International Journal of Hybrid Information Technology
Vol.8, No.6 (2015), pp.205-216
http://dx.doi.org/10.14257/ijhit.2015.8.6.20

A Priority-driven ACO Algorithm for DAG Task Scheduling in Cloud Environment

Haitao Xie, Hongwei Chen and Chunzhi Wang

School of Computer Science, Hubei University of Technology, Wuhan, China
markxie@163.com

Abstract

Task scheduling in cloud environment is a key technical problem on how to allocate available cloud resources to cloud users. Usually, a task in cloud environment can be cut into a serials of subtasks, which have precedence and dependency relationships among themselves. We describe the problem by means of a DAG (Directed Acyclic Graph) model. Then, we propose a priority algorithm for DAG task scheduling, and a priority-driven ACO (Ant Colony Optimization) algorithm for DAG task scheduling on the basis of the DAG model. Finally, we compare these two algorithms with the greedy algorithm through simulation in the CloudSim platform. The simulation results show that the priority-driven ACO algorithm is effective to solve DAG task scheduling problem in cloud environment.

Keywords: *Cloud Computing, DAG, Task Scheduling, ACO, Priority, Virtual Machine*

1. Introduction

How to allocate available resources to cloud users is a pivotal study issue on cloud computing [1]. In the actual cloud environment, most work of the task scheduling is concentrated in the dependency-based task optimization. When a large task is performed in cloud environment, the task can be split in a serial of interdependent subtasks, then these subtasks will be allocated to different virtual machines. Usually, subtasks with dependencies can be expressed by DAG (Directed Acyclic Graph). Compared with the independent task scheduling problem, the task scheduling problem with dependent relationship needs solve the problem on assignments of subtasks, at the same time it also needs consideration on the time of subtasks allocated to virtual machines because the precursor subtask scheduling will have a big effect on the subsequent subtasks. So, the task scheduling issue with dependent relationship is very complicated.

Scholars started to study DAG task scheduling problem in the last century. Figure 1 is a classification diagram of DAG task scheduling algorithms, and the algorithms falls into the following two categories: (1) The performance-based priority algorithm. This kind of algorithm gives priority attention to the task completion time, and it can be divided into the meta-heuristic algorithm and the heuristic algorithm. The meta-heuristic algorithm mainly includes genetic algorithm and PSO (Particle Swarm Optimization) algorithm. The literature [2] utilized genetic algorithm to solve DAG task scheduling problem in the heterogeneous system; and literatures [3, 4] used PSO algorithm to optimize task scheduling problem in cloud environment. The heuristic algorithm can be classified into cluster scheduling algorithm [5, 6], single task scheduling algorithm, hierarchical scheduling algorithm [7], and table-driven scheduling algorithm. (2) The QoS-based algorithm. This kind of algorithm needs to consider different kinds of resource constraints such as time constraint, budget constraint and so on. The literature [9] puts forward a heuristic algorithm to solve the complicated task scheduling problem with dependency relationships and QoS demands. In addition, literatures [10-12] began to study the DAG task scheduling problem in cloud environment.

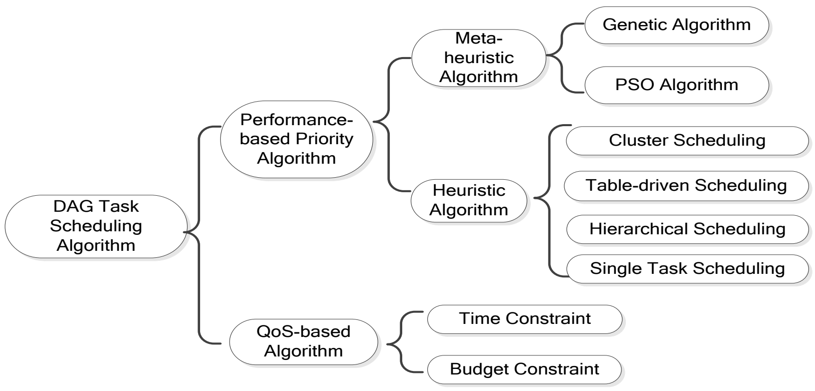

Figure 1. The Architecture of ATN on SLA in Cloud Environment

A series of experimental results show that ant colony algorithm has strong ability to find good solution and it is not easy to fall into local optimum, because the algorithm takes advantage of the positive feedback principle to accelerate the evolution process to a certain extent, and this is a kind of parallel algorithm in nature to find a better solution by ongoing communication and transmission among individuals. Literatures [13-15] began to study the application of cloud computing based on the ACO algorithm.

