Research on Composite SaaS Placement Problem Based on Ant Colony Optimization Algorithm with Performance Matching Degree Strategy

Zhenzhang Liu[1], Zhihui Hu[1], Lee K. Jonepun[2]
[1]School of Electrical and Information Engineering
Hubei University of Automotive Technology
Hubei, China
[2]Universidad de Tarapacá
Casilla 6-D, Arica, Chile
liuzzlr@163.com

Journal of Digital Information Management

ABSTRACT: *Cloud computing is a term referring to the latest new computing paradigm based on the Internet where a program or application runs on a connected server or servers rather than on a local computing device. Software as a Service (SaaS) is one of the most important computing services in cloud computing. The advantages of SaaS products are well recognized including lower cost of entry, minimal demands on IT and easy upgrades as well as more rigorous. However, this approach has introduced new challenges in SaaS resource management in data centres. One of the challenges is deploying softwares comprised of service and data components over servers in cloud to provide users with required services easily and quickly in SaaS cloud computing. This paper analyzes the SaaS Placement Problem (SPP) and proposes a mathematical model for Composite SaaS placement in cloud. An Ant Colony Optimization Algorithm (ACOA) with performance matching degree Strategy is applied to the problem with the aim of minimizing the total estimated execution time of all the SaaS components and reducing the algorithm computation time. The ACOA is then evaluated against the Genetic Algorithm in experiments and the results show that the ACOA applied to the SPP not only has quicker computation time but also generates better solutions than the Genetic Algorithm (GA).*

Subject Categories and Descriptors
F.2.1 [Analysis of Algorithm and Problem Complexity]: Numerical Algorithm and Problems
H.3.5 [Information Storage and Retrieval]: Online Information Services

General Terms: Cloud Computing, Ant Colony Optimization Algorithm

Keywords: Cloud Computing, SaaS Placement Problem, Ant Colony Optimization, Software as a Service, Virtual Machine

Received: 11 April 2014, Revised 13 May 2014, Accepted 1 June 2014

1. Introduction

Cloud computing is an emerging computing paradigm which allows tenants to easily share configurable computing network resources such as servers, storage and software through a Cloud Service Provider (CSP) allowing companies to focus on their business activities rather than IT functions [1-2].These services can be divided into three categories: 1) Infrastructure as a Service (IaaS) that offers computing infrastructure as a solution to user computing and storage problems, 2) Platform as a Service (PaaS) that targets application developers allowing them to design, develop, deploy and test activities in Cloud platforms, and 3) Software as a Service (SaaS) which offers an alternative for locally installed software [3-4]. These services require different levels of application development and flexibility.

To most users SaaS refers to using software that is not stored on the user's hard drive and is licensed by site, seat, or pay-as-you-go terms with the added benefit of not having to buy additional hardware or software for support

[5]. As a result, the greater demand for SaaS vendors to deploy software services rapidly on huge cloud servers and the allowance of users to hire their required services easily and quickly present significant cloud computing application challenges. The problem of placing SaaS components and their related data in the Cloud is referred to as a SaaS Placement Problem (SPP) [6] while a composite SaaS is defined as software comprised of several software components and data components. The composite SaaS placement problem is to determine where and how each of the components including software and data components should be deployed for optimal performance in a cloud computing environment.

While cloud service providers and software suppliers have been paying considerable attention to SaaS placement, some of these methods were not designed for the cloud while others focused mostly on the resource consumption by the components and not with the placement of the component's data [7-8]. Recently, some heuristic approaches such as Genetic Algorithms (GA) and Evolutionary Algorithms (EA) have been proposed for the composite SaaS placement problem [9-11]. Yusoh et al. [6] proposed the problem formulation and modelling of a composite SaaS placement problem on clusters of Cloud physical servers while other researchers have experimented with genetic algorithms. In such cases a Penalty-based Genetic Algorithm was applied to solve the SaaS optimization problem in such way that it considers not only the placement of the software components of a SaaS, but also the placement of the SaaS data. An evolutionary algorithm was proposed to solve SaaS service comprehensive placement and resource optimization with a feasible and satisfactory solution while an Cooperative Co-Evolutionary Algorithm (CCEA) was applied to the same problem aiming at improving the quality of the solution[9-10]. Based on the experimental results, the proposed algorithms above always produced a feasible and satisfactory solution to all the test problems but the computation time taken is quite long. Ni Z W et al. [11] have attempted to minimize the total estimated execution time of SaaS components using an ant colony optimization (ACO) approach with a better solution.

The rest of the paper is organized as follows. In section 2,the composite SaaS placement model in the cloud and formulates the optimization problem are introduced. Section 3 describes the basic principle of ACOA. Section 4 proposes an algorithm based on ACOA with performance matching degree strategy to solve the SPP. Section 5 evaluates ACOA effectiveness as compared to the GA approach and concluding remarks are presented in Section 6.

