An ACO-Based Scheduling Strategy on Load Balancing in Cloud Computing Environment

Wei-Tao Wen*, Chang-Dong Wang§, De-Shen Wu† and Ying-Yan Xie‡
School of Mobile Information Engineering, Sun Yat-sen University, Zhuhai, P. R. China, 519082
Email: *332974874@qq.com, §changdongwang@hotmail.com, †515076287@qq.com, ‡1807637279@qq.com

***Abstract*—Overload balance of cloud data centers is a matter of great concern. Live migration of virtual machines presents an effective method to realize load balancing and to optimize resources utilization. With the rapidly increasing scale of cloud data centers, traditional centralized migration strategy begins to show lack of scalability and reliability. In this paper, we propose a novel distributed VM migration strategy based on a metaheuristic algorithm called Ant Colony Optimization. In our ACO-VMM Strategy, local migration agent autonomously monitors the resource utilization and launchs the migration. At monitoring stage, it takes both the previous and current system condition into account to avoid unnecessary migrations. Besides, it adopts two different traversing strategies for ants in order to find the near-optimal mapping relationship between virtual machines (VMs) and physical machines (PMs). Experimental results show that ACO-VMM outperforms the existing migration strategies by achieving load balance of whole system, as well as reducing the number of migrations and maintaining the required performance levels.**

***Index Terms*—overload balance; distribution; migration strategy; ant colony optimization**

I. Introduction

In recent years, with the advent of cloud computing, more and more computing and storage resources are clustered in data centers. On cloud computing platforms, resources are provided as service and by needs, and it is stick to the Service Level Agreement (SLA) [1]. However, in such a scenario, how to efficiently manage resources brings enormous challenges to platform providers.

At present, thanks to the rapid development of virtualization technology, employing virtual machines as computing resources in cloud computing platform becomes a new way to solve the problem of dynamic resources management. In cloud systems, virtualization plays an important role by allowing online sharing of computing resources [2]. It is also able to carry out live migrations between virtual machines (VMs) and physical machines(PMs) based on the change of load. Live migration refers to the technique which can deploy a VM from the original PM to a new PM without disconnection between clients. Some Virtual machine monitors, such as XEN [3], KVM [4], can support the migrations of the VMs on different PMs. Therefore, the problem discussed above becomes how to efficiently and dynamically manage the virtual infrastructure. Due to the highly dynamic heterogeneity of resources on cloud computing platform, in order to improve resource utilization, VMs must be properly allocated and load balancing must be guaranteed [5]. Therefore, we need to find valid ways to schedule VM resources to realize load balancing in cloud computing and to optimize resource utility.

Over the years, there are many researches on VM scheduling strategy. One well-known strategy is centralized VM live migration scheduling scheme, as shown in Fig. 1. The scheduler mainly consists of two components: central controller and local migration controller. The central controller obtains all the physical resources utilization on the whole, and then initializes the VM migration based on pre-specified policies so as to achieve load balancing and high resource utilization. The local migration controller sends its physical resources utilization to center controller and receives instructions to start migration. However, researchers have found some defects of such framework like lack of reliability and scalability [6]. If the central controller breaks down, it will easily lead to load imbalance of the system. Moreover, the efficiency is another bottleneck of the entire system.

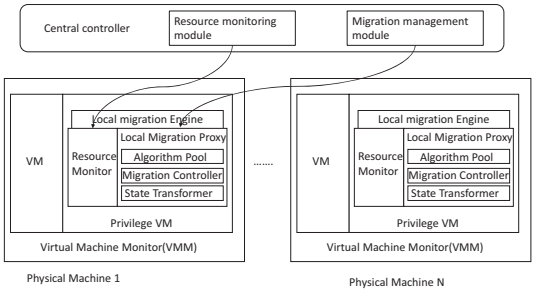

Fig. 1. The traditional centralized virtual machine live migration framework.

To this end, some distributed strategy would be a better choice, e.g. the one proposed by Wang *et al.* [6]. However, this strategy also has disadvantages such as simple trigger strategy and misuse of pheromone.