In the paper, we focus on how to optimize DAG task scheduling problems in cloud environment by improved ACO algorithm. This paper is organized as follows: In Section 2, we define and describe the DAG task scheduling model in cloud environment. Then, two different algorithms are provided to solve the DAG task scheduling problem in cloud environment. We propose a priority algorithm for DAG task scheduling in Section 3, and a priority-driven ACO (Ant Colony Optimization) algorithm for DAG task scheduling in Section 4 on the basis of the DAG model. In Section 5, we compare these two algorithms with the greedy algorithm through simulation in the CloudSim platform. Finally, Section 6 concludes the work.

2. The DAG Task Scheduling Model in Cloud Environment

We adopt DAG to describe a series of interdependent subtasks in this paper. It is a NP-hard problem to allocate these subtasks to appropriate virtual machines to achieve the goal such as the shortest scheduling time, minimum cost and high throughput. We will put forward different scheduling algorithms in cloud environment to get an approximate optimal solution.

A DAG workflow is a large task which can be divided into a series of subtasks. The two-tuples $G(V,E)$ can describe a DAG workflow, there into $V=\{V_1,V_2,V_3...V_k \mid k\in Z, k<\infty\}$ represents nodes or subtasks, and $E=\{(V_i,V_j)\mid V_i,V_j\in V\}$ represents the partial ordering relation of nodes or subtasks. In a DAG workflow, an ingress node is a node without any parent nodes, and an egress node is a node without any child nodes.

When a task is divided into a series of subtasks, communication delays among subtasks can take up a certain amount of time and increase system overhead. So communication delays among subtasks need to be taken into account. In a DAG workflow, the weigh of each side represents the communication time between the corresponding subtasks. For example in Figure 2, the communication time between the subtask A and B is 1.04.

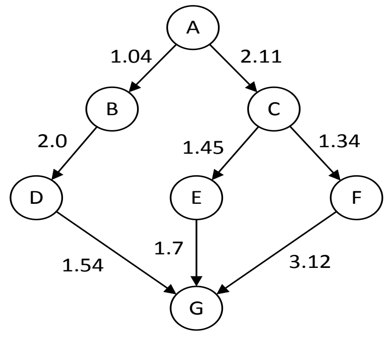

Figure 2. Diagram of a Paradigmatic DAG

Assumed that the length of the subtask V_i is $L_i(i=1,2,...)$, and the processing capability of the virtual machine Vm_j is $P_j(j=1,2,3...)$, then the execution time on the subtask V_i allocated to the virtual machine Vm_j is:

$$t(V_i, Vm_j) = \frac{L_i}{P_j} \tag{1}$$

So the average execution time on the subtask V_i allocated to all virtual machines is:

$$\bar{t} = \sum_{j=1}^{n} \frac{t(V_i, Vm_j)}{n} \tag{2}$$

Thus the average execution time on all subtasks allocated to all virtual machines is:

$$\overline{T} = \sum_{k=1}^{s} \frac{\overline{t_k}}{s} = \sum_{i=1}^{s}\sum_{j=1}^{n} \frac{t(V_i, Vm_j)}{n \times s} \tag{3}$$

In formula 3, s represents the number of subtasks, and n is the number of virtual machines. As a result of subtasks with dependencies, subtasks can be layered. If the scheduling algorithm ensures that subtasks in each layer have enough concurrent degree, which means subtasks in each layer can run in the corresponding virtual machine almost at the same time, then the average completion time of each layer is about equals to $\overline{T}$. The total scheduling time of a DAG workflow involves from start allocation of ingress nodes to end running of egress nodes, so the average total scheduling time is:

$$E(T) = D \times \overline{T} = D \times \sum_{i=1}^{s}\sum_{j=1}^{n} \frac{t(V_i, Vm_j)}{n \times s} \tag{4}$$

In formula 4, D represents the depth of subtasks in a DAG workflow. For example, the depth of subtasks in figure 2 is 4. Supposed that the length of all subtasks are same, and the capability of all virtual machines are same, then the total scheduling time is equals to $E(T)$, and the execution time of each path in DAG workflow is also equivalent. Supposed that the longest path in the DAG workflow is the critical path CP, then the total scheduling time is actually the execution time of critical path CP, so the scheduling time FT can be defined as:

$$FT = \sum_{i=1}^{n} t(V_i, Vms) \ \ \{V_i \in CP\} \tag{5}$$

In formula 5, n is the number of subtasks in the critical path CP, and Vms are the corresponding virtual machines allocated to subtasks in the critical path CP.