2. SAAS Placement Model in Cloud

Three main players in a cloud SaaS scenario are the cloud infrastructure provider, the SaaS vendor and the SaaS customer.In the Infrastructure as a Service model, the provider provides infrastructure to tenants in the form of virtual services and to deliver their services they need to build and deploy a resource pool with vast physical resources such as servers, storage and networks for the tenants. Vendors need to state platform requirements, computation and storage capacity and transmission efficiency according to the demands of the customer and submit the application software and data to the provider who is responsible for hosting and placing the SaaS in the cloud based on an agreement between the vendor and the customer. The composite SaaS has to be placed strategically, as the SaaS components and associated data are dependent on each other. To meet the needs of the three players, application software and related data can be deployed on multiple virtual machines. In turn a virtual machine can place multiple application softwares from SaaS vendors in the form of Service Components (SC) representing particular business functions and data files known as the Data Component (DC). Some components may need to call other components and access data files stored in cloud storage server in the process of execution. An example of such a SaaS placement model is shown in Figure 1.

A primary objective of SPP is to select an appropriate virtual machine for each of the service components and related data components in the SaaS such that desired requirements are satisfied and the SaaS performance is optimal for cloud computing. To find the most appropriate virtual machine to deploy the SaaS service component and related data component, three aspects in the calculation process need to be considered carefully as defined below to understand our use of ACOA for solving this optimization issue [6, 9-11].

Definition 1 Estimated Data Transfer Time (EDTT): EDTT is the estimated time taken for transferring data between storage servers and computing servers. As some of the components may require an access to data in storage servers, EDTT is computed as follows:

$$EDTT(sc_k) = \sum_{V_i, V_j \in E} \frac{AD_{sc_k} \times 8}{B_{p_i, d_j}} + L_{p_i, d_j} \qquad (1)$$

where SC_k represents a component of SaaS service, AD_{SC_K} is the total bytes of amount of read/write task of SC_k, B_{p_i, d_j} is the bandwidth of the link involved, and L_{p_i, d_j} is the latency that may occur in the link.

Definition 2 Estimated Component Processing Time (ECPT): ECPT is the estimated processing time for a component in a selected computing server. It is based on the task size of a component, the processing capability of the selected server and the value of EDTT if there is any. If a component accesses more than one storage servers,only the maximum value of EDTT will be considered. ECPT is computed as follows:

$$ECPT(sc_k) = \frac{TS_{sc_k}}{PC_{sc_k}} + max\ (EDTT) \qquad (2)$$

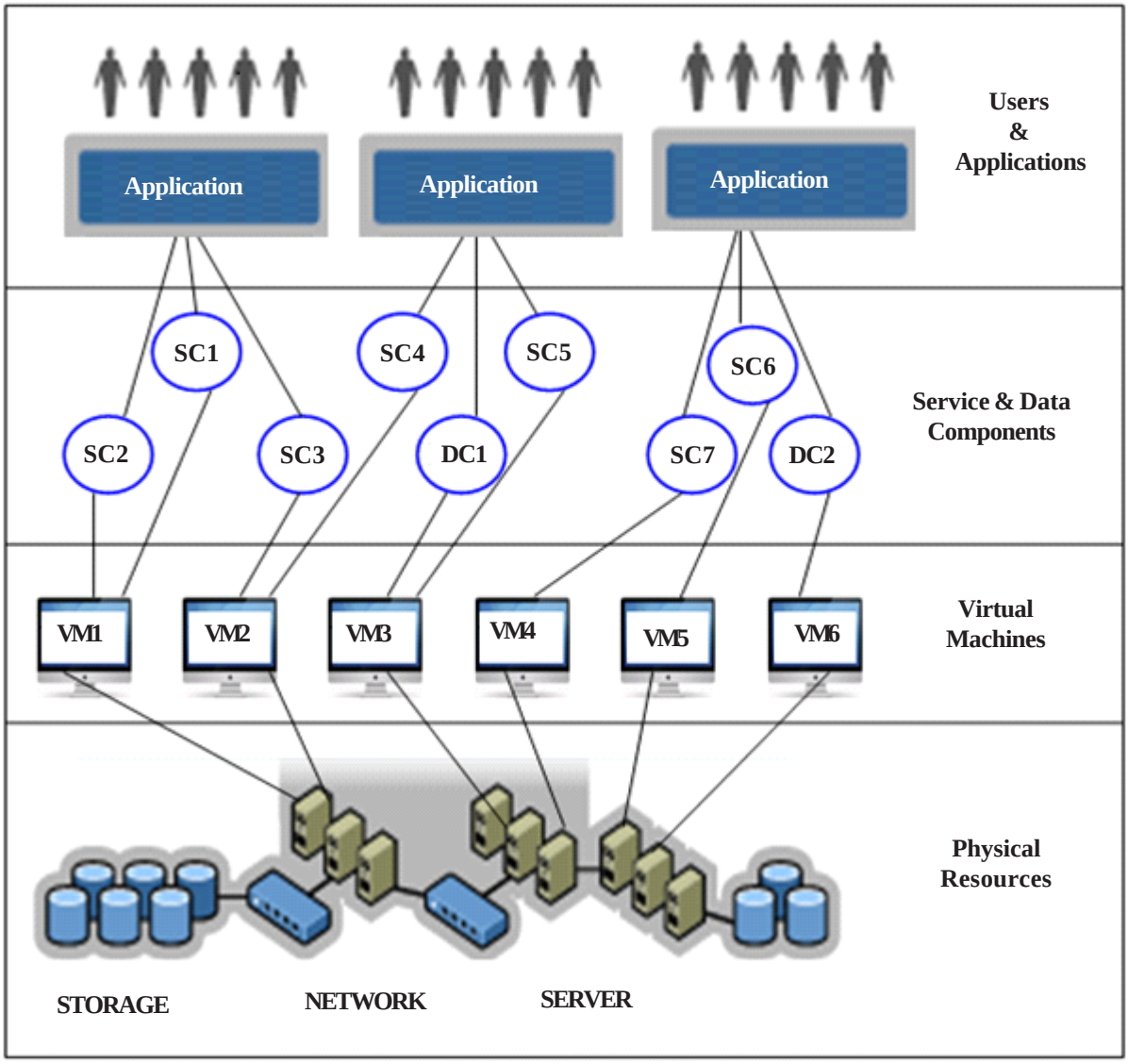

Figure 1. SaaS deployment model in cloud

where TS_{SC_k} is the task size of component SC_k in bytes and PC_{SC_k} represents the processing capability of the computing server in bytes per second.

Definition 3 Estimated Total Execution Time (ETET): ETET is the estimated total execution time for the entire SaaS that is being deployed in the Cloud. ETET is defined by the sum of all components' estimated processing time multiplied by its weighing placement in SaaS as shown in Equation 3.

$$ETET(SaaS) = \sum_{i=1}^{n} ECPT(i) \times W_i \qquad (3)$$

where n is the number of SaaS components and W_i represents the weight value of component SC_i.

3. Ant Colony Optimization Algorithm (ACOA)

Ant colony optimization [12-13] was introduced by Dorigo and colleagues as a novel nature-inspired metaheuristic for the solution of hard Combinatorial Optimization (CO) problems in the early 1990s and from a computational point of view, has proven to be a suitable approach to SPP as a large-scale and complex combinatorial optimization solutions with constraints. The algorithm was developed to simulate the collective foraging behavior of real ants and optimizes a kind of swarm intelligence based on artificial pheromones where the pheromone amount left by the virtual foraging ants is calculated by largest resulting in a path that the ants will most likely choose to achieve their goal. The algorithm was first used to solve the Traveling Salesman Problem (TSP) and now it is widely used in solving combinatorial optimization problems such as job-shop scheduling, network routing, vehicle routing problems and the like [13-15]. The algorithm model is as follows [16-17]:

If m is number of ants in the ant colony, $P_{ij}^{k}(t)$ stands for the probability of ant k select path from v_i to v_j.

$$P_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}(t)]^{\beta}}{\sum_{s \in allowed_k} [\tau_{is}(t)]^{\alpha} \cdot [\eta_{is}(t)]^{\beta}}, & if\ j \in allowed_k \\ 0, & otherwise \end{cases} \qquad (4)$$

where assume v_i ($i = 1, 2,..., n$) stands for a node on path of ants foraging, $\tau_{ij}(t)$ is pheromone concentration on path from v_i to v_j, $allowed_k$ is ant k current can select nodes set, $\eta_{ij}(t)$ stands for expectation of ant from v_i to v_j which is also called the heuristic factor, and assume $\eta_{ij} = 1 / d_{ij}$, d_{ij} ($i, j = 1, 2,..., n$) stands for the distance between adjacent nodes v_i and v_j. The values of parameters α and β, $\alpha > 0$ and $\beta > 0$, determine the relative importance of pheromone value and heuristic factor.

When an ant has traversed all the nodes, it completes a cycle and the path traversed is a solution of the problem. At this time, for each path pheromone update:

$$\tau_{ij}(t + n) = (1 - \rho).\ \tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{5}$$

where ρ stands for the degree of pheromone decreases with time, $\rho \in (0, 1)$ and $\Delta\tau_{ij}(t)$ stands for the pheromone increment.

Depending on the pheromone update strategy, Dorigo stipulated three basic ant colony algorithm models as following:

1. Art-Cycle model

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{L_k}, & \text{if ant } k \text{ passes } (i, j) \text{ in the cycle} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where Q stands for information intensity and L_k is the total path length traversed in the loop of ant k.

2. Ant-Quantity model

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{d_{ij}}, & \text{if ant } k \text{ passes } (i, j) \text{ between } t \text{ and } t + 1 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

3. Ant-Density model

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q & \text{if ant } k \text{ passes } (i, j) \text{ between } t \text{ and } t + 1 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

The differences in these models is that formula (7) and formula (8) use local information to update the pheromone increment while formula (6) makes use of the overall information to update the pheromone increment after the ant completes a cycle path and has good performance in solving TSP. In practical applications, it is necessary to select a different pheromone updating strategy according to a concrete a mathematical model to achieve an optimal effect.

4. The ACOA for Saas Placement Problem

As the aim of SPP is to achieve the optimal performance of the SaaS, all the service components and their associated data components must be deployed to an appropriate virtual machine to obtain minimum total execution time of SaaS. Part of the SPP network model is a directed graph $G = (V, E)$, where V is the set of vertices which consists of components including service components and data components and virtual machines. Each virtual machine has its own attributes that are relevant to the SaaS requirements including computation capacity, storage capacity and memory capacity. Each SaaS component has resource requirements such as computation capacity, memory requirements and storage requirements. E stands for the set of edge such as (v_i, v_j), where v_i is a node which represents a component, v_j represents a node standing for a virtual machine, and the edge e, (v_i, v_j), represents the deployed component i on the virtual machine j.