In this paper, we propose a novel distributed VM migration strategy to solve the above problems. In our strategy, distributed local migration agents autonomously monitor the resource utilization of each PM. Unlike some research just considering the CPU utilization, our monitoring strategy is more complex by additionally taking the memory utilization and the bandwidth condition into account. If the agents finds that the PM is in high load condition, it will launch the migrations immediately. However, in order to avoid unneces-

978-1-4673-9295-2/15 $31.00 © 2015 IEEE
DOI 10.1109/FCST.2015.41

sary migrations triggered by single utilization peaks, we also take both the previous and current conditions of the system into consideration while monitoring the resource utilization. Besides, we propose a dynamic VM migration strategy that uses a highly adaptive optimization algorithm called Ant Colony Optimization. The proposed ACO-Based VM Migration Strategy (ACO-VMM) uses artificial ants to find a near-optimal mapping of PMs and VMs. The performance of the proposed ACO-VMM strategy is evaluated by using CloudSim toolkit package [7]. Experimental results show that ACO-VMM achieves load balancing and reasonable resource utilization, which outperforms existing migration strategies in terms of number of VM migrations and number of SLA violations.

II. Related Work

So far, many researches have been conducted on the problem of load imbalance and resource utilization in terms of various scheduling strategies. Liu *et al.* [8] proposed a solution to load imbalance in live migration of virtual machines, which addresses the clustering conflict problem in traditional techniques of load balance solution by choosing target node based on weighted probability of forwarding. Liu *et al.* [9] developed a new scheduling policy based on workload characteristic, which triggers migration by a multi-threshold method and accomplishes the task of choosing VMs to migrate and migration target. Yang *et al.* [10] proposed a PM-LB virtual machine deployment algorithm with consideration of system load balance. This algorithm has standard description of physical nodes using the method of performance vector. Pei *et al.* [11] presented a virtual machine placement policy based on PSO (Particle Swarm Optimization). Lei *et al.* [12] presented a load balancing system and designed a scheduling algorithm based on multi-objective genetic algorithm to get a new mapping relationship between PMs and VMs.

However, in the study of dynamic scheduling system, many researches adopted the centralized strategy on migration which lacks scalability and reliability. Therefore, Wang *et al.* [6] proposed a distributed strategy of virtual machine migration. This strategy creates load balance model of hosts and divides several load balance domains. Migration agents monitor its usage of resources by themselves. Once overloaded, virtual machine migration will be triggered. This algorithm absorbs the usage of pheromone in Ant System (AS) [13] and sets new maximum and minimum values of pheromone, which are used to restrict ants traversing and enhances the ability of global searching. The shortcomings of this dynamic migration framework include simple trigger strategy and misuse of pheromone. Single trigger would easily cause a large number of migrations happening, which increases the overhead of dynamic migration greatly. Also, ant colony optimization is an algorithm which depends on the positive feedbacks of the pheromone. But the above distributed strategy does not take this advantage. To this end, in our novel ACO-VMM strategy, we reconsider the usage of pheromone in different times when the ants are traversing.

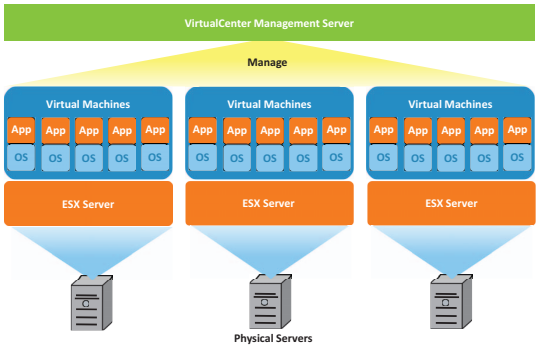

Fig. 2. The mapping of VMs and physical machines.

