Cloud testing scheduling based on improved ACO

Yang Zheng[1,2 a], Lizhi Cai[*2,3 b], Shidong Huang[4,c] , Jiawen Lu[1,d] and Pan Liu[5,e]

[1]*College of Information Science & Engineering of East China University of Science and Technology, Shanghai 200237, China,*

[2]*Shanghai Key Laboratory of Computer Software Evaluating & Testing, Shanghai 201112, China*

[3]*Shanghai Industrial Technology Institute, Jinsu Road, Shanghai 201206, China*

[4]*Department of Computer Science and Technology of Xinyang Normal University, Henan 464000, China*

[5]*College of Computer Engineering and Science, Shanghai Business School, Shanghai 201400, China*

[a]*sunioryang@163.com,* [b]*clz@ssc.stn.sh.cn,* [c]*lovehuishouzan@163.com,* [d] *1184407232@qq.com,* [e]*panl008@163.com*

Abstract.

Resources scheduling plays an important role in Cloud testing. The completion time for the entire testing works in Cloud testing and the cost of Cloud services could both reduce a lot through good scheduling strategies. This paper mainly focuses on the dependencies between testing tasks and proposes ACO_TD(ACO based on testing task dependencies). ACO_TD not only possesses advantages of ACO, but also makes up shortcomings of ACO such as slow convergence and easy falling into local optimization. CloudSim is used for simulation experiment, and ACO_TD has acquired faster execution speed and better load balancing of VM compared with RR, GA and ACO in experiment. The advantages of ACO_TD become more and more obvious as the scale of testing tasks in Cloud grows.

Keywords: Cloud, Cloud testing, VM, ACO, CloudSim

Introduction

With the continuous development of Cloud computing technology [1], software testing has been confronted with unprecedented challenges [2]. In the process of traditional software testing, users often need to install software, set up

testing environment and test product according to test manual or test procedure. Cloud computing not only allows users to share cloud resources with minimum cost, but also overcomes shortcomings of traditional software testing such as expensive cost of testing tools, limited testing capability and complex operations for setting up testing environment. Users needn't to set up testing environment, and could finish testing work through distributed computing.

During Cloud testing, users would get testing results after submitting testing scripts to Cloud platform. The Cloud platform selects appropriate VM scheduling strategies which would directly affect the efficiency Cloud testing. The scheduling in Cloud testing is actually to select a suitable dynamic combination which used to allocate VM to testing task. The features of ACO [3,4] such as parallel distribution and flexibility, make it suitable to solve scheduling problems in Cloud testing [5,6]. A lot of scholars have already studied testing task scheduling in Cloud so far. Zuo et al. [7] proposed advanced Min-Min algorithm based on pre-classification, taking the computing ability and communications of resources into consideration. Zhang et al. [8] presented a grouping and polymorphic ACO, which divided ants into groups on basic of function and reduced the average completion time.

This paper has a further study about ACO's mechanism [9], analyses the special features of testing task scheduling in Cloud and proposes ACO_TD which is based on ACO and testing task dependencies. Section 2 describes VM scheduling and testing task dependencies in Cloud testing. Section 3 describes how to schedule VM by ACO_TD. Section 4 shows the advantages of ACO_TD through simulation experiments. Section 5 summarizes the work of this paper and plans on the future work.

Scheduling in Cloud testing

1) VM scheduling in Cloud testing

Cloud testing is on the basic of Cloud computing, which has many features such as high reliability, large scale, virtualization, versatility and on-demand services. Users could get relevant testing reports after submitting testing tasks to the deployed testing environment. Appropriate VM scheduling strategies would be necessary when there're several testing tasks waiting in the queue or in execution. VM scheduling usually is an important part of Cloud testing whose essence is a kind of NP-complete problem. The goal of VM scheduling generally is to finish testing work within expected time, acquire high utilization of VM and get high throughput of Cloud testing.

The process of VM scheduling in Cloud testing is described as shown in Fig. 1. After users submit testing tasks, the Cloud platform would allocate VM for tasks according to selected scheduling strategy. The completion time usually determines resource cost because Cloud services charge in a "pay as you go" way. Therefore, good VM scheduling strategy not only improves efficiency but also reduces cost.