In the cloud environment, various dependent service components and data components are deployed to virtual machines where m is the number of dependent service components and data components and n is the number of virtual machines in cloud. For algorithm description and implementation convenience, $SC = \{SC_1, SC_2, \ldots SC_m\}$ stands for the set of service and data components. $SCP = \{SCP_1, SCP_2, \ldots SCP_m\}$ stands for the requirement for computation capability of each component although a data component does not require any computation capability. $SCM = \{SCM_1, SCM_2, \ldots SCM_m\}$ stands for the storage capacity needs of each component. $VM = \{VM_1, VM_2, ... VM_n\}$ stands for the set of virtual machines, $VMP = \{VMP_1, VMP_2, ... VMP_n\}$ represents computation capability of each virtual machine, $VMM = \{VMM_1, VMM_2, ... VMM_n\}$ stands for the storage capacity of each virtual machine, and $allowed = \{allowed_1, allowed_2, ... allowed_m\}$ is the set of assignable virtual machines which components can be deployed to. The set of assignable virtual machines of each component will change according to the deployment situation. $W = \{W_1, W_2, ... W_m\}$ stands for the importance weights of each component in SaaS.

To improve the efficiency of algorithm, we assume that the number of ants in ant colony algorithm is equal to the number of components to be deployed in SaaS, τ_{ij} stands for the pheromone intensity on path (v_i, v_j) and its initial value is set to a fixed value of Q. Pheromone increase is calculated after each deployment of a component according to the Formula as follows: $\Delta\tau_{ij} = Q/ECPT_{ij}$, where $ECPT_{ij}$ is the estimated processing time of SC_i on VM_j.

η_{ij} is a heuristic factor of the ant colony algorithm to express the expectation of SC_i deployment in VM_j, In order to express the η_{ij}, we introduce the following concept:

Definition 4 Performance Matching Degree (PMD): stands for proximity degree between the component resource requirement and virtual machine performance. The PMD

is computed as follows:

$$PMD_{ij} = \sqrt{(VMP_j - SCP_i)^2 + (VMM_j - SCM_i)^2} \quad (9)$$

Where assume $\eta_{ij} = PMD_{ij}$, the algorithm expects to deploy a component to the virtual machine with best performance aiming at minimizing execution time of the component.

In the first round placement, each of the components randomly select a virtual machine from an allowed virtual machine table for deployment with each round of deployment calculating all choice probabilities P_{ij} between components and virtual machines according to formula (4), deploying each component to virtual machine accordance with the size of P_{ij}.

A placement scheme results when all components are deployed to the appropriate virtual machines in accordance with the ACOA; each virtual machine meets the computation needs and storage requirements of components deployed and the related information of this deployment scheme are saved. $SCV = \{SCV_1, SCV_2, SCV_m\}$ stands for the virtual machines on which components are deployed. The total estimated execution time of this deployment scheme can be expressed as formula (10):

$$ETET(SaaS) = \sum_{i=1}^{m} ECPT_{i, SCV_i} \times W_i \quad (10)$$

The ACOA for Composite SaaS calculates the time spent on finding the placement solution for each of the configurations and counts the minimal/maximal and average values of estimated total execution time of all deployment schemes.The algorithm is described as below:

Initialization algorithm constructs assignable virtual machine tables for all components:

for $i = 1$ *to* m {
  for $j = 1$ *to* n {
    if $VMP_j - SCP_i >$ *threshold AND* $VMM_j - SCM_i >$ *threshold*
      VM_j *join allowed*$_i$ *which stands for assignable virtual machine table of component* SC_i *;*
    end if
  }
}

In order to ensure the vitality of virtual machine, we introduce the concept of threshold. If the difference between existing capabilities of a virtual machine and a component requirement is below the threshold, the component cannot be deployed to the virtual machine.

In the first round placement, each component is randomly deployed to a virtual machine which meets the capacity requirement of the component. The first round placement *algorithm is described as follows:*

for $i = 1$ *to* m {
  randomly select a virtual machine VM_j *from assignable virtual machine table of component* SC_i *and deploy* SC_i *on* VM_j*;*
  $ETET = ETET + ECPT_{i,j} * W_i$*;*
  // reduce the available resources in virtual machine VM_j
  $VMP_j = VMP_j - SCP_i$*;*
  $VMM_j = VMM_j - SCM_i$*;*
  Adjust assignable virtual machine table of un-deployed components;
  $\Delta\tau_{ij} = Q/ECPT_{ij}$*;*
  $\tau_{ij} = (1-\rho)\ \tau_{ij} + \Delta\tau$*;*
}

In the later rounds of placement, select a component SC_i in turn according to the chosen probability P_{ij} and deploy it on the virtual machine VM_j. The Algorithm is described as follows:

for $NC = 2$ *to* NC_{max}
{
calculate the chosen probability P_{ij} *of the assignable virtual machine according to the formula (4);*
for $i = 1$ *to* m {
  select component SC_i *according to the size of* P_{ij} *from big to small and deploy it on the virtual machine* VM_j*;*
  $ETET = ETET + ECPT_{i,j} * W_i$*;*
  // reduce the available resource of virtual machine VM_j
  $VMP_j = VMP_j - SCP_i$*;*
  $VMM_j = VMM_j - SCM_i$*;*
  adjust assignable virtual machine table of un-deployed components;
  $\Delta\tau_{ij} = Q/ECPT_{ij}$*;*
  $\tau_{ij} = (1-\rho)\ \tau_{ij} + \Delta\tau$*;*
 }
}