3. The Priority Algorithm for DAG Task Scheduling

The priority algorithm for DAG task scheduling aims to calculate the priority of each subtask through certain strategy, then distributes subtasks to the corresponding virtual machines according to the priority of subtasks. The priority of subtasks is defined in terms of the average completion time and the weight value of the DAG workflow.

$$p_i = \overline{t_i} + \max\{p_j + w_{ij} \mid j \in i\_next\} \tag{6}$$

In formula 6, $\bar{t_i}$ represents the average execution time on the subtask V_i allocated to the corresponding virtual machines in the DAG, and i_next represents the subsequent subtask set of the subtask V_i, and w_{ij} represents the weight value of the edge (V_i, V_j) in the DAG. It is important to note that the egress node in the DAG workflow has no corresponding child nodes, so its priority is:

$$p_i = \bar{t_i} = \sum_{j=1}^{n} \frac{t(V_i, Vm_j)}{n} \tag{7}$$

Supposed that there exists a DAG workflow like Figure 2, and the length of each subtask is shown in Table 1.

Table 1. The Length of Subtasks

Name of the subtask	Length	Name of the subtask	Length
A	30218	E	40325
B	44157	F	29856
C	25698	G	32568
D	31562		

Moreover, supposed that there are 3 virtual machines in the system, and the processing capability of virtual machines is shown in Table 2.

Table 2. The Processing Capability of Virtual Machines

Name of the virtual machine	Capability(MIPS)
1	3000
2	3500
3	4000

According to the above data, we can calculate the execution time on each subtask allocated to all virtual machines and the priority of each subtask. The information is shown in Table 3.

Table 3. The Execution Time in a Virtual Machine and Priority of Each Subtask

Task	Vm_1	Vm_2	Vm_3	Average value	Priority
A	10.07	8.63	7.55	8.75	470
B	172	12.62	11.04	12.79	391
C	8.57	7.34	6.42	7.44	31.71
D	10.52	9.02	7.89	9.14	20.12
E	13.44	11.52	10.08	11.68	22.82
F	9.95	8.53	7.46	8.65	21.21
G	10.86	9.31	8.14	9.44	9.44

After calculating the priority of each subtask, the algorithm will allocate these subtasks to the corresponding virtual machines. The flow diagram of the algorithm is shown in figure 3. The explanation of the algorithm is as follows.

(1) Calculate the priority of each subtask according to formula 6 and formula 7.

(2) Determine the scheduling sequence according to the priority.

(3) Take the subtask from the DAG workflow in descending order. The subtask with highest priority will be allocated to the available virtual machine with powerful processing capability. If all virtual machines are busy, the subtask will be allocated to the

first idle virtual machine. The algorithm can calculate the actual start time of the subtask *BT* before it is scheduled, and compare the actual completion time of subtasks with the higher priority *CT*. If a subtask with higher priority meets the condition *CT*>*BT*, the algorithm sets virtual machines scheduling the subtask in the busy state. If the number of subtasks with higher priority which meets the condition *CT*>*BT* is more than the number of virtual machines, the algorithm can calculate the first idle virtual machine through *CT*.

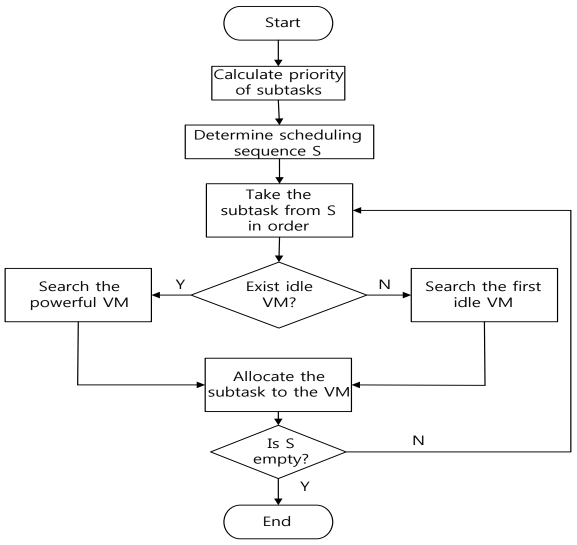
Start
Calculate priority of subtasks
Determine scheduling sequence S
Take the subtask from S in order
Search the powerful VM
Y
Exist idle VM?
N
Search the first idle VM
Allocate the subtask to the VM
Is S empty?
N
Y
End

Figure 3. The Flow Diagram of Priority Algorithm for DAG Task Scheduling

The pseudocode of the algorithm is shown in Table 4.

Table 4. Pseudocode of Priority Algorithm for DAG Task Scheduling

Priority algorithm for DAG task scheduling

```
input the DAG workflow $G(V,E)$
initialize the parameters of $V_i \in V$ and $Vms$
calculate priority of $V_i$ incrementally
sort $V_i$ in ascending order, and save $V_i$ to List $L$
for node $V_i \in L$ do
    if $V_i \rightarrow$ parent$\rightarrow$ state == $Finished$
        for each $Vm_i \in Vms$ do
            if $Vm_i \rightarrow$status == $idle$
                save $Vm_i$ to set $S$
            end if
            choose the powerful $Vm_j$ in $S$
            let $V_i \rightarrow Vm_j$
        end for
    end if
end for
calculate total schedule time
```

According to the above allocation strategy, the allocation process of each subtask in Figure 2 is as follows:

(1) Calculate the priority of each subtask, and getting the scheduling sequence in descending order ABCEFDG.

