An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing

Xiao-Fang Liu, *Student Member, IEEE*, Zhi-Hui Zhan, *Member, IEEE*, Jeremiah D. Deng, *Member, IEEE*, Yun Li, *Member, IEEE*, Tianlong Gu, and Jun Zhang, *Fellow, IEEE*

***Abstract*—Virtual machine placement (VMP) and energy efficiency are significant topics in cloud computing research. In this paper, evolutionary computing is applied to VMP to minimize the number of active physical servers, so as to schedule underutilized servers