5. Evaluation

The ACOA to solve the SPP in cloud environment described above has been implemented and tested. The programming language used was visual C in a windows environment. All the experiments were conducted in a desktop computer with 3.4 GHz Intel Core i7 CPU 2600 K and 8 GB RAM. Server and link parameter settings are as follows: link bandwidth is 10 Mbit/S, link latency is the distance value from node v_i to v_j, AD_{SC_k} values are randomly from range of 100 to 200 M, TS_{SC_k} values are randomly from range of 20 to 200 M, PC_{SC_k} is fixed at 10 Gb/S.

The experiments were designed to evaluate the speed-up ratio and the quality of the solutions of the ACOA when it is used for solving different sizes of composite SaaS

placement problems and to study how the computation time of the ACOA and components total estimated execution time would increase when the size of the SPP increases. Furthermore, in order to evaluate the performance of ACOA in solving the SPP problem we apply the Genetic Algorithm (GA) for the same problem and compare the solutions produced by the ACOA with those produced by the GA based on the ETET of the SaaS [18-19].

The algorithm for solving the SaaS placement problem based on ACOA was divided into two cases. In the first case, we assumed the number of virtual machines to be fixed and the number of components of SaaS service gradually increasing and respectively carried out ACOA and GA to deploy components in SaaS. We recorded algorithm computation times and calculated all components total estimated execution times in two

VM	SC	DC	Algorithm	ETET(s)			Computation Time (s)		
				Min	Max	Ave	Min	Max	Ave
800	10	10	ACOA	1209.4	1498.2	1360.2	247.4	265.7	258.7
			GA	1228.7	1521.9	1381.9	275.8	285.9	277.8
800	20	20	ACOA	1434.5	1706.7	1588.5	394.3	421.4	409.3
			GA	1490.2	1773	1650.2	401.1	425.5	414.1
800	30	30	ACOA	1805.8	2157.5	1878.2	496.0	521.8	512.1
			GA	1868.4	2232.3	1943.3	533.7	565.3	555.6
800	40	40	ACOA	1680.5	1864.1	1743.4	541.8	578.0	562.7
			GA	1726.2	1914.8	1790.8	578.2	621.1	605.1
800	50	50	ACOA	1723.4	1915.6	1788.6	625.3	674.2	651.4
			GA	1787.9	1987.3	1855.5	673.5	724.4	700.4

Table 1. Experimental Results of the ACOA and GA on the Number of SaaS Components

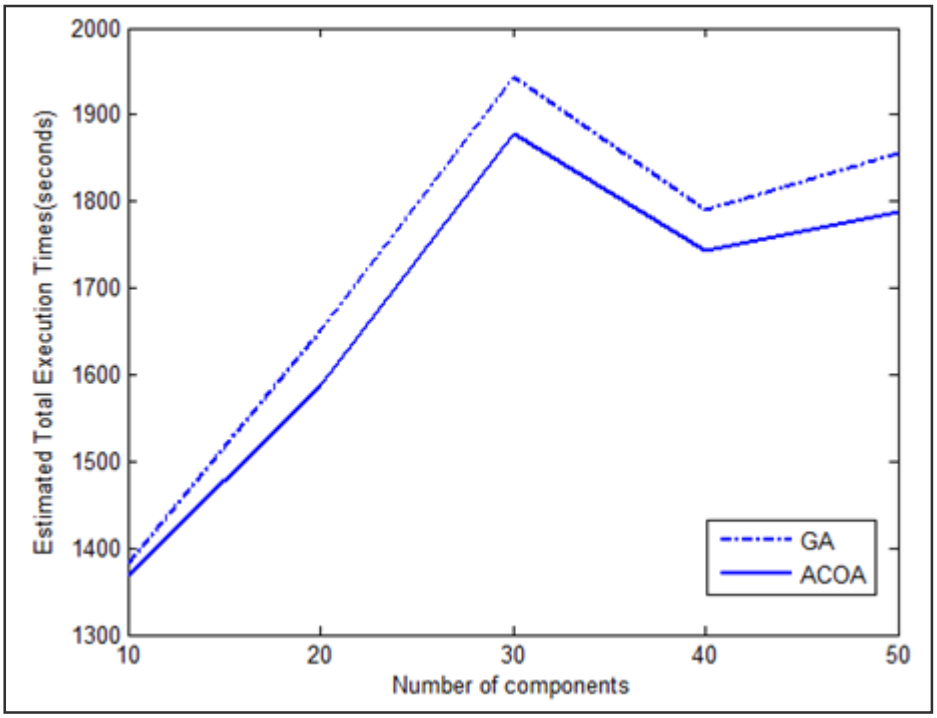

Figure 2. Comparison of the ETET Obtained by the ACOA and the GA on the Number of Components

algorithms to find the increasing trend of the components total execution time of ACOA and GA when the number of software components and data components increases. In the second case, we assumed the component number as unchanged and the number of virtual machines in the cloud gradually increasing, respectively applied the ACOA and GA, recorded the algorithms computation times and calculated all components estimated total execution times.