(2) Take the subtask A from the scheduling sequence, then it is allocated to the VM 3, which is the idle virtual machine with the powerful processing capability;

(3) Take out subtasks B and C, then the subtask B is allocated to the VM 3 due to its higher priority, and the subtask C is allocated to the VM 2;

(4) Take out the subtask E, and it is allocated to the VM 2 because the subtask C will be finished before the subtask B, and the VM 1 and VM 2 are available for the subtask E;

(5) Take out the subtask F, and it is allocated to the VM 2 because only VM 2 is idle;

(6) Take out the subtask D, and it is allocated to the VM 3 because only VM 3 is idle;

(7) Take out the subtask G, and it is allocated to the VM 3 because all virtual machines are idle.

Assumed that A → 1 represents the subtask A is allocated to VM 1. From the above scheduling process, the final scheduling result in figure 2 is A → 3, B → 3, C → 2, D → 3, E → 2, F → 1, G → 3. Seen from the scheduling result, the VM 3 with powerful processing capability has processed four subtasks, while VM 2 has processed two subtasks and VM 1 has processed only one subtask. The advantage of the priority algorithm for DAG task scheduling is to reduce or avoid busy waiting time of subtasks effectively. However, the algorithm can not solve the problem of load balancing effectively.

4. The Priority-driven ACO Algorithm for DAG Task Scheduling

In order to further enhance the performance of the above algorithm, we put forward the priority-driven ACO algorithm for DAG task scheduling. The algorithm incorporates the advantage of priority algorithm, and considers the priority of subtasks and the state of virtual machines when allocating virtual machines to subtasks. If a subtask can be allocated to multiple virtual machine, each scheduling scheme may make the overall scheduling time much difference. So, the algorithm defines *dc* as difference between the longest execution time and the shortest execution time when a subtask is allocated to a virtual machine.

$$dc = \max\{t(V_i, Vm_j)\} - \min\{t(v_i, Vm_j)\} \tag{8}$$

In formula 8, $t(V_i,Vm_j)$ represents the execution time on the subtask V_i allocated to the virtual machine Vm_j. The priority of the subtask is defined as:

$$p_i = (\bar{t}_i + \max\{p_j + w_{ij} \mid j \in i\_next\}) \times a + dc \times b \tag{9}$$

In formula 9, $\bar{t}_i$ represents the average execution time on the subtask V_i allocated to the corresponding virtual machines in the DAG, and i_next represents the subsequent subtask set of the subtask V_i, and w_{ij} represents the weight value of the *edge* (V_i, V_j) in the DAG. *a* and *b* are elasticity coefficients, and they represent relative important degree between average execution time and *dc* in task scheduling, and 0<*a*<1,0<*b*<1,*a*+*b*=1. As shown in table 5, supposed that *a*=0.7 and *b*=0.3, we can calculate the average execution time, *dc* and priority of each subtask from Figure 2.

Table 5. Priority of Subtasks (*a*=0.7,*b*=0.3)

Name of subtasks	Average execution time	*dc*	Priority
A	8.75	2.52	32.05
B	12.79	3.68	25.54
C	7.44	2.15	22.84
D	9.14	2.63	187
E	11.68	3.36	16.98
F	8.65	2.49	15.59
G	9.44	2.72	7.42

From the above Table, we can determine the scheduling sequence in descending order

ABCEFDG. After determining the scheduling sequence, ants begin traveling around to find the solution. If ants find the solution, they would update pheromone. Then the matching probability of a virtual machine in the moving process of ants is:

$$pb_i = \frac{[phr(i,j)]^{\alpha}[ins(i,j)]^{\beta}}{\sum_{j=1}^{n}[phr(i,j)]^{\alpha}[ins(i,j)]^{\beta}} \tag{10}$$

In formula 10, $ins(i, j)$ represents the earliest heuristic value on the subtask V_i allocated to the virtual machine j, and it can be defined as follows:

$$ins(i,j) = t(V_i, Vm_j) + \max_{k \in i\_next}\{t(V_k, Vm_j) + w_{ik}\} + EST_{ij} \tag{11}$$

In formula 11, EST_{ij} represents the earliest execution time on the subtask V_i allocated to the virtual machine j, and $phr(i, j)$ represents pheromone values, which can be recorded by a matrix. The matrix records the pheromone information about each subtask to each virtual machine, and will be initialized before the iteration of the ant colony. After ants find a solution, the matrix will update the pheromone.