III. The Model Design of VM Scheduling

A cloud data center consists of a large number of heterogeneous PMs with different resource capacities. At any given time, a cloud data center usually serves many simultaneous users and as a result, VMs with different resource capacities are created on PMs [14]. To efficiently and dynamically manage the VM resources, in this section, we first present the model of VM. Then we come up with the expression of load in a cloud data center. Finally we propose the evaluation of load balance in a data center.

A. VM Model

As shown in Fig. 2, several VMs may be deployed into a PM, and the number of the VMs in each PM can be different from each other. The set of all the PMs in a data center is $P = \{p_1, p_2, \ldots, p_N\}$, where N is the number of PMs and $p_i, \forall i = 1, \ldots, N$ represents the i-th PM. Besides, the set of all the virtual machines in a data center is $V = \{v_1, v_2, \ldots, v_M\}$, where M is the number of VMs and $v_i, \forall i = 1, \ldots, M$ represents the i-th VM.

B. The Expression of Load

During a time zone of T when the data center is serving various requests, in order to know the condition of PM in different time, we need to divide time T into n time periods. Hereby we define $T = [(t_0, t_1), (t_1, t_2), \ldots, (t_{n-1}, t_n)]$. In the definition, (t_{k-1}, t_k) refers to time period k. And we suppose the load of VMs is relatively stable in every period, so we can define the load of the j-th VM on the i-th physical machine in period k as $V_i(j, k)$. As a result, the average load of the$j$-th VM on th i-th physical machine in time T is

$$\overline{V_i(j,T)} = \frac{1}{T}\sum_{k=1}^{n} V_i(j,k)(t_k - t_{k-1}) \tag{1}$$

$V_i(j,k)$, the resources used by the j-th VM on the i-th PM in period k, is defined as

$$V_i(j,k) = \frac{1}{K}\left(\overline{U}_{cpu}^2 + \overline{U}_{mem}^2 + \overline{U}_{net}^2\right) \tag{2}$$

where

$$K = \overline{U}_{cpu} + \overline{U}_{mem} + \overline{U}_{net} \tag{3}$$

$$\overline{U}_{cpu} = 1/2\left(U_{cpu_{t_k}} + U_{cpu_{t_{k-1}}}\right) \tag{4}$$

$$\overline{U}_{mem} = 1/2\left(U_{mem_{t_k}} + U_{mem_{t_{k-1}}}\right) \tag{5}$$

$$\overline{U}_{net} = 1/2\left(U_{net_{t_k}} + U_{net_{t_{k-1}}}\right) \tag{6}$$

The notation $U_{cpu_{t_k}}$ means the utilization of CPU by the j-th VM on the i-th PM in the moment k, and it is similar for the notations U_{memt_k}, and $U_{net_{t_k}}$. Since we have assumed that in every period k, the load of VMs is relatively stable, thus $U_{cpu_{t_k}}$, $U_{mem_{t_k}}$, and $U_{net_{t_k}}$ are all constants.

After defining the resources used by certain VM, we can summarize the load of PM p_i as follows,

$$P\left(i,T\right) = \sum_{j=1}^{m_i} \overline{V_i\left(j,T\right)} \tag{7}$$

where m_i represents the number of the VMs running on the PM p_i.

Periodically, the local migration agents will monitor the load of the PM. If the load of PM P_i exceeds the pre-set threshold, will launch the migrations. The value of the threshold is set by experience.

C. The Evaluation of Load Balance

Here we adopt variance to evaluate the load condition of the whole system.

$$\sigma\left(T\right) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\overline{P\left(T\right)} - P\left(i,T\right)\right)^2} \tag{8}$$

where

$$\overline{P\left(T\right)} = \frac{1}{N}\sum_{i=1}^{N} P\left(i,T\right) \tag{9}$$

The larger value of the $\sigma\left(T\right)$, the much more imbalanced the load is in a data center. With larger $\sigma\left(T\right)$, resource are used more unevenly in the data center. Therefore, the key point for overload balance problem is to make $\sigma\left(T\right)$ as small as possible.