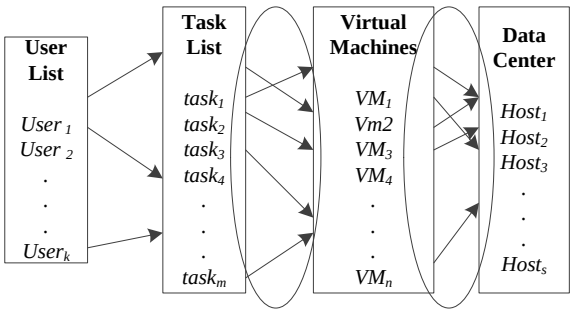

Fig. 1 Task scheduling model in Cloud testing

2) Description of testing task dependencies in Cloud testing

In addition to appropriate VM scheduling strategies, dependencies between testing tasks also affect testing efficiency. During traditional software testing, test dependencies have a great influence on collaboration and decomposition of tasks. Similarly, testing task dependencies in Cloud testing would influence the testing efficiency a lot. If the dependencies between testing tasks have been taken into consideration, the number of scheduling VM would reduce and the efficiency of Cloud testing would improve. In Cloud testing, the dependencies between $task_i$ and $task_j$ could be divided into the following two types:

(1) Single dependency

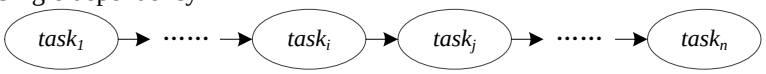

Fig. 2 One-on-One dependency relationship between testing tasks

Single dependency is One-on-One dependency, which means there's a one-to-one match between $task_i$ and $task_j$. In Fig. 2, each testing task (except the first task and the last task) has a precursor and a subsequent, and has a direct influence on other testing tasks. The outcome of $task_i$ is input of $task_j$, and $task_j$ depends on $task_i$.

(2) Multiple dependencies

Multiple dependencies often refer to several testing tasks, which usually could be divided into three categories:

a) One-to- Many dependency

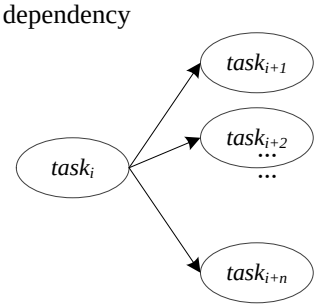

Fig. 3 One-to-Many dependency relationship between testing tasks

In Fig. 3, $task_i$ is the precursor of $task_j (j = i+1, i+2, ..., i+n)$, and $task_j$ won't be executed until $task_i$ is over. The outcome of $task_i$ is input of $task_j$, and $task_j$ depends on $task_i$.

b) Many-to-One dependency

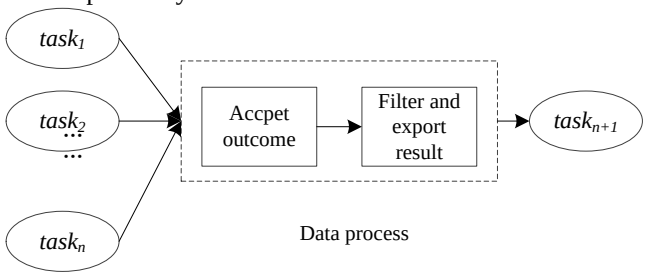

Fig. 4 Many-to-One relationship between testing tasks

In Fig. 4, $task_i (i = 1, 2, ..., n)$ is the precursor of $task_{n+1}$, and $task_{n+1}$ won't be executed until $task_i$ is over. The outcome of $task_i$ is input of $task_{n+1}$, and $task_{n+1}$ depends on $task_i$.

c) Many-to-Many dependency

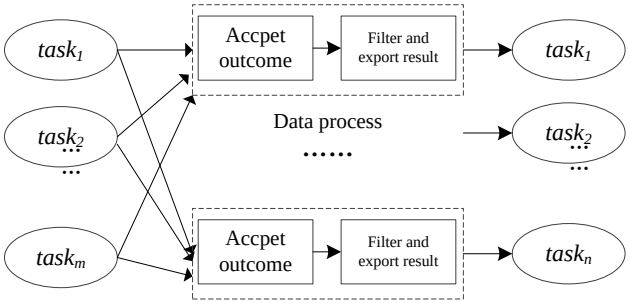

Fig. 5 Many-to-Many relationship between testing tasks

In Fig. 5, $task_i (i = 1, 2, ..., m)$ is the precursor of $task_j (j = 1, 2, ..., n)$ and $task_j$ won't be executed until $task_i$ is over. Therefore, $task_j$ depends on $task_i$, and the length of $task_i$ restricts the start execution time of $task_j$.