The flow diagram of priority-driven ACO algorithm for DAG task scheduling is shown in Figure 4, and the scheduling process of the algorithm is as the following:

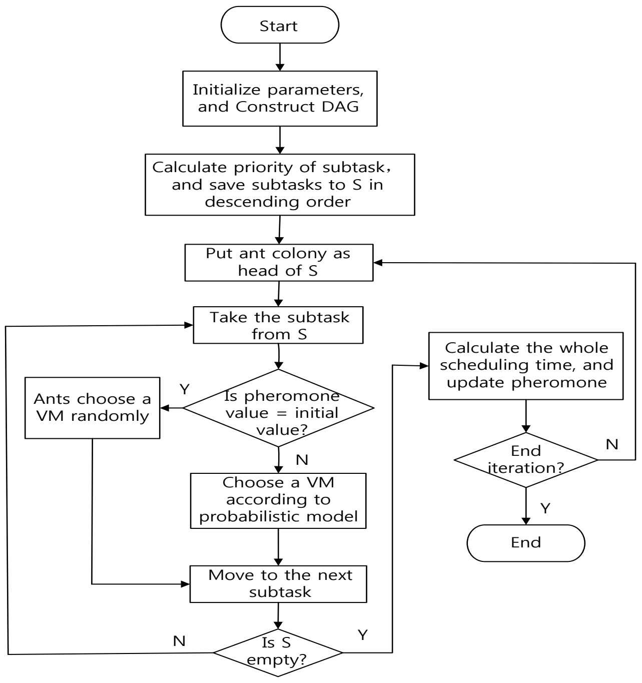
Start
Initialize parameters, and Construct DAG
Calculate priority of subtask, and save subtasks to S in descending order
Put ant colony as head of S
Take the subtask from S
Is pheromone value = initial value?
Y
Ants choose a VM randomly
N
Choose a VM according to probabilistic model
Move to the next subtask
N
Is S empty?
Y
Calculate the whole scheduling time, and update pheromone
End iteration?
N
Y
End

Figure 4. The Flow Diagram of Priority-driven ACO Algorithm for DAG Task Scheduling

(1) Calculate the priority of each subtask in the DAG workflow through the top-down approach.

(2) Determine the scheduling sequence according to the priority, and initialize the matrix of the pheromone information about each subtask to each virtual machine.

(3) Let ants traverse the above scheduling sequence successively, then allocate subtasks to virtual machines according to the given probability model. In the traversal process of ants, if current pheromone value is still equals to the initial constant, this shows that no ants have passed through the path, then randomly choose a virtual machine to process the subtask.

(4) After ants traverse all subtasks, the algorithm calculates the whole scheduling time, then update pheromone matrix according the scheduling time.

(5) Repeat steps 3~4 through multiple iterations. The number of iterations is a empirical value, and the traversal solution set will convergence when the iteration reaches a certain number of times.

The pseudocode of the algorithm is shown in Table 6:

Table 6. Pseudocode of Priority-driven ACO Algorithm for DAG Task Scheduling

Priority-driven ACO algorithm for DAG task scheduling

```
input the DAG workflow $G(V,E)$
initialize the parameters of $V_i \in V$ and $Vms$
calculate priority of $V_i$ incrementally
sort $V_i$ in ascending order, and save $V_i$ to List $L$
initialize the ant colony and the pheromone matrix $M$ to $C$
for $i$=1$\rightarrow$ $m$ (number of ants)
    for $j$=1$\rightarrow$$n$ (length of list $L$)
        look $M_{ij}$ (pheromone value $edge(i,j) \in E$ ) in $M$
        if $M_{ij}$==$C$
            Randomly choose a $Vm_j \in Vms$
        else
            choose the best $Vm_j$ according to formula(10)
        end if
        let $V_i \rightarrow Vm_j$
    end for
    calculate the total scheduling time $T$
    update $M$ by $T$
end for
```

5. Simulation Results and Analysis

In order to verify the effectiveness of the scheduling algorithms, the greedy algorithm, the priority scheduling algorithm and the priority-driven ACO algorithm are embedded into the CloudSim3.0 [16] in this paper. We compare their performance and draw the relevant experimental data under Windows 7 operating system and Eclipse programming platform. The experiment is on the basis of the establishment of DAG workflow, adopts the hierarchical way to build the DAG workflow, and uses gaussian random to set parameters which obey normal distribution. The flow diagram of DAG construction is shown in figure 5, and the explanation is as follows:

(1) Set the parameters about virtual machines such as their quantity, average processing capability and average processing capability variance, obtain concrete data through gaussian random, and save the data into a list.

(2) Set the parameters about subtasks such as their quantity, average length and average length variance, obtain concrete data through gaussian random, and save the data into a temporary list.

(3) Form a DAG workflow by associating subtasks. Since subtasks in a DAG workflow may have more than one parent node or one child node, the experiment adopts the hierarchical way to create association in order to avoid the formation of a ring.

