Research on Cloud Task Scheduling Based on Load Balancing Ant Colony Optimization

Hai-tao HU and Xiao-rong LUO*

School of Control and Computer Engineering North China Electric Power University Beijing, China
*Corresponding author

Keywords: Cloud computing, Task scheduling, Ant colony optimization, Load balancing.

Abstract. For the problem of resource load unevenness of task scheduling in cloud computing, this paper proposes Load Balancing Ant Colony Optimization (LBACO) based on load conditions of virtual machines. At the same time, the pheromone initialization, heuristic function, state transition probability, and pheromone updating method in the ant colony algorithm (ACO) are improved. The improved scheduling strategy was experimented on the CloudSim platform. The experimental results show that compared with ACO and RR algorithms, the LBACO algorithm achieves better performance in terms of makespan of task sets and the system load balance.

Introduction

In recent years, cloud computing[1] has become a new platform for modern distributed computing environments, which helps provide computing resources for users over the Internet. Cloud computing is a model for ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or Service provider interaction[2].

With the continuous expansion of the scale of cloud computing, traditional scheduling algorithms have failed to meet the needs of clouds, many scholars have begun to study the application of intelligent algorithms to task scheduling. [3] proposed Multi-Objective Particle Swarm Optimization (MOPSO) to reduce energy consumption and makespan. [4] proposed a parallel genetic algorithm (PGA) to improve the convergence speed of the algorithm and provide an optimal solution to the cloud scheduling problem.

Ant Colony Optimization (ACO)[5,6] is a classical algorithm for solving combinatorial optimization problems and is effective for solving cloud computing task scheduling problems. However, although the ant colony algorithm can shorten makespan, it does not consider the problem of system load balancing, and the algorithm may fall into a local optimal solution. The purpose of load balancing is to improve resource utilization while reducing makespan. Inappropriate task scheduling strategy may make the load between virtual machines unbalanced[7].

To solve above problems, this paper proposes Load Balancing Ant Colony Optimization (LBACO) algorithm. Experimental results compared to ACO and Round-Robin (RR) algorithm show that the LBACO algorithm satisfies expectation.

The rest of this paper is organized as follows. Section 2 introduces the task scheduling model. The implementation of load balancing mechanism is presented in Section 3. Section 4 introduces the proposed LBACO algorithm in detail. Section 5 presents the simulation results. Finally, Section 6 concludes this paper.

Task Scheduling Model

The essence of task scheduling in cloud computing is to allocate tasks to virtual machines without violating Service Level Agreement (SLA)[8] to improve operating efficiency and resource utilization rate. In order to simplify the process of task scheduling, following assumptions are made[9-10].
The tasks are independent of each other and there is no dependency before or after. The tasks cannot be interrupted or migrated to other virtual machines during execution. The computing capability of each resource in the datacenter is known.

The task set $T = \{T_1, T_2, ..., T_m\}$, $m$ represents the number of tasks submitted by the user. The virtual machine set $VM = \{VM_1, VM_2, ..., VM_n\}$, $n$ represents the number of virtual machines.

The expected execution time of the $i$-th task $T_i$ on the $j$-th virtual machine $VM_j$ is represented by a matrix $E = \{E_{ij}\}_{0<i<m, 0<j<n}$, $E_{ij} = \frac{\text{length}_i}{\text{mips}_j} + \frac{\text{filesize}_i}{\text{bw}_j}$. Where $\text{length}_i$ represents the length of task $T_i$, $\text{filesize}_i$ represents the size of task $T_i$, and $\text{mips}_j$, $\text{bw}_j$ refer to the calculation capability and bandwidth of the virtual machine $VM_j$. The total time spent on $VM_j$ to execute the tasks can be expressed by $E_j$, which is defined as follows:

$$ E_j = \sum_{T \in Task_j} E_{ij} $$

$Task_j$ is a set of tasks that are executed on the virtual machine $VM_j$.

**Load Balancing Mechanism**

An efficient load balancing solution requires potentially reducing over-provisioning of resources. In order to make better use of resources, a load balancing mechanism is introduced to avoid resources overload or underload. According to the load condition of the virtual machine, this paper proposes load balance factor $Load_j$ to represent the load of virtual machine $VM_j$.

$$ Load_j = 1 - \frac{(E_j - E_{avg})}{(E_{max} - E_{min})}. $$

Where $E_{avg}$, $E_{max}$, $E_{min}$ represent the average execution time of $n$ virtual machines, the maximum execution time of virtual machines in $n$ virtual machines, and the shortest execution time.