IV. Implementation of ACO-Based VM Migration Strategy

The Ant Colony Optimization [13] is an algorithm for finding near-optimal paths based on the behavior of ants foraging for food. At first, the ants traverse through diverse paths randomly. When an ant finds food, it walks back to the colony leaving "markers" (pheromone) that shows the path to the food. When other ants come across the markers, they are likely to follow the path with a certain probability. If they do, they then populate the path with their own markers as they bring the food back. As more ants find the path, it gets stronger until there are a couple streams of ants traveling to various food sources near the colony.

In our novel ACO-VMM strategy, we reconsider the usage of pheromone by establishing two completely different traversing strategy for ants.

- The first one, called Positive Traversing Strategy, makes the ants possibly traverse through PMs with high pheromone.
- The other, called Negative Traversing Strategy, makes the ants less likely to traverse through PMs with high pheromone.

Once the load condition of the PM p_i exceeds the threshold, the local migration controller will sort all VMs in p_i according to their average loads which are calculated in the arithmetic formula 1. The higher the average load a VM has, the higher priority it has. Then we adds the VM with greatest priority into a list. We will re-do this operation by adding necessary VMs until $P\left(i,T\right)$ is small enough. After that, the local migration controller will produce several ants to traverse. The ants traverse through different PMs and generate pheromone according to the load condition of PM and bandwidth between PMs. If the load condition of destination PM is higher, or the bandwidth between the original PM and destination PM is less, the ants will leave more pheromone. In most of the iterations, the global pheromone is continuously accumulated, and the ants will be more likely to traverse through those PMs which are in high load condition according to the Positive Traversing Strategy. However, in the last iteration, the ants will possibly traverse through PMs with lower load condition since we adopt Negative Traversing Strategy. Thus, we can find out a list of PMs with possibly low load condition after all ants finishing traversing in the last iteration. The PMs in the list will be matched with VMs that are in the sorted array. In the end, we can obtain a new mapping relationship between VMs and PMs.

The pseudocode of the proposed ACO-based VM Migration (ACO-VMM) is given in Algorithm 1. The N in the algorithm is the iteration times. From step 3 to step 8, the purpose is to let the pheromone accumulate in the PMs in high load condition as much as possible.

A. The Rules of Ants Traversing

If a PM is in high load condition, it will generate several ants with certain size of memory to traverse. The VM that needs to migrate is stored in the memory of the ant. When the ant starts to traverse, the next node that will be chosen is selected based on the the amount of pheromone from the original PM to the selected PM. In the first strategy (Positive Traversing Strategy), in order to search for hosts in high load condition, the more pheromone on the way, the more likely the ant chooses the way to traverse. In time t, the possibility for the ant from node i to node j in Positive Traversing Strategy is as follows,

$$p(i,j) = \begin{cases} \dfrac{\left[\tau_{ij(t)}\right]^{\alpha}\left[net_{ij}(t)\right]^{\beta}}{\sum\limits_{k \notin vis(t)}\left[\tau_{ij(t)}\right]^{\alpha}\left[net_{ij}\left(t\right)\right]^{\beta}} & j \notin vis(t) \\ 0 & otherwise \end{cases} \tag{10}$$

Algorithm 1 ACO-Based VM Migration

1: **if** the load condition of the i-th PM $P(i,T)$ exceeds the threshold **then**
2: Find out the sorted list of VMs that should migrate.
3: **while** $i < N-1$ **do**
4: Produce ants.
5: Traversing according to the Positive Traversing Strategy.
6: Update the amount of pheromone.
7: $i = i+1$
8: **end while**
9: Produce ants.
10: Traversing according to the Negative Traversing Strategy.
11: Obtain a list of PMs in possibly low load condition.
12: Match the PMs with VMs.
13: Get a new mapping relationship.
14: **end if**

where $\tau_{ij}(t)$ denotes the amount of pheromone from node i to node j, $vis(t)$ represents hosts that the ants have visited before time t, $net_{ij}(t)$ represents initiation factor which considers the network bandwidth on edge (i,j), α and β represent the relative importance between pheromone and initiation factor respectively. Usually we set $a \geqslant 0$ and $b \geqslant 0$.