(4) Set the number of layer in the DAG workflow and the number of nodes in each layer randomly (the first layer usually contains only a root node), and make the sum of each layer of nodes equal to the given number of subtasks. Then, match the subtasks of each layer by means of the temporary list in step 2, and set the parent node randomly in the process of matching.

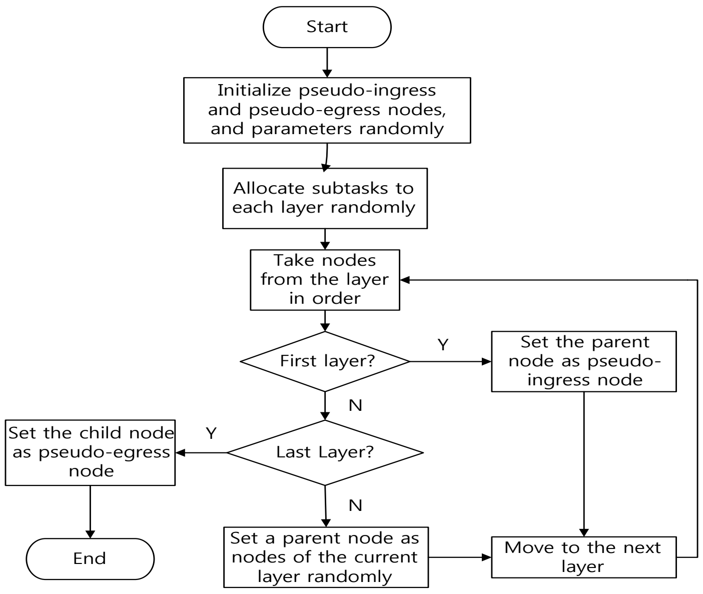
Start
Initialize pseudo-ingress and pseudo-egress nodes, and parameters randomly
Allocate subtasks to each layer randomly
Take nodes from the layer in order
First layer?
Y
Set the parent node as pseudo-ingress node
N
Last Layer?
Y
Set the child node as pseudo-egress node
N
Set a parent node as nodes of the current layer randomly
Move to the next layer
End

Figure 5. The Flow Diagram of DAG Construction

In the process of the formation of DAG, nodes in each layer set parent node randomly rather than setting child nodes, because the way of setting child nodes cannot make nodes in each layer generate strong connection, just as shown in Figure 6(a); while the way of setting parent node can make nodes in each layer generate strong connection, just as shown in Figure 6(b).

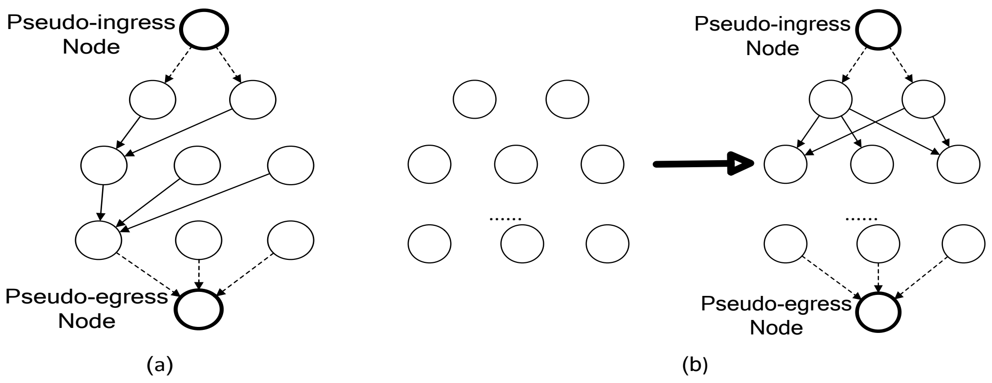

Figure 6. The Diagram of Two Different DAG Construction Strategies

Figure 7 shows simulation comparison of three algorithms on scheduling time, and simulation parameters are as follows: the number of virtual machines are 30 virtual machines; their average processing capability is 1000, and variance of their processing capability is 200; the number of subtasks is ranged from 200 to 400; the average length of subtasks is 10000, and variance of their length is 100; the number of layer in the DAG workflow is 10, and subtasks distribute randomly in each layer; and nodes in each layer set parent node randomly. Seen from simulation results in Figure 7, when the number of subtasks is less than 300, the number of subtasks in each layer is relatively small, the system has enough virtual machines to process subtasks, so the total scheduling time of these three algorithms is not distinct. However, with the increase of number of subtasks, the total scheduling time of these three algorithms varies distinctly, and the priority-driven ACO algorithm can better build optimal solution in the global scope.

