed MO-ACO algorithm. Section V presents the simulation results. Finally, Section VI concludes this paper.

2017 5th International Conference on Computer-Aided Design, Manufacturing, Modeling and Simulation (CDMMS 2017)
AIP Conf. Proc. 1834, 040039-1–040039-11; doi: 10.1063/1.4981635
Published by AIP Publishing. 978-0-7354-1504-1/$30.00

RELATED WORK

At present, the cloud computing task scheduling mechanism has not yet formed a unified standard and norms. Many scholars have studied the task scheduling from the makespan, the optimal span, the cost, the reliability, the energy consumption and so on as the optimization goal according to the characteristics of cloud computing task scheduling.[8] focus on virtual machine load balancing, and propose a cloud computing task scheduling algorithm based on load balancing ant colony optimization algorithm. [9] proposed a task scheduling algorithm based on improved particle swarm, which takes into account the total task completion time and the total task completion cost, but does not consider the system load balancing. [10] focus on multi-dimensional QoS, and propose a multi-dimensional QoS cloud scheduling algorithm based on immune clone to meet the resource load and user's time requirement. [11] proposed based on ACO and CUCKOO hybrid algorithm to reduce the task execution time. A multi-input multi-output feedback control of dynamic resource scheduling algorithm was proposed [12] to guarantee optimal effectiveness under time constraints. This algorithm considers the task execution time, cost, and utilization of resources (CPU, Memory). The paper [13] proposes a multi-objective task scheduling method by minimizing makespan and costs in a heterogeneous multi-cloud environment.

MODEL AND PROBLEM STATEMENT

Task Scheduling Model

Cloud computing task scheduling can be described as the allocation of n independent tasks assigned to m virtual machine implementation, which according to the optimization objectives to achieve, build a match between tasks and virtual machine to achieve optimal scheduling.

In order to simplify the complexity of the scheduling process, make the following assumptions:

1) The tasks are independent of each other and there is no dependency before or after
2) The unit cost of running the task on each resource is known
3) The task is not interrupted during execution

Figure 1 shows the process of task scheduling.

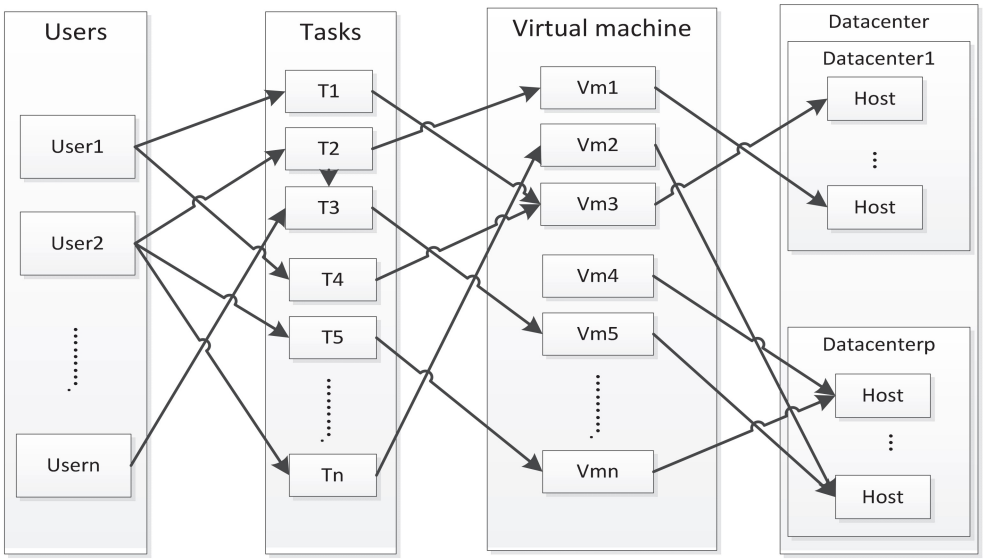

FIGURE 1. Task scheduling process

Problem Statement

Task set is defined as T={$t_1,t_2,...,t_m$},m represents the number of tasks. The virtual machine collection is defined as VM={$vm_1,vm_2,...,vm_n$}, n represents the number of virtual machines. The matching relationship of a task on a virtual machine can be represented by a matrix M:

$$M = \begin{pmatrix} m_{11} & m_{12} & ... & m_{1n} \\ m_{21} & m_{22} & ... & m_{2n} \\ ... & ... & ... & ... \\ m_{m1} & m_{m2} & ... & m_{mn} \end{pmatrix} \tag{1}$$