$Load_j$ is inversely proportional to $E_j$ and decreases as $E_j$ increases. When $E_j < E_{avg}$, the value of $Load_j$ is greater than 1, indicating that the resources on the virtual machine $VM_j$ are free, and the task may be preferentially allocated to the $VM_j$ to improve resource utilization. When $E_j > E_{avg}$, the value of $Load_j$ is less than 1, indicating that this virtual machine is overload. If you continue to allocate tasks to $VM_j$, it may cause load imbalance and reduce the system operating efficiency.

**Load Balancing Ant Colony Optimization**

**Problem Description**

All ants are randomly placed on the task nodes in initial state. In an iteration, ants select the suitable virtual machines until all the tasks are assigned, which generates an allocation scheme\(^{[11]}\). The number of iterations is represented by $NC$, $1 < NC < NC_{max}$, $NC_{max}$ is the maximum number of iterations the algorithm runs. Each ant has a task taboo $Tabu$. After the task is assigned, it is added to $Tabu$, indicating this task is no longer accessed.

**Initializing Pheromone**

$\tau_{ij}$ represents the pheromone concentration of the task $T_i$ transmitted to the virtual machine $VM_j$. The initial pheromone is represented by $\tau_0$, $n$ is the number of virtual machines, $\tau_0$ is $E_{ij}/n$

**State Transition Probability Based on Load Balancing**

When an ant selects a virtual machine for a task, it not only considers the execution performance of the virtual machine, but also considers the load condition, and applies the load balancing factor $Load_j$ to the state transition probability $P_{ij}$ to avoid resource overload or underload. The probability that ant $k$ selects virtual machine $VM_j$ for task $T_i$ is calculated as Eq.(2).
\[ P^i_q(t) = \begin{cases} \left[ \tau_i(t) \right]^\alpha \left[ \eta_i(t) \right]^\beta, & i \in \text{allowed}_k \\ \sum_{i \neq j} \left[ \tau_i(t) \right]^\alpha \left[ \eta_i(t) \right]^\beta, & \text{others} \end{cases} \]

\[ \text{allowed} = \{0, 1, 2, \cdots, m-1\} - \text{Tabu} \]

\[ \eta_i = \frac{1}{E_i} \cdot \text{Load}_j \]

Where \( \text{allowed}_k \) indicates that the tasks have not yet been assigned, heuristic function \( \eta_{ij} \) indicates the desired degree of assignment of the task \( T_i \) to the virtual machine \( VM_j \), \( \alpha \) is a pheromone factor and \( \beta \) is a heuristic expectation factor.

**Pheromone Update Rule**

**Pheromone Local Update Rule.** After an ant completes an iteration, it matches corresponding virtual machine for all tasks, the pheromone is update locally. The update rule is defined as follows:

\[ \tau_i(t+1) = (1-\rho)\tau_i(t) + \rho \cdot \Delta \tau_i(t) \]

\[ \Delta \tau_i(t) = \sum_k \Delta \tau_{ij}^k(t) \]

Where \( \rho \) is the pheromone volatilization factor that represents the degree of pheromone volatilization per unit time. The larger \( \rho \) is, the faster the pheromone volatilizes, \( \rho \in (0, 1) \). \( \Delta \tau_{ij}(t) \) represents the pheromone increment of the path between task \( T_i \) and virtual machine \( VM_j \) during \( t \) time. \( \Delta \tau_{ij}^k(t) \) is the pheromone increment of the path between task \( T_i \) and virtual machine \( VM_j \) generated by ant \( k \), which is given by Eq.(7).

\[ \Delta \tau_{ij}^k(t) = \begin{cases} \frac{Q}{L^k(t)}, & \text{if } (i, j) \in T^k(t) \\ 0, & \text{others} \end{cases} \]

Where \( Q \) is a constant, \( T^k(t) \) is the allocation scheme that ant \( k \) finishes in this iterative iteration, and \( L^k(t) \) is its length (expected execution time), it is defined as follows:

\[ L^k(t) = \max_{j \in J} \left[ \sum_{i \in I_j}(E_i) \right] \]

Where \( I_j \) refers to the set of tasks assigned to virtual machine \( VM_j \).