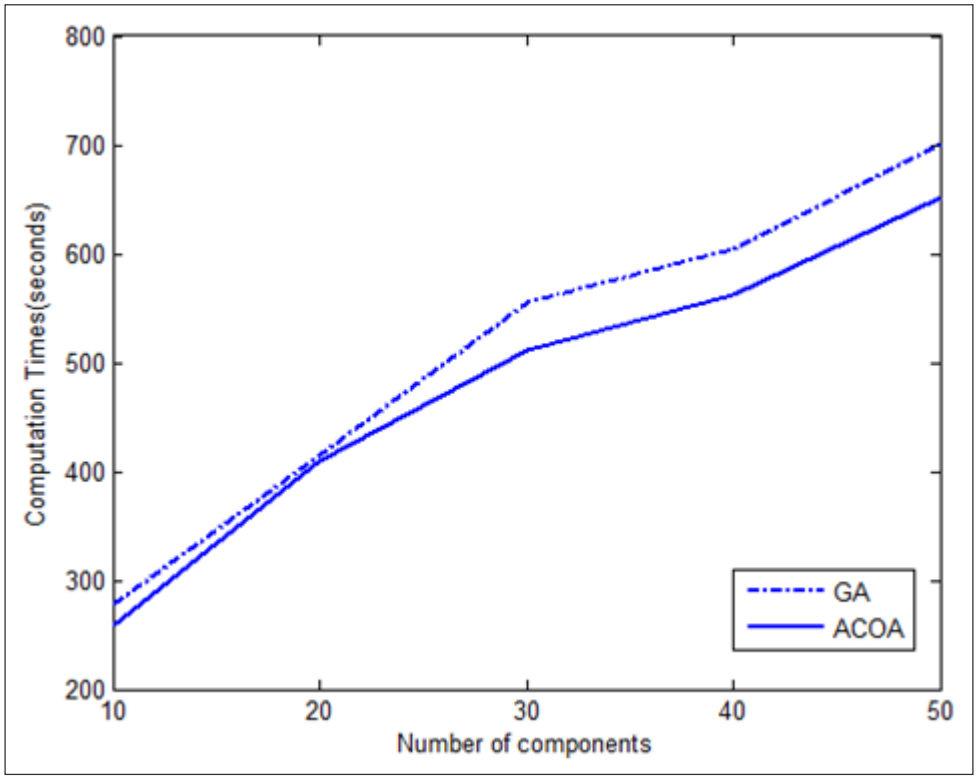

Figure 3. Comparison of the Computation Times Obtained by the ACOA and the GA on the Number of Components

VM	SC	DC	Algorithm	ETET (s)			Computation Time (s)		
				Min	Max	Ave	Min	Max	Ave
300	30	30	ACOA	967.8	1098.1	1035.1	48.6	58.8	54.5
			GA	996.8	1131.7	1066.2	50.4	68.0	57.6
600	30	30	ACOA	1046	1233.2	1115.3	315.3	340.5	329.9
			GA	1088.4	1270.4	1148.8	334.6	365.7	351.8
900	30	30	ACOA	1125.9	1280.7	1214.8	645.4	703.5	679.9
			GA	1189.7	1319.1	1251.2	688.2	755.9	725.3
1200	30	30	ACOA	1144.7	1299.5	1233.2	1202.1	1298.6	1251.9
			GA	1196.8	1338.5	1270.2	1282.7	1381.9	1335.4
1500	30	30	ACOA	1243.1	1361.2	1301.1	1663.5	1796.2	1732.5
			GA	1298.4	1402.3	1340.5	1865.3	1903.1	1847.5.

Table 2. Experimental Results of the ACOA and GA on the Number of Virtual Machines

5.1 Experiments on the Nnumber of SaaS Components

Two sets of experiments for the above two cases were conducted. The parameters setting for ACOA and GA and results of the experiment in the first set of experiments are listed in Table 1. In this experiment, the number of virtual machines in SaaS was fixed at 800 while the numbers of service components and data components were set from 10 to 50 with an increment of 10. Considering the nature of the ACOA and GA, each of the test cases was repeated 10 times. The test results include the minimal/maximal and average values of the components total estimated execution time of these tests.

Figure 2. displays how the average values of the components total estimated execution time of the composite SaaS placement produced by the ACOA and GA increases when the size of the composite SaaS increases.

Figure 3 displays how the average computation times of the ACOA and GA increased when the size of the composite SaaS increased.