Where m_{ij} represents the allocation relationship between the i-th task and the j-th virtual machine. The expected execution time of the task on the virtual machine is represented by the matrix ET:

$$ET = \begin{pmatrix} et_{11} & et_{12} & ... & et_{1n} \\ et_{21} & et_{22} & ... & et_{2n} \\ ... & ... & ... & ... \\ et_{m1} & et_{m2} & ... & et_{mn} \end{pmatrix} \tag{2}$$

Here,

$$et_{ij} = \frac{t_{length\_i}}{vm_{comp\_j}} \tag{3}$$

Represents the execution time of task i on virtual machine j. t_{length_i} indicates the length of task i, that is, the total instruction size that needs to be executed. vm_{comp_j} represents the processing power of the virtual machine j. vm_{comp_j} is calculated as:

$$vm_{comp\_j} = vm_{mips\_j} \times vm_{penum\_j} \tag{4}$$

In the formula, vm_{mips_j} represents the computing power of the virtual machine j. vm_{penum_j} represents the number of cpus for virtual machine j. The transmission time of the task can be expressed by the matrix ER:

$$ER = \begin{pmatrix} er_{11} & er_{12} & ... & er_{1n} \\ er_{21} & er_{22} & ... & er_{2n} \\ ... & ... & ... & ... \\ er_{m1} & er_{m2} & ... & er_{mn} \end{pmatrix} \tag{5}$$

Here,

$$er_{ij} = \frac{t_{inputfilesize\_i}}{vm_{bw\_j}} \tag{6}$$

Indicates the time at which task i is transferred to virtual machine j. $t_{inputfilesize\_i}$ represents the size of the data i for task i. vm_{bw_j} represents the bandwidth of the virtual machine j. Therefore, the completion time of a single task is:

$$TaskRunTime_{ij} = et_{ij} + er_{ij} \tag{7}$$

Then the completion time of the virtual machine j is the sum of the completion times of all the tasks assigned to the virtual machine, expressed as:

$$vm_{\text{completetime}\_j} = \sum_{i=1}^{k} TaskRunTime_{ij} \tag{8}$$

Since each virtual machine executes in parallel, the task's completion time is the execution time of the last virtual machine that completes the task. It can be expressed as:

$$C\_Time(I) = \max(vm_{completetime\_j}) \quad j \in [1, n] \tag{9}$$

Assuming the unit cost of virtual machine j is 3, the cost of completing all subtasks is defined as formula (10):

$$TotalCost(I) = \sum_{j=1}^{n} vm_{completetime\_j} \times \text{UCost}_{\text{j}} \tag{10}$$

Define the Time Constraint Function in the Task Scheduling Process

The time constraint function in the task scheduling process is defined as formula (11):

$$Time\_cf(I) = \frac{C\_Time(I) - C\_Time_{\min}}{C\_Time_{\max} - C\_Time_{\min}} \tag{11}$$

In the formula, C_Time_{min} indicates the time of the task which is running on the best virtual machine, C_Time_{max} indicates the time at which the task runs on the worst virtual machine. They are calculated as formula (11), (12).

$$C\_Time_{\min} = \frac{\sum_{\text{i}=1}^{m} t_{length\_i}}{n \times \max(vm_{comp\_j})} + \frac{\sum_{i=1}^{m} t_{inputfilesize\_i}}{n \times vm_{bw\_j}} \tag{12}$$

$$C\_Time_{\max} = \frac{\sum_{\text{i}=1}^{m} t_{length\_i}}{n \times \min(vm_{comp\_j})} + \frac{\sum_{i=1}^{m} t_{inputfilesize\_i}}{n \times vm_{bw\_j}} \tag{13}$$