The second strategy (Negative Traversing Strategy) is similar to the first strategy. Notice that we no longer use $\tau_{ij(t)}$ but use $\frac{1}{\tau_{ij(t)}}$ to help us find PMs in low load condition. By measuring the reciprocal of the amount of pheromone, we can easily get the results.

$$p(i,j) = \begin{cases} \dfrac{\left[1/\tau_{ij(t)}\right]^{\alpha}\left[net_{ij}(t)\right]^{\beta}}{\sum\limits_{k \notin vis(t)} \left[1/\tau_{ij(t)}\right]^{\alpha}\left[net_{ij}(t)\right]^{\beta}} & j \notin vis(t) \\ 0 & otherwise \end{cases} \tag{11}$$

In addition, we need to refresh the pheromone each time after all ants have finished traversing in one iteration. Only after an ant visits the node, the pheromone in the path is increased. If the ant choose node j when it is in node i, then the pheromone in the bidirectional way from node i to node j and from node j to node i is increased. The pheromone is increased in the following way:

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{12}$$

In the equation, ρ is the evaporation rate and $0 < \rho < 1$. $\Delta\tau_{ij}(t)$ is the quantity of pheromone laid on edge (i,j), which is defined as:

$$\Delta\tau_{ij}(t) = \frac{1}{\left[net_{ij}(t)\right]^{\gamma}} + P(j,t) \tag{13}$$

In the equation, $P(j,t)$ is the load condition of the j-th PM in a relatively short time t which is defined in (7). γ is a parameter to determine the relative importance of the heuristic value with respect to the load condition.

B. Restriction of Pheromone

In order to prevent the amount of pheromone from being too large or too small, we should restrict the value of the pheromone in the interval $[\tau_{min}, \tau_{max}]$. If the value exceeds the interval, it will be reset as follows,

$$\begin{cases} \tau_{ij}(t) = \tau_{min} & \tau_{ij}(t) < \tau_{min} \\ \tau_{ij}(t) = \tau_{max} & \tau_{ij}(t) > \tau_{max} \end{cases} \tag{14}$$

The minimum value τ_{min} is most often experimentally chosen. The maximum value τ_{max} may be calculated analytically provided that the optimum ant tour is known.

C. Matching Algorithm

After all of the iterations have finished, we get a sorted list of the VMs ${L_{VM}}^{+} = \{v_1, v_2, \ldots, v_i\}$ that need to migrate and a list of destination PMs ${L_{PM}}^{+} = \{p_1, p_2, \ldots, p_k\}$ that are likely to be destination PMs. With these two lists, we start to match the VMs with the PMs. In order to find the best mapping relationship, we use a fitness function which is suggested in (15). The fitness function is defined as follows

$$\begin{aligned} Fitness(v_i, p_j) = & \frac{p_{cpu\_j} - v_{cpu\_i}}{v_{cpu\_i}} \cdot \frac{p_{mem\_j} - v_{mem\_i}}{v_{mem\_i}} \cdot \\ & \frac{p_{net\_j} - v_{net\_i}}{v_{net\_i}} \cdot \frac{p_{disk\_j} - v_{disk\_i}}{v_{disk\_i}} \end{aligned} \tag{15}$$

In the equation, v_{cpu_i}, v_{mem_i}, v_{net_i}, v_{disk_i} represent the resources that the i-th VM needs, while p_{cpu_j}, p_{mem_j}, p_{net_j}, p_{disk_j} represent the surplus resources that the the j-th PM has. If the value of fitness function is smaller than zero, it means that there is not enough resource for the i-th VM to migrate to the j-th PM. Each VM will compare with all the PMs in the list so as to find the most appropriate PM. After that, if all the PMs are not suitable, we will store the information of the VM in another migration list and restart ACO-Based VM Migration Algorithm. The algorithm used to match VMs with PMs is described in Algorithm 2. To begin with, we input ${L_{VM}}^{+}$ and ${L_{PM}}^{+}$.