Through the description of testing task dependencies, we could infer that the dependencies between testing tasks play an important role during the Cloud testing. A lot of resources could be saved once these dependencies have been considered about.

VM scheduling based on ACO_TD

1) Problem description

Suppose there're n testing tasks and m VMs in Cloud testing. Matrix $TDD_{n\times n}$ is used to describe the dependency relationship between testing tasks shown as Eq. 1. VM is exclusive and unshared, and would not be reallocated until the testing tasks finish or fail.

$$TDD_{ij}=\begin{cases}1, task_j \ \ depends\, on\, task_i\\ 0, task_j \ \ is\ independent\, of \ \ task_i\end{cases} \tag{1}$$

The expected completion time matrix ETC_{ij} describes how long $task_i$ runs on vm_j, shown as Eq. 2. VM scheduling mainly intends to set up mapping between testing task and VM. Apart from taking advantages of testing task dependencies, this paper also makes use of ACO's global optimization to search for best mapping in the process of scheduling.

$$ETC_{n\times m}=\begin{bmatrix}ETC_{11} & \cdots & ETC_{1m}\\ \vdots & \ddots & \vdots\\ ETC_{n1} & \cdots & ETC_{nm}\end{bmatrix} \tag{2}$$

2) Description of ACO_TD

The final result of combinatorial optimization by ACO is to select suitable VM for each testing task. The solution could be $s=\{X_1, X_2 \ldots, X_n\}$, and X_i stands for decision variable. The number of testing tasks is n, and the domain of X is a set of $(task_i, vm_j)$ which turns to be the Cartesian product of $\{task_1, task_2, ..., task_n\}$ and $\{vm_1, vm_2, ..., vm_m\}$. $(task_i, vm_j)$ means allocating vm_j to $task_i$. The process of VM scheduling by ACO_TD could be described as follows and Fig. 6 shows the flow chart.

Step1: Initialize the pheromone matrix. The initialized pheromone value is related to VM computing capacity(*MIPS*), network bandwidth, the length of testing tasks, etc. shown as Eq. 3. Users initialize $TDD_{n\times n}$ according to the dependencies between testing tasks.

$$\tau_0=\frac{1.0}{(m*\sum_{k=0}^{m-1} minETC_k)}$$

(3)

Step2: Calculate the probability of allocating vm_j to $task_i$ at the moment of t(s). According to $TDD_{n\times n}$, search $task_k$ which $task_i$ depends on. If $task_k$ exists, then allocate VM used by $task_k$ to $task_i$; else calculate the probability according to Eq. 4. c_{ik} stands for the total minimum execution time by VM, and η_{ik} stands for VM computing ability. If the current calculation time is minimum, then set it as bestExecutionTime.

$$p_{ij}^{k}=\begin{cases}\dfrac{[c_{ij}(t)]^{\alpha}*[\eta_{ij}(t)]^{\beta}}{\sum_{k\epsilon Allowed_k}[c_{ik}(t)]^{\alpha}*[\eta_{ik}(t)]^{\beta}}, (j\in allowed_k)\\ 0, otherwise\end{cases}$$

(4)

Step3: Update pheromone. Pheromone update in this paper includes global update and local update.

a) Local pheromone update

Ants update local pheromone from $(task_i, vm_i)$ to $(task_{i+1}, vm_{i+1})$ according to Eq. 5, when ants allocate VM for testing task.

$$\tau_i(t_1)=(1-\rho)*\tau_i(t)+\rho*\tau_0$$

(5)

b) Global pheromone update

Ants would update the global pheromone according Eq. 6, when all the testing tasks have been executed. L_k stands for the entire execution time got by ant k, and Q is a constant, shown in Eq. 7.

$$\tau_{ij}(new)=(1-\rho)*\tau_{ij}(old)+\sum_{k=1}^{m}\Delta\tau_{ij}^{k}$$

(6)

$$\Delta\tau_{ij}^{k}=\begin{cases}\dfrac{Q}{L_k}, if\ the\ k^{th}\ ant\ chooses\ (Task_i, Vm_j)\\ 0, otherwise\end{cases}$$

(7)

Step4: Start the next cycle until the number of cycles reaches the maximum.