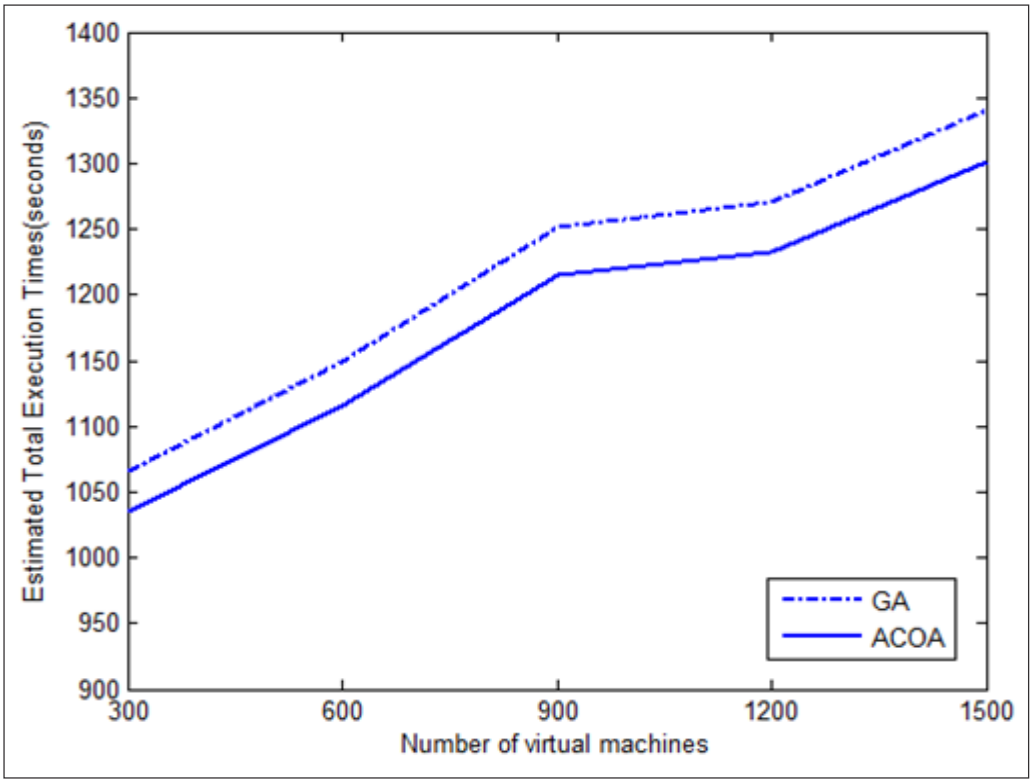

Figure 4. Comparison of the ETET Obtained by the ACOA and the GA on the Number of Virtual Machines

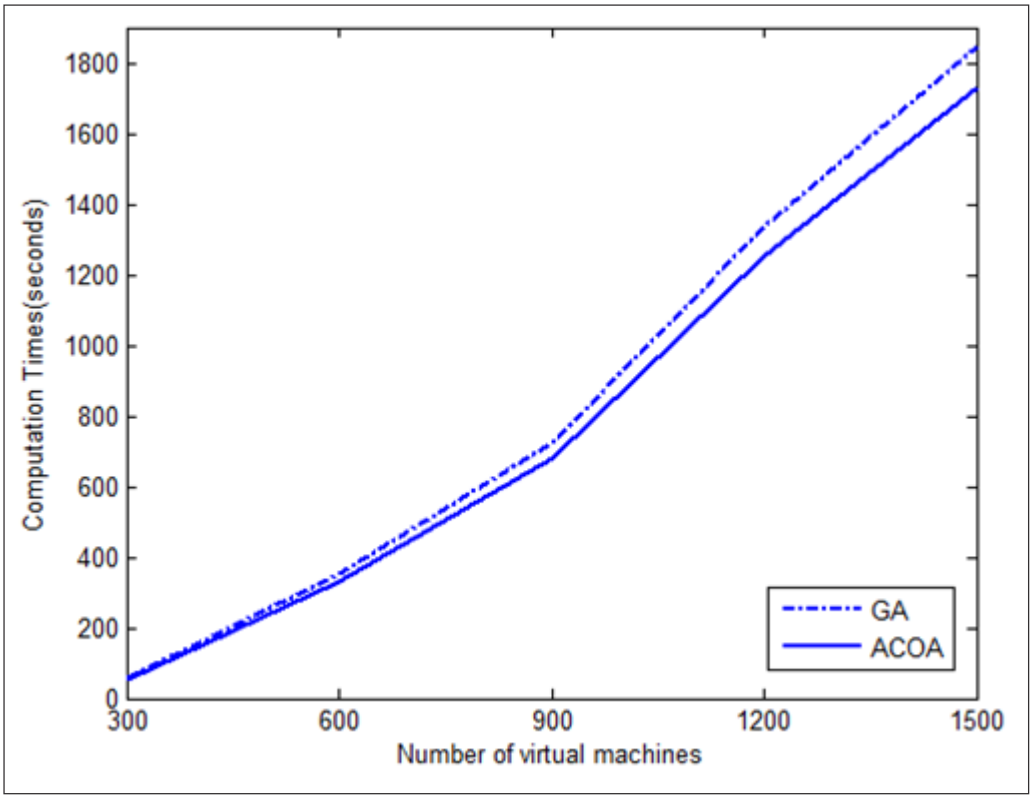

Figure 5. Comparison of the Computation Times Obtained by the ACOA and the GA on the Number of Virtual Machines

From the experimental results it can be seen that not only the average values of the components total estimated execution time of the composite SaaS placement produced by the ACOA is superior to that produced by the conventional GA but also the average computation time of the ACOA is less than that of the GA. As shown in Figure.2, on the basis of the steady number of virtual machines, the components total estimated execution time and the algorithm computation time increases with the increase in the Service and data components. The

experimental results show that the SaaS vendor should give full consideration to the effect of increasing the number of components on the overall performance of SaaS.