V. Experimental Design and Setup

In this section, experiments have been conducted to show the effectiveness of our strategy. In particular, we have analyzed the performance of our method with different evaporation rates ρ. The analysis show that our method can generate consistently stable results in some interval. Apart from that, we have also compared our method with four existing methods, comparison results of which have confirmed the effectiveness our approach.

A. Experimental Design

To evaluate our proposed algorithm, we have used popular CloudSim toolkit package [7]. CloudSim is a simulator that can provide a generalized, and extensible simulation framework which enables seamless modeling, simulation, and experimentation of emerging cloud computing infrastructures and application services. By using CloudSim, researchers can deploy different resources on cloud and examine their

Algorithm 2 Matching Algorithm

Input: ${L_{VM}}^+$, ${L_{PM}}^+$.
$BestFit = -\infty$.
for $\forall v_i \in {L_{VM}}^+$ **do**
  for $\forall p_j \in {L_{PM}}^+$ **do**
    Calculate $Fitness\,(v_i, p_j)$.
    if $Fitness\,(v_i, p_j) > BestFit$ **then**
      $BestFit = Fitness\,(v_i, p_j)$.
    end if
  end for
  if $BestFit >= 0$ **then**
    Select the p_j as the destination PM.
    Migrate v_i.
    ${L_{VM}}^+ - \{v_i\}$.
  end if
end for
if ${L_{VM}}^+ \neq \emptyset$ **then**
  Restart Algorithm 1.
end if

method. Also, it provides performance metrics that evaluate the simulation result, such as energy consumption, number of SLA violations, which are urgent issues in cloud computing.

Our experiment mainly base on the random workload designed from reference [15]. In the experiment environment, there are two types of PM, which are HP ProLiant ML110 G4 (1x[Xeon 3040 1860 MHz, 2 cores], 4GB) and HP ProLiant ML110 G5 (1x[Xeon 3075 2660 MHz, 2 cores], 4GB). Four types of VM are deployed on the PMs with a single core, which are High-CPU Medium Instance (2.5 EC2 Compute Units, 0.85 GB), Extra Large Instance (2 EC2 Compute Units, 3.75 GB), Small Instance(1 EC2 Compute Unit, 1.7 GB) and Micro Instance (0.5 EC2 Compute Unit, 0.633 GB). In the experiment, we considered 100 PMs and 120 heterogeneous VMs in a data center. To illustrate the advantages of our proposed method, the utilization of CPU and bandwidth of one application is generated stochastically, and the utilization of RAM is set as 50%. Also, the evaporation rate in ant colony optimization is set as 0.2 as we not only need to accumulate pheromone in the iteration times but also prevent the quick convergence resulted from the over-accumulated pheromone.

B. Experimental Results

We compare our algorithm with other four algorithms provided by reference [15] which are Inter Quartile Range Random Selection algorithm (IQR_RS) with safe parameter of 1.5, Median Absolute Deviation Random Selection (MAD_RS) with safe parameter of 2.5, Local Regression Random Selection algorithm (LR_RS) with safe parameter of 1.2 and the Static Threshold Random Selection algorithm (THR_RS) with safe parameter of 0.8. We evaluate the performance of our algorithm in terms of three aspects, which are Quality of Service (QoS), number of migrations and load condition variance.

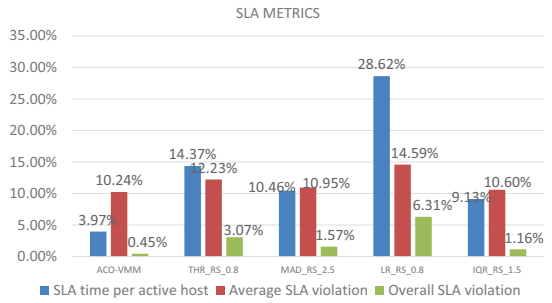

Fig. 3. The comparison of SLA in different algorithms.