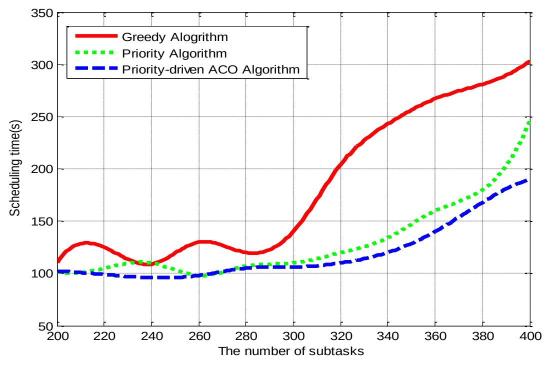

Figure 7. Simulation Comparison of Three Algorithms on Scheduling Time

Supposed that load factor is the standard quantity deviation of subtasks allocated to virtual machines. If some virtual machines allocate a large number of subtasks, while others are quite free, the system load is imbalanced. Figure 8 shows simulation comparison of three algorithms on load factor, and simulation parameters are as follows: the number of virtual machines are 20 virtual machines; their average processing capability is 1200, and variance of their processing capability is 150; the number of subtasks is ranged from 100 to 400; the average length of subtasks is 15000, and variance of their length is 200; the number of layer in the DAG workflow is 10, and subtasks distribute randomly in each layer; and nodes in each layer set parent node randomly. Seen from simulation results in figure 8, compared with the greedy algorithm and the priority algorithm, the priority-driven ACO algorithm can realize simple load balancing by scheduling idle virtual machines with higher priority in cloud environment.

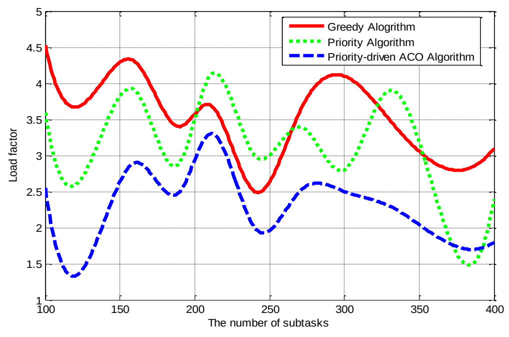

Figure 8. Simulation Comparison of Three Algorithms on Load Factor

The number of iterations is very important for the experiment when using the priority-driven ACO algorithm. Too little iteration leads to the situation that ant colony will search for the optimal solution randomly, while too much iterations will affect the efficiency of the algorithm. Figure 9 shows the influence of the iterative number to priority-driven ACO algorithm, and simulation parameters are as follows: the number of virtual machines are 30 virtual machines; their average processing capability is 1000, and variance of their processing capability is 200; the number of subtasks is ranged from 200 to 400; the average length of subtasks is 10000, and variance of their length is 100; the number of layer in the DAG workflow is 10, and subtasks distribute randomly in each layer; and nodes in each layer set parent node randomly. Seen from simulation results in Figure 9, the algorithm needs around 100 iterations to start the convergence when the number of subtasks is 200, while the algorithm needs around 150 iterations to start the convergence when the number of subtasks reaches 300, and the algorithm needs around 200 iterations to start the convergence when the number of subtasks reaches 400. In addition, the total scheduling time obviously prolongs when the number of subtasks reaches 400 because there exists busy waiting phenomenon during the execution time of some subtasks.

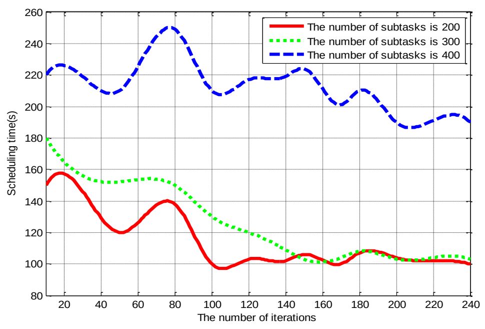

Figure 9. The Influence of the Iterative Number to Priority-driven ACO Algorithm

6. Conclusions

In the paper, two different algorithms are provided to solve the DAG task scheduling problem in cloud environment. There into, the priority algorithm for DAG task scheduling is a bottom-up approach in calculating priority of subtasks, and dispatches subtasks in accordance with priority. On the basis of the above algorithm, we present a priority-driven ACO algorithm to optimize DAG task scheduling in cloud environment. The priority algorithm pays attention to enhance resource utility and increase throughput of virtual machines, while the priority-driven ACO algorithm attaches importance to global performance optimization on the problem. In the end, we compare these algorithms through simulation in the CloudSim platform, and verify the efficiency of these algorithms.

Acknowledgements

This work has been supported by the General Program for National Natural Science Foundation of China (No. 61170135), the National Natural Science Foundation of China for Young Scholars (No. 61202287), the Emergency Management Program for National Natural Science Foundation of China (No. 61440024), and the General Program for Natural Science Foundation of Hubei Province in China (No. 2013CFB020).