Define the Cost Constraint Function

The cost constraint function is defined as formula (14):

$$C\_Cost(I) = \frac{TotalCost(I) - TotalCost_{\min}}{TotalCost_{\max} - TotalCost_{\min}} \tag{14}$$

$TotalCost_{min}$ indicates that the user submits the task to perform the least cost on a virtual machine in parallel, and $TotalCost_{max}$ indicates that the task that the user submits is the most expensive to perform on a virtual machine in parallel. The calculation method is as formula (15), (16):

$$TotalCost_{\min} = C\_Time_{\min} \times MIN(UCost_j) \tag{15}$$

$$TotalCost_{\max} = C\_Time_{\max} \times MAX(UCost_j) \tag{16}$$

The Objective Function of Time and Cost

We use the linear weighting method to construct the objective function,

$$L = w_1 Time\_sf + w_2 C\_Cost \tag{17}$$

In the function, $w_1+w_2=1$, w_1 is the weight factor of time and w_2 is the weight factor of the cost. They are adjusted according to the actual situation. If $w_1=0$, $w_2= 1$, then the objective function becomes only the cost-related objective function. If $w_1= 0.5$, $w_2= 0.5$, then the final task completion time and the cost are equally important. If $w_1= 1$, $w_2= 0$, then the cost is ignored, only the final completion time as the optimization target.

MULTI - OBJECTIVE OPTIMIZATION OF CLOUD COMPUTING TASK SCHEDULING BASED ON IMPROVED ANT COLONY ALGORITHM

The ant colony optimization is a heuristic bionic algorithm proposed by Italian scholar M.Dorigo, which is mainly used to solve the optimal solution of combinatorial optimization problem. And cloud computing task scheduling problem has been proved as NP-Hard problem, so the ant colony algorithm used in cloud computing task scheduling can greatly improve the scheduling efficiency. In this paper, based on the study of the standard ant colony algorithm, it is applied to the cloud computing task scheduling to re-establish the model.

In the algorithm, each ant looks for the appropriate virtual machine for the corresponding task. The ants finally find an optimal matching scheme according to the optimized target. The ants find the optimal solution in parallel, and communicate with each other through information. We adjust the pheromone on the task with the virtual machine path, which affects the selection of the ant for the other ant and the next iteration for the ant. In this paper, the initial pheromone, heuristic function and pheromone update rule of ant colony algorithm are improved by combining the objective function to be optimized.

Initializing Pheromone

τ_0 is the initial value of the pheromone on the path between the task and the virtual machine. It is calculated as Equation (18):

$$\tau_0 = vm_{\text{comp\_j}} / vm_{avgcomp} \tag{18}$$

$vm_{avgcomp}$ is the average MIPS for the virtual machines. And it calculated as Equation (19):

$$vm_{avgcomp} = \frac{\sum_{j=1}^{n} vm_{comp\_j}}{m} \tag{19}$$

Vm Choosing Rule for Next Task

In this paper, we use ACS state transition rule [14].The state transition rule as follows:

$$s = \begin{cases} \arg\max_{j \in allowed_k} \{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}\}, & if \quad q<q_0 \\ S_1, & otherwise. \end{cases} \tag{20}$$

$$s_1 = p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}(t)]^{\beta}}{\sum_{j \in allowed_k} [\tau_{ij}(t)]^{\alpha} * [\eta_{ij}(t)]^{\beta}}, & if \quad j \in allowed_k \\ 0, others \end{cases} \tag{21}$$