5.2 Experiments on the Number of Virtual Machines

In these experiments, we set the number of service components and data components at 30 with the number of virtual machines in the cloud gradually increasing with an increment of 300 from 300 to 1500. As the case with the previous experiments, each of the test cases was repeated 10 times. The parameters setting for ACOA and GA and results of the experiment in this set of experiments is listed in Table 2.

Figure 4 displays how the average values of the components total estimated execution time of the composite SaaS placement produced by the ACOA and GA increases when the number of virtual machines increases.

Figure 5 displays how the average computation times of the ACOA and GA increased when the number of the virtual machines increased.

From the experimental results,Although the ACOA is superior to GA in solving the composite SaaS placement problem,the average value of algorithm computation time appears to grow nearly linearly with the number of the virtual machines.The experimental results show that during the placement of SaaS we should control the scale of virtual machines while the physical resources remain unchanged in order to obtain a fast response speed of SaaS service to meet users' needs better.

6. Conclusion

This paper analyzes the composite SaaS placement problem in a cloud and presents a mathematical model that minimizes the total estimated execution time of SaaS components by proposing an ACOA with performance matching degree Strategy to solve the problem. The ACOA is designed in such way that it considers not only the placement of service components of a SaaS but the placement of related data components of the SaaS as well. The results of simulation experiment illustrate that the components' total estimated execution time appears to grow nearly linearly with the number of the virtual machines while physical resources remain unchanged. At the same time, the SaaS vendors should give full consideration to the effect of increasing the number of components on the overall performance of SaaS because the components total estimated execution time increases with the increase in the Service and data components. In addition the experimental results show that the components' total estimated execution time of the composite SaaS placement produced by the ACOA is superior to that produced by the conventional GA with the computation time of the ACOA less than that of the GA. In conclusion ACOA can be applied effectively to the composite SaaS Placement Problem to improve the response speed of SaaS services better to meet user needs.

References

[1] Madhavaiah C, Bashir I, et al. (2012). Defining cloud computing in business perspective: a review of research. *The Journal of Business Perspective*, 16 (3) 163-173.

[2] Miao, K, X., He, J. (2012). Cloud computing and open data centers. *Intel® Technology Journal*, 16 (4) 8-18.

[3] Alali F A, Yeh C L. (2012). Cloud computing: overview and risk analysis. *Journal of Information Systems*, 26 (2) 13-33.

[4] Cusumano, M. (2010). Cloud computing and SaaS as new computing platforms. *Communications of the ACM*, 53 (4) 27-29.

[5] Baldwin, M., Cromity, J. (2012). SaaS and Cloud computing, the rise of compartmentalizing users online via subscription. *New Review of Information Networking*, 17(2) 120-126.

[6] Yusoh, Z I M, Tang, M. (2010). A penalty-based genetic algorithm for the composite SaaS placement problem in the cloud. Evolutionary Computation (CEC), IEEE Congress on, IEEE, p. 1-8.

[7] Karve, A., Kimbrel, T, et al. (2006). Dynamic placement for clustered web applications. *In*: Proceedings of the 15th international conference on World Wide Web, ACM, p. 595-604.

[8] Kwok, T., Mohindra, A. (2008). Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications. Service-Oriented Computing–ICSOC 2008, Springer Berlin Heidelberg, p. 633-648.

[9] Yusoh Z I M, Tang M. (2012). Composite saas placement and resource optimization in cloud computing using evolutionary algorithms. Cloud Computing (CLOUD), IEEE 5th International Conference on, IEEE, p. 590-597.

[10] Yusoh, Z I M., Tang, M. (2010). A cooperative coevolutionary algorithm for the composite SaaS placement problem in the cloud. Neural Information Processing, Theory and Algorithms, Springer Berlin Heidelberg, p. 618-625.

[11] Ni, Z. W, Pan, X. F, et al. (2012). An Ant Colony Optimization for the Composite SaaS Placement Problem in the Cloud. *Applied Mechanics and Materials*, p. 130 3062-3067.

[12] Dorigo, M., Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical computer science*, 344 (2) 243-278.

[13] Stützle, T., Dorigo, M. (1999). ACO algorithms for the quadratic assignment problem. New ideas in optimization, p. 33-50.

[14] Dorigo, M., Gambardella, L M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation*, *IEEE Transactions*, 1 (1) 53-66.

[15] Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2 (4) 353-373.

[16] Dorigo, M., Maniezzo, V, et al. (1996). Ant system: optimization by a colony of cooperating agents. Systems, Man, and Cybernetics, Part B: *Cybernetics, IEEE Transactions*, 26 (1) 29-41.

[17] Dorigo M, Caro G, et al. (1999). Ant algorithms for discrete optimization. *Artificial life*, 5 (2) 137-172.

[18] Zhao, C., Zhang, S, et al. (2009). Independent tasks scheduling based on genetic algorithm in cloud computing. Wireless Communications, Networking and Mobile Computing, WiCom'09. 5th International Conference on. IEEE, p. 1-4.

[19] Kaur, S., Verma, A. (2012). An efficient approach to genetic algorithm for task scheduling in cloud computing environment. *International Journal of Information Technology and Computer Science* (IJITCS), 4 (10) 74-79.