References

[1] L. Rodero-Merino, E. Caron, A. Muresan and F. Desprez, "Using clouds to scale grid resources: An economic model", Future Generation Computer Systems, vol. 28, no.4, **(2012)**, pp. 633-646.
[2] F. A. Omara and M. M. Arafa, "Genetic algorithms for task scheduling problem", Journal of Parallel and Distributed Computing, vol. 70, no. 1, **(2010)**, pp. 13-22.
[3] S. Pandey, L. Wu, S. M. Guru and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments", 24th IEEE International Conference on Advanced Information Networking and Applications, **(2010)**, pp. 400-407.
[4] S. Zhan and H. Huo, "Improved PSO-based Task Scheduling Algorithm in Cloud Computing", Journal of Information and Computational Science, vol. 9, no. 13, **(2012)**, pp. 3821-3829.
[5] Z. Liu, T. Qin, W. Qu and W. Liu, "DAG cluster scheduling algorithm for grid computing", 14th IEEE International Conference on Computational Science and Engineering, **(2011)**, pp. 632-636.
[6] J. G. Barbosa J. G. and Moreira Belmiro, "Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters", Parallel Computing, vol. 37, no. 8, **(2011)**, pp. 428-438.
[7] N. A. Bahnasawy, O. Fatma, M. A. Koutb And M. Mervat, "A new algorithm for static task scheduling for heterogeneous distributed computing systems", African Journal of Mathematics and Computer Science Research, vol. 4, no. 6, **(2011)**, pp. 221-234.

[8] X. Tang, K. Li, G. Liao and R. Li, “List scheduling with duplication for heterogeneous computing systems”, Journal of parallel and distributed computing, vol. 70, no. 4, **(2010)**, pp. 323-329.
[9] N. Al-Oudat and M. Govindarasu, “Task scheduling in heterogeneous distributed systems with security and QoS requirements”, International Journal of Communication Networks and Distributed Systems, vol. 9, no. 1, **(2012)**, pp. 21-36.
[10] D. Kliazovich, J. E. Pecero, A. Tchernykh, P. Bouvry, S. U. Khan and A. Y. Zomaya, “CA-DAG: Communication-aware directed acyclic graphs for modeling cloud computing application”, IEEE International Conference on Cloud Computing, **(2013)**, pp. 277-284.
[11] F. Ren and J. Yu, “Multiple DAGs scheduling based on lowest transportation and completion time algorithm on the cloud”, 7th ChinaGrid Annual Conference, **(2012)**, pp. 33-35.
[12] B. Saovapakhiran, G. Michailidis and M. Devetsikiotis, “Aggregated-DAG scheduling for job flow maximization in heterogeneous cloud computing”, GLOBECOM-IEEE Global Telecommunications Conference, **(2011)**, pp. 1-6.
[13] S. Xue, M. Li, X. Xu and J. Chen, “An ACO-LB algorithm for task scheduling in the cloud environment”, Journal of Software, vol. 9, no. 2, **(2014)**, pp. 466-473.
[14] C. Mateos, E. Pacini and C. G. Garino, “An ACO-inspired algorithm for minimizing weighted flowtime in cloud-based parameter sweep experiments”, Advances in Engineering Software, vol. 56, **(2013)**, pp. 38-50.
[15] K. Nishant, P. Sharma, V. Krishna, C. Gupta, P. Singh, K. Nitin and R. Rastogi, “Load balancing of nodes in cloud using ant colony optimization”, 14th International Conference on Modelling and Simulation, **(2012)**, pp. 3-8.
[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose and R. Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms”, Software: Practice and Experience, vol. 41, no. 1, **(2011)**, pp. 23-50.

Authors

Haitao Xie, He received the B.S. degree in industrial electric automation from Wuhan University of Science and Technology, Wuhan, China, in 1999, and the M.S. and Ph.D. degrees in Electronic Information Engineering from Huazhong University of Science and Technology, Wuhan, China, in 2004 and 2009, respectively, he is currently a lecturer at School of Computer Science in Hubei University of Technology, Wuhan, China. His research interests include networking, information security and cloud computing.

Hongwei Chen, In 2006, he graduated from Nanjing University of Posts & Telecommunications and received PHD degree in China, majored in Communication and Information System. He is an associate professor at School of Computer Science in Hubei University of Technology, Wuhan, China. From August of 2013 to February of 2014, he was an academic visiting scholar at Temple University in USA. Now his major study field is Peer-to-Peer Computing, Cloud Computing and SDN.

Chunzhi Wang, She is a professor and dean at School of Computer Science in Hubei University of Technology, Wuhan, China. She received PHD degree at Wuhan University of Technology, interested in Peer-to-Peer Computing, Cloud Computing and network security. She is a member of CCF, ACM and IEEE.