Where is q a random number uniformly distributed in [0...1], q_0 is a parameter ($0 \le q_0 \le 1$). q_0 determines the relative importance between new knowledge utilization and exploration of a priori. s_1 is a random variable selected according to the probability distribution given in 21. The parameters α and β control the relative weight of the pheromone trail and the visibility information respectively. η_{ij} is the heuristic function that indicates that task i chooses the expected intensity of execution on the virtual machine j. The calculation method is as follows:

$$\eta_{ij} = Load_j \times \frac{1}{et_{ij}} \tag{22}$$

Where et_{ij} is defined in equation (3). In the case of not considering l, the smaller the execution time is, the larger the probability of selecting the current virtual machine for the task. $Load_j$ is the load function of the virtual machine. It is defined as formula (23):

$$Load_j = 1 - ((E_j - E_{avg}) / \sum_{J \in vm} E_J \tag{23}$$

Where E_j is the execution time of the virtual machine j and E_{avg} is the average execution time for all virtual machine execution times. When the value of E_j is large, then $Load_j$ will be relatively small, indicating that the load of virtual machine j is relatively large and the execution time of virtual machine j is longer. As a result the value of η_{ij} is relatively small, then the probability that the task i is assigned to the virtual machine j is small. Similarly, when E_j is small, the b value will be relatively large, indicating that the load of virtual machine j is relatively small and the execution time of virtual machine j is shorter. As a result the value of η_{ij} is relatively large, then the probability that the task i is assigned to the virtual machine j is large. This makes it possible to dynamically adjust the load on the system.

Pheromone Local Update Rules

When an ant matches the corresponding virtual machine for all tasks, the pheromone on the mapping path of the matching scheme is updated locally. The update rules are as follows:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t) \tag{24}$$

Where ρ is the pheromone volatilization factor, indicating the degree of volatilization of pheromones per unit time and $1-\rho$ indicates the degree of residual pheromone, $\rho \in (0,1]$. The greater the ρ, the faster the pheromone volatilizing, then the smaller the effect of the past search solution for the present solution. $\Delta\tau_{ij}(t)$ is the pheromone increment of the t time of the path between task i and virtual machine j. The calculation is as follows:

$$\Delta\tau_{ij}(t) = \begin{cases} \frac{1}{L}, & if\,(i,j) \in T^k(t) \\ 0, & otherwise \end{cases} \tag{25}$$

L is the value of the objective function for the ant to find a solution, which is defined by formula (17). $T^k(t)$ is the tour done by ant k at iteration t.

Pheromone Global Update Rule

When all the ants complete an iteration, the optimal matching scheme in this iteration is obtained, and the global optimal matching scheme is obtained by comparing with the global optimal matching scheme obtained before iteration. Then, we update the pheromone on the mapping path between the task and the virtual machine on the global optimal scheme. The update rules are as follows:

$$\tau_{ij}(t+1) = (1-\rho_1)\tau_{ij}(t) + \rho_1\Delta\tau_{ij}(t) \tag{26}$$

Where ρ_1 is the pheromone volatilization factor, $1-\rho_1$ is the degree of residual pheromone, $\rho_1 \in (0,1]$

$\Delta\tau_{ij}(t)$ is calculated as follows:

$$\Delta\tau_{ij}(t) = \begin{cases} \frac{1}{L_{best}}, & if\,(i,j) \in T^+, \\ 0, & otherwise \end{cases} \tag{27}$$

Where L_{best} is the optimal solution of the objective function from the beginning iteration to the current iteration, and its value is calculated by the formula (17).

Programming Steps of the proposed MO-ACO

- Step1. Initialize the pheromone .Set the maximum number of iterations, the pheromone energetic factor α, the expected heuristic factor β, the volatile factors ρ and ρ_1, the number of ants m, and p_0.
- Step2. Place all ants at the starting VMs randomly.
- Step3. Each ant calculates the probability of the current task selected on each virtual machine in the set of optional virtual machines based on the formula (20) (21). And then the ant chooses the matching *VM* for the current task according to the roulette method. And then add the selected *VM* to the taboo table.
- Step4. When an ant completes a solution, update the pheromone on the matching scheme path found by the ant according to formula (24), (25). Compare with the previous optimal solution and update the optimal solution.
- Step5. If all the ants end their tour, Nc = Nc + 1(Nc is the number of iterations), calculate the global optimal solution and update the pheromone on on the optimal solution path according to formula (26), (27); otherwise, repeat Step3.
- Step6. If the current number of iterations is less than the maximum number of iterations, Clear taboo table and return to Step2. Otherwise, end the iteration and output the best solution.