1) Quality of Service: To meet the desired QoS is a crucial requirement in cloud computing environment. Usually QoS requirements are formalized in the form of SLAs, which specify enterprise service-level requirements for data center in terms of minimum latency or maximum response time. Beloglazov and Buyya [15] proposed a useful metric called SLAV (SLA Violations) to evaluate the SLA delivered by a VM in an IaaS cloud. It considers both the SLA Violations due to Over-utilization (SLATAH) and SLA Violations due to Performance Degradation in Migrations (PDM). They are set as follows

$$SLATAH = \frac{1}{N}\sum_{i=1}^{N}\frac{T_{s_i}}{T_{a_i}} \tag{16}$$

where N is the number of PMs; T_{s_i} is the total time during which the i-th PM has experienced the utilization of 100% leading to an SLA violation; T_{a_i} is the total of the i-th PM being in the active state.

$$PDM = \frac{1}{M}\sum_{j=1}^{M}\frac{C_{d_j}}{C_{r_j}} \tag{17}$$

where M is the number of VMs; C_{d_j} is the estimate of the performance degradation of the j-th VM caused by migrations; C_{r_j} is the total CPU capacity requested by the j-th VM during its lifetime.

$$SLAV = SLATAH \times PDM \tag{18}$$

The SLATAH and PDM contribute equal importance to the level of SLA violations. Figure 3 shows the SLA violation metrics of each algorithm. We select SLATAH, average SLAV and overall SLAV as main considerations. The result shows that our algorithm has obviously less number of violations than other algorithms.

2) Number of Migrations: Live VM migration is a costly operation that includes some amount of CPU processing on the source PM, the link bandwidth between the source and destination PMs, the down time of the services on the migrating VM, and the total migration time [16]. One of our objectives is to minimize the number of migrations. In our experiment, we use 1Gbps network links. Figure 4 shows the number of VM migrations of each algorithm. Our algorithm

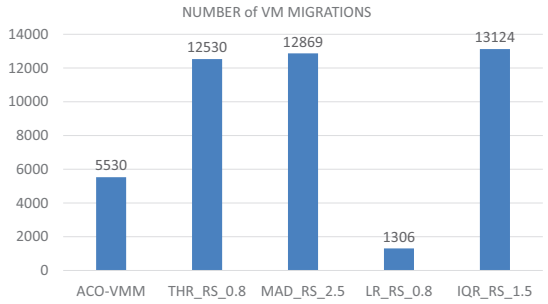

Fig. 4. The comparison of number of migrations in different algorithms.

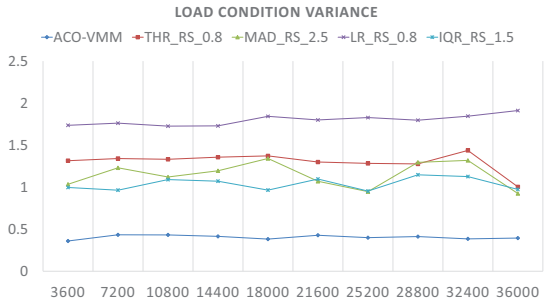

Fig. 5. The comparison of load balance level in different algorithms.

has apparently reduced the number of migrations comparing to other algorithms. Although LR_RS_0.8 has the least number of migrations, it also has worst QoS as in Fig. 3. Instead, our algorithm both reduces the number of migrations and satisfies QoS.

3) Load Condition Variance: As discussed before, the load condition of a data center is evaluated by the variance of the load conditions of all PMs. Figure 5 shows the load condition variance of each algorithm. Each point in the figure stands for the variance of the data center in a given time. In the experiment, we sample the load condition of each PM per 3600 seconds and then calculate the variance. For describing the variation of the variance, we choose 10 continuous samples for each algorithm. The result shows that our algorithm has the least variance, which means that it is more in line with load balancing.