The VM scheduling described in this paper is mainly to allocate exclusive VM to testing tasks. Those testing tasks with dependency would get the same VM. The VM resources would be reallocated when all the testing tasks have been executed.

Experimental Results and Discussion

In order to verify the effect on VM scheduling by ACO_TD, this paper selects CloudSim [10] for simulation. DataCenter, Cloudlet, VM and auxiliary classes have been used to simulate computing resources, network resources, testing tasks and VM resources. DataCenterBroker has also been used to construct different allocation algorithms in the simulation environment. Table 1 shows the parameters configuration for simulation platform. Users customize TDD according to the dependencies between testing tasks, and initialize TDD on the basic of task dependencies.

Table 1 Parameters configuration for testing tasks in simulation experiment

VM	VM(*MIPS*)	Cloudlet Number	Cloudlet length	Ant Number	Iteration Number
6	600-2000	20	400-5000	5	300
6	600-2000	40	400-5000	15	300
6	600-2000	60	400-5000	25	300
6	600-2000	80	400-5000	35	300

Table 2 Parameters configuration for ACO

Parameter Type	Value
Inspiration factor α	$\alpha = 1.0$
Expectation inspiration factor β	$\beta = 5.0$
Pheromone volatilization factor ρ	$\rho = 0.5$
Pheromone density Q	$Q = 5.0$

The algorithms selected during experiment include RR(Round-Robin Scheduling), GA(greedy algorithm), ACO and ACO_TD. According to the work of D.Ramesh et al.[11], the parameters in the experiment could be set as shown in Table 2. These four algorithms share the same simulation environment, cloudlets and VM resources. In order to get fair testing result, the result in the experiment is the mean value of 10 experiment results. Fig.7 shows the comparison among these algorithms, when cloudlets rise from 20 to 80.

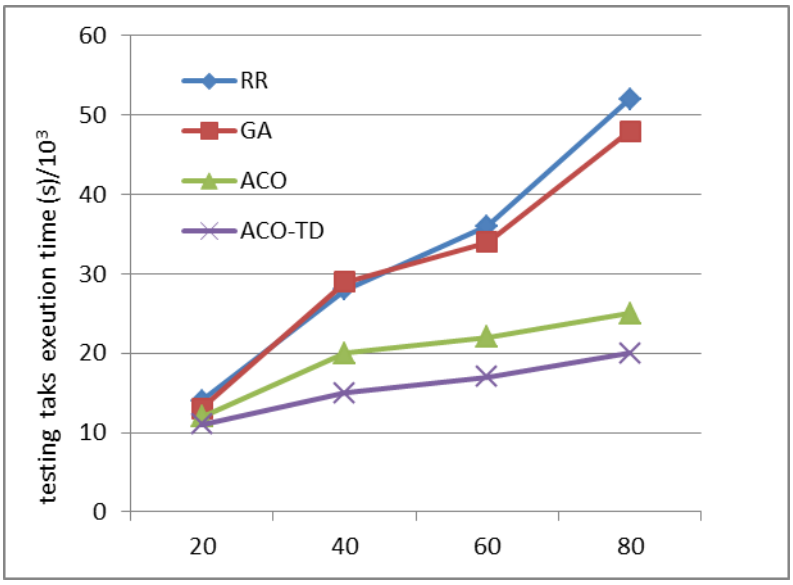

Fig. 7 Execution time for testing tasks scheduling

In Fig. 7, the advantage of ACO_TD becomes more and more obvious as the amount of testing tasks increases. The convergence of ACO is an important factor that influences the scheduling effectiveness and restricts the scheduling speed as the amount of testing tasks increase. Tasks dependencies make up such shortcomings and avoid getting into local optimization. ACO_TD also takes feedback to update pheromone density and improves efficiency.