IMPLEMENTATION & EXPERIMENTAL RESULTS

In order to verify the effectiveness of the algorithm, this study has done a simulation experiment in the open source cloudsim cloud simulation platform. And compared with the ACO algorithm and Min-Min algorithm.

Parameters Setting of Cloud Simulator

The experiment is implemented with 2 Datacenters and 50-250 tasks under the simulation platform. The length of the task is from 5000 MI (Million Instructions) to 100000 MI. The parameters setting of cloud simulator are shown in Table 1.The numbers of virtual machine is 10.And every vm has 1-3 pe.

TABLE 1. Parameters setting of cloudsim

Entity Type	Parameters	values
Task (cloudlet)	Length of task	5000-100000
	Total numbers of task	50-250
	File size	300-5000
Virtual machine (vm)	MIPS of pe	512-1024
	Numbers of vm	10
	Number of pe per vm	1-3
	bandwidth	500-1200
	memory	512-2048
	Storage	100000-800000
	Unit cost of vm	1-10

Parameters setting of the basic ACO and MO-ACO

The parameters of the algorithm are referenced to the paper [15].At the same time we tested and compared the performance of 10 groups of different α, β and ρ parameters. Then we choose the best set of parameters as the parameters in experiments. The parameters' setting is shown in Table 2.

TABLE 2. Parameters setting of MO-ACO

Parameters	Values
α	1
β	5
ρ	0.4
ρ_1	0.5
Numbers of ant (m)	10
q_0	0.9
Maximum number of iterations($N_{\max}$)	30

Definition of Load Balancing Evaluation Function in Experiment

Experiments are based on the standard deviation formula of the execution time of the system virtual machine to evaluate the system load. The smaller the value is, the more balanced the load of the whole system is as follows:

$$BL = \sqrt{\frac{\sum_{i=1}^{m} (LB_{\mathrm{i}} - LB_{\mathrm{aver}})^2}{m}} \tag{28}$$

Where LB_{aver} is the average execution time of the virtual machine, which is calculated as follows:

$$LB_{aver} = \frac{\sum_{i=1}^{m} LB_i}{m} \tag{29}$$

LB_i is the total time of execution of virtual machine i , m is the number of virtual machines.

Experimental results

In the cloudsim, set the same environmental parameters as the premise, we simulate the Min-Min, ACO algorithm and MO-ACO algorithm respectively.

In the three aspects of makespan, cost and load balance, three algorithms have been compared respectively. The experimental results are shown in Figures 2 to 4.

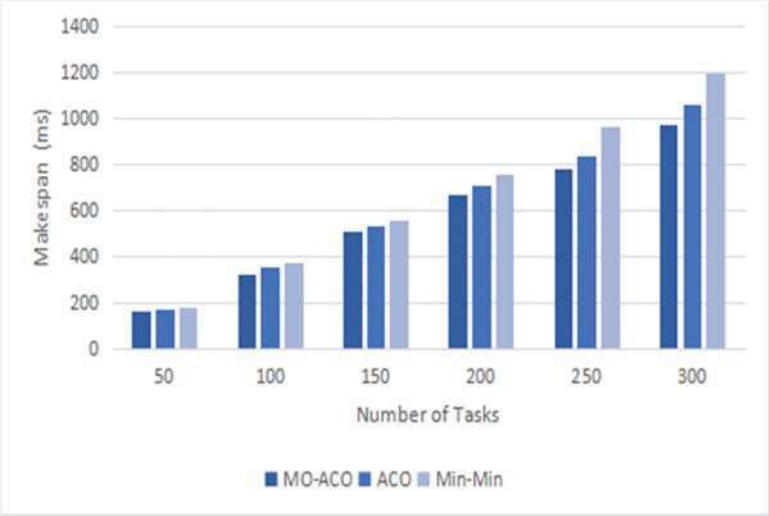

FIGURE 2. The makespan

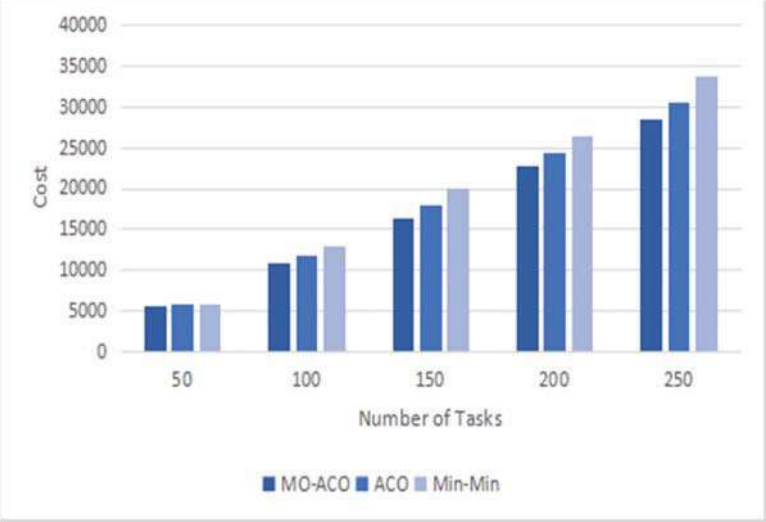

FIGURE 3. The total costs.

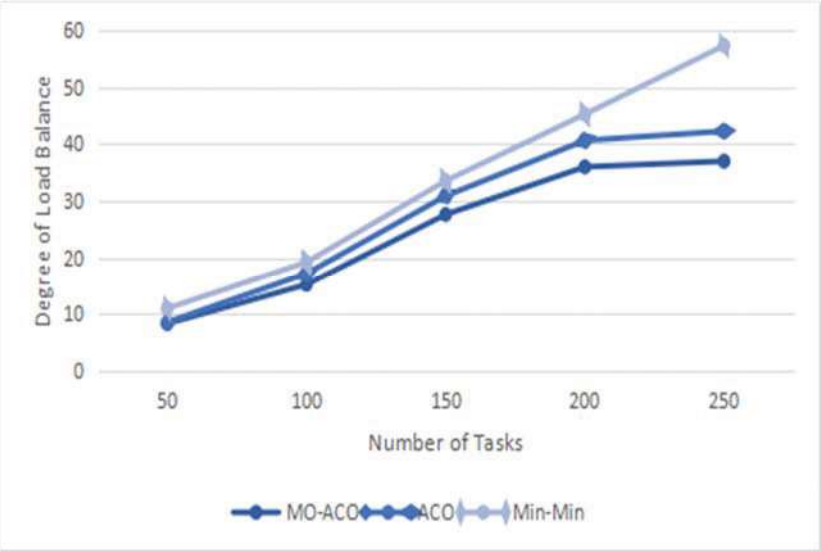

FIGURE 4. Average degree of load balance

Figure 2 shows the Turing results of the MO-ACO algorithm, the ACO algorithm, and the Min-Min algorithm in the same experimental environment. As can be seen from Figure 2, when the number of tasks is small, the makespan of the tasks set of the three algorithm is relatively close. With the increase in the number of tasks, the makespan of MO-ACO algorithm and ACO algorithm is significantly better than Min-Min. This is because the MO-ACO algorithm and the ACO algorithm have a better performance in solving the combinatorial optimization problem. At same time we can see that the makespan of the TCL-ACO algorithm is slightly better than that of the basic ACO algorithm.

Figure 3 shows the total costs of the task for the three algorithms. From the figure we can see that when the number of tasks is less, the three tasks of the total implementation of the cost difference is not very obvious. However, with the increase in the number of tasks, MO-ACO algorithm task execution costs is significantly lower than the other two algorithms. The reason is that the MO-ACO algorithm adjusts the corresponding pheromone through the objective function that includes the cost consideration to guide the ant and obtain a better solution with the costs as the constraint target.