VI. Conclusion and Future Work

In this paper, we propose a distributed VM migration strategy called ACO-VMM with high reliability and scalability. Since finding the best mapping relationship is strictly NP-hard, the ACO-VMM strategy solves the problem of overload balance in data centers by using Ant Colony Optimization strategy to find a near-optimal solution. When compared to the existing migration strategie, ACO-VMM not only reduced the number of migration, but also minimized SLA violations. More importantly, it achieves load balance in a data center.

As a future work, we plan to find better solutions to the problem of overload balance by using improved ACO algorithm, such as the Ant Colony System(ACS) and the MAX-MIN Ant System $M - MAS$. Besides, we intend to improve the threshold from a fixed value to an adaptive value. We also look forward to combine some other heuristic methods with ACO-VMM. There are lots of algorithm which have been proved work well with ACO algorithm like genetic algorithm or simulated annealing algorithm.

Acknowledgment

This project was supported by CCF-Tencent Open Research Fund, the PhD Start-up Fund of Natural Science Foundation of Guangdong Province, China (No. 2014A030310180), the Fundamental Research Funds for the Central Universities (46000-3161006), Undergraduate Innovation Program of Sun Yat-sen University, and Zhuhai Academy of Social Sciences philosophyproject (No. 2014157).

References

[1] J.-H. GU, J.-H. HU, T.-H. ZHAO, and G.-F. SUN, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," 2010, pp. 89–96.

[2] Y. Sahu, R. Pateriya, and R. K. Gupta, "Cloud server optimization with load balancing and green computing techniques using dynamic compare and balance algorithm," 2013, pp. 527–531.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warifield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 164–177, 2003.

[4] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," *Proceedings of the Linux Symposium*, no. 1, pp. 255–230, 2007.

[5] L. Cherkasova, D. Gupta, and A. Vahdat, "When virtual is harder than real: Resource allocation challenges in virtual machine based it environments," *Technical Report HPL-2007-25*, February 2007.

[6] G.-B. WANG, Z.-T. MA, and L. SUN, "On load balancing-oriented distributed virtual machine migration in cloud environment," *Computer Applications and Software*, vol. 30, no. 30, pp. 1–2, 2013.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. 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, pp. 23 – 50, 2011.

[8] Y.-Y. LIU, Q.-Y. GAO, and Y. CHEN, "A method on load balancing of virtual machine resources in virtual computing environment," *Computer Architecture*, vol. 36, no. 16, 2010.

[9] J.-J. LIU, G.-L. CHEN, and C.-X. HU, "Virtual machine migration scheduling strategy based on load characteristic," *Computer Engineering*, vol. 37, no. 17, pp. 276–278, 2011.

[10] X. YANG, Z.-T. MA, and L. SUN, "Performance vector-based algorithm for virtual machine deployment in infrastructure clouds," *Journal of Computer Applications*, vol. 32, no. 11, p. 1, 2012.

[11] Y. PEI, J. WU, and X. WANG, "Virtual machine placement strategy based on particle swarm optimization algorithm," *Computer Engineering*, vol. 38, no. 16, p. 16, 2012.

[12] Z. Lei, J. Xiang, Z. Zhou, F. Duan, and Y. Lei, "A multi-objective scheduling strategy based on moga in cloud computing environment," *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference*, vol. 1, pp. 386 – 391, 2012.

[13] M. Dorigo and M. Birattari, "Ant colony optimization," *Computational Intelligence Magazine, IEEE*, vol. 1, pp. 36–39, 2006.

[14] F. Farahnakian, A. Ashraf, T. Pahikkala, P. Liljeberg, J. Plosila, I. Porres, and H. Tenhunen, "Using ant colony system to consolidate vms for green cloud computing," *Services Computing, IEEE Transactions*, vol. 8, pp. 187 – 198, 2014.

[15] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, http://onlinelibrary.wiley.com/doi/10.1002/cpe.1867/full," 2011.

[16] A. Murtazaev and S. Oh, "Sercon: Server consolidation algorithm using live migration of virtual machines for green computing," *IETE Technical Review*, vol. 28, no. 3, pp. 212 – 231, 2011.