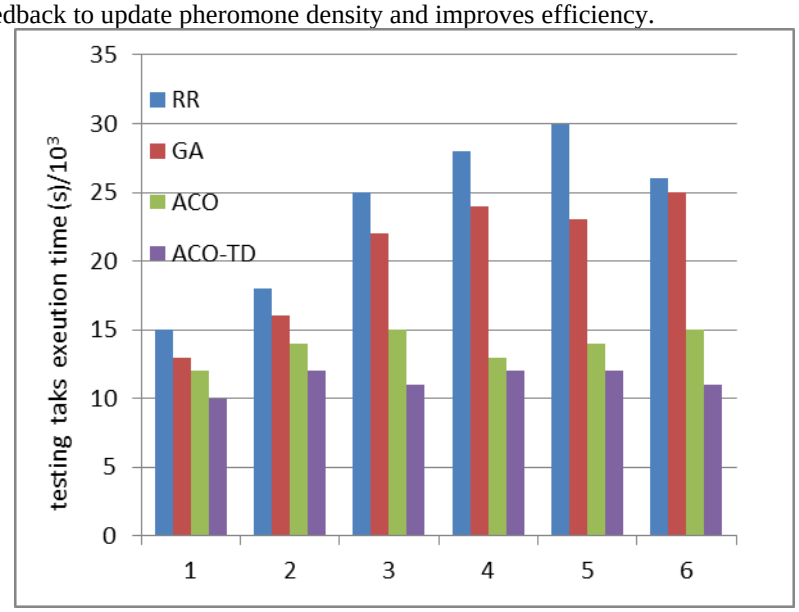

Fig. 8 Comparison of VMs' load balance among four algorithms

Fig. 8 shows load balancing of 6 VMs when the number of testing tasks reaches 60. In Fig 8, the load balancing is worst for RR. GA selects VM according to testing tasks' current situation and searches for local optimal solution instead of global optimal solution. Therefore, the load balancing of VM is not well at times. ACO treats an allocation as an object for search and obtains good load balancing of VM. However, ACO's convergence restricts its ability while ACO_TD takes advantages of testing tasks dependencies to overcome this weak point.

Conclusion

In order to reasonably execute testing tasks in Cloud testing, the minimum execution time and load balance of VM should both be taken into account. In reference to the study of ACO during these years, this paper proposes to take testing task dependencies into consideration. In this paper, an allocation for task was treated as a search object, and ACO_TD has got good results. In the process of experiment, CloudSim has been used for simulation to compare performance among different algorithms including RR, GA, ACO and ACO-TD.

The experiment results have shown the advantage of ACO_TD, which executes testing tasks within minimum time and acquires good load balancing of VM. However, the testing task dependencies would not only affect the total execution time, but also influence the convergence. If testing task dependencies change a lot, then the efficiency of ACO_TD would also change. Thus, ACO_TD still could be improved greatly in the future.

Acknowledgements

This work was financially supported by Program of Shanghai Subject Chief Scientist (13XD1421800), STCSM Project (14511106804, 13511505303 and 13DZ0500700), Shanghai Natural Science Fund (13ZR1429600), National Natural Science Foundation (6127097), and Foundation of Shanghai Committee of Science and Technology (14YF1412700).

References

[1] Fox A, Griffith R, Joseph A, et al. Above the clouds: A Berkeley view of cloud computing[J]. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 2009, 28: 13

[2] Collard R. Performance innovations, testing implications[J]. Software Test & Performance Magazine, 2009, 6(8): 19-20.

[3] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," in Proceedings of the first European conference on artificial life, 1991, pp. 134-142.

[4] A. Colorni, M. Dorigo, and V. Maniezzo, "An Investigation of some Properties of an``Ant Algorithm"," in PPSN, 1992, pp. 509-520.

[5] Wang Tianqing, Xie Jun, Zeng Zhou. Study of grid resource scheduling strategy based on ant colony algorithm[J]. Computer Engineering & Design, 2007, 28(15): 3611-3612.
[6] Zha Yinghua, Yang Jingli. Task scheduling in Cloud Computing based on improved ant colony optimization[J]. Computer Engineering & Design, 2013, 34(5): 1716-1719.
[7] ZUO Liyun, ZUO Liffeng. Cloud computing scheduling optimization algorithm based on reservation category[J]. Computer Engineering & Design, 2012, 33(4): 1357-1361
[8] ZHANG Chunyan, LIU Qinglin,MENG Ke. Task allcation based on ant colony optimization in cloud computing [J]. Journal of Computer Application, 2012, 32(5):1418-1420.
[9] Dorigo M, Birattari M, Stutzle T. Ant colony optimization[J]. Computational Intelligence Magazine, IEEE, 2006, 1(4): 28-39.
[10] Calheiros R N, Ranjan R, Beloglazov A, et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. Software: Practice and Experience, 2011, 41(1): 23-50.
[11] Duan H, Ma G, Liu S. Experimental study of the adjustable parameters in basic ant colony optimization algorithm[C]//Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. IEEE, 2007: 149-156.