Figure 4 shows the load balancing of the three algorithms. In the same number of tasks, we compared the three algorithms, it can be found that the value of load balancing of system of the MO-ACO algorithm is smaller, the system load more balanced. This is because the load balancing function is established in the MO-ACO algorithm and is well applied to the heuristic function, which affects the calculation of the transfer probability formula. The virtual machine with heavy load in the system is less likely to be selected.

CONCLUSIONS AND FUTURE WORK

In this paper we have proposed the MO-ACO algorithm, which considers the makespan, cost and load balancing. The algorithm establishes the constraint function of task completion time, cost and load, improves the basic ant colony algorithm heuristic function and pheromone update rule, and adopts the pseudo - random transition probability rule in ant colony system. We experimented with the algorithm in the cloudim, and the experiment proved the effectiveness of the algorithm. In practice, cloud computing is more complex. In future research, we will consider the dependency between tasks, increase the number of tasks in the experiment, consider the factors such as customer satisfaction.

ACKNOWLEDGMENTS

This work was financially supported by the Commission of Science and Technology Research Projects of Chong qing, No.KJ130533.

REFERENCES

1. Travis J.KringJ, Qiao Ruiping. LabVIEW University Practical Course [M]. Beijing: Electronic Industry Press, 2008.
2. Delimitrou, Christina, and Christos Kozyrakis. "Qos-aware scheduling in heterogeneous datacenters with paragon." ACM Transactions on Computer Systems (TOCS) 31.4 (2013): 12.
3. Huang, Qi-yi, and Ting-lei Huang. "An optimistic job scheduling strategy based on QoS for Cloud Computing." Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference on. IEEE, 2010.pp.673-675
4. Chang, Fangzhe, Jennifer Ren, and Ramesh Viswanathan. "Optimal resource allocation for batch testing." Software Testing Verification and Validation, 2009. ICST'09. International Conference on. IEEE, 2009.pp.91-100
5. Dorigo, Marco, and Luca Maria Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem." IEEE Transactions on evolutionary computation 1.1 (1997): 53-66.
6. Salari, E., and K. Eshghi. "An ACO algorithm for graph coloring problem." Computational Intelligence Methods and Applications, 2005 ICSC Congress on. IEEE.
7. Zhang, Xiaoxia, and Lixin Tang. "CT-ACO-hybridizing ant colony optimization with cyclic transfer search for the vehicle routing problem." Computational Intelligence Methods and Applications, 2005 ICSC Congress on. IEEE, 2005.
8. ZHANG Huan-qing, ZHANG Xue-ping, WANG Hai-tao, et al.Spatial Scheduling Task Scheduling Based on Load Balancing Ant Colony Optimization Algorithm [J]. Microelectronics and Computer, 2015 (5): 31-35.
9. FENG Liangliang, et al. "Task Scheduling Algorithm Based on Improved Particle Swarm Optimization in Cloud Computing Environment." [J].Computer Engineering, 2013, 39 (5): 183-186,191.
10. SUN Da-wei, CHANG Gui-ran, LI Feng-yun, et al. Optimization algorithm of multi-dimensional QoS resource scheduling based on immune clone [J] .Acta Electronic Journal, 2011, 39 (8): 1824-1831.
11. Babukarthik R G, Raju R, Dhavachelvan P. Hybrid algorithm for job scheduling: Combining the benefits of ACO and Cuckoo search [M]//Advances in Computing and Information Technology. Springer Berlin Heidelberg, 2013: 479-490.
12. Zhu, Qian, and Gagan Agrawal. "Resource provisioning with budget constraints for adaptive applications in cloud environments." Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. ACM, 2010.
13. Panda, Sanjaya K., and Prasanta K. Jana. "A multi-objective task scheduling algorithm for heterogeneous multi-cloud environment." Electronic Design, Computer Networks & Automated Verification (EDCAV), 2015 International Conference on. IEEE, 2015.
14. Dorigo, Marco, and Luca Maria Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem." IEEE Transactions on evolutionary computation 1.1 (1997): 53-66.
15. Duan, Haibin, Guanjun Ma, and Senqi Liu. "Experimental study of the adjustable parameters in basic ant colony optimization algorithm." Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. IEEE, 2007.