**Pheromone Global Update Rule.** After all ants have completed an iteration, the optimal matching scheme in this iteration is obtained. we update the pheromone on the mapping path between the task and the virtual machine on the global optimal scheme. The global update formula is:

\[ \tau_i(t+1) = (1-\rho)\tau_i(t) + \frac{Q}{L_{\text{best}}(t)}, \text{ if } (i, j) \in T_{\text{best}}(t) \]

\[ L_{\text{best}}(t) = \min[L^k(t)] \]

Where \( Q \) is a constant, \( T_{\text{best}}(t) \) is the optimal path from the beginning iteration to the current iteration and its value is calculated by \( L_{\text{best}}(t) \).
Experimental Design and Analysis

In order to verify the effectiveness of LBACO algorithm, the performance of LBACO algorithm is evaluated under CloudSim platform [12] and compared with the ACO and RR algorithms.

Parameters Setting of CloudSim and LBACO

The experiment is implemented with one Datacenter with 10 VMs and 50-300 tasks under the simulation platform. The parameters setting of CloudSim are shown in Table 1.

Several values for each parameter are evaluated while all the others were held constant on 150 tasks. The default value $\alpha=1$, $\beta=1$, $\rho=0.5$, the number of ants $k=10$ and the maximum number of iterations $N_{C_{\text{max}}}=100$. Only one of the parameter values is changed in each experiment. The parameter values tested were $\alpha \in \{0.5,1,1.5,2,2.5,3\}$, $\beta \in \{1,2,3,4,5,6\}$, $\rho \in \{0.3,0.4,0.5,0.6,0.7\}$. Determining the optimal value of each parameter through experiments, the best parameter values are shown in Table 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>length</td>
<td>1000~20000</td>
<td>$\alpha$</td>
<td>2.5</td>
</tr>
<tr>
<td>filesize</td>
<td>200~2500</td>
<td>$\beta$</td>
<td>6</td>
</tr>
<tr>
<td>m</td>
<td>50~300</td>
<td>$\rho$</td>
<td>0.6</td>
</tr>
<tr>
<td>Virtual</td>
<td></td>
<td>$Q$</td>
<td>125</td>
</tr>
<tr>
<td>Machine</td>
<td></td>
<td>$k$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$N_{C_{\text{max}}}$</td>
<td>100</td>
</tr>
</tbody>
</table>

Experimental Results and Analysis

In the following experiments, We compared the makespan of different number of task sets. Figure 1.(a) shows the makespan of task sets of the LBACO, ACO, and RR algorithms. The makespan of LBACO is reduced by 42% to 49% compared to RR, and is 23% to 30% lower than ACO. With the increase of the number of tasks, the makespan of the tasks of LBACO is significantly better than ACO and RR algorithms. That is because LBACO have a better performance in solving task scheduling problem.

The degree of imbalance ($DI$) to evaluate the load balancing of the three algorithms after the task scheduling is completed. $DI$ is defined as $(E_{\text{max}}-E_{\text{min}})/E_{\text{avg}}$.

The value of $DI$ is smaller, the system load is more balanced. The values of $DI$ of the three algorithms when processing 50 to 300 tasks are shown in Figure 1.(b). The results show the value of
The DI of the LBACO is 1.4 to 1.8 lower than the RR, and is 0.4 to 0.9 lower than ACO. It can be seen that the LBACO can achieve better system load balance than RR and ACO algorithms.

Conclusion
This paper proposed the LBACO algorithm to shorten the makespan of task scheduling with load balancing and improve resource utilization in cloud environment. We have experimentally evaluated the performance of LBACO algorithm in applications with the number of tasks varying from 50 to 300. The simulation results show that the LBACO algorithm outperforms ACO and RR algorithm in makespan and the entire system load balance.

In this work, the tasks are assumed independently of each other, and the order of execution is no dependency before or after. However, there is precedence constraint between tasks in practical applications. In future work, the priority relationship between tasks will be considered.

Acknowledgments
This work was supported by the National Natural Science Foundation of China, No.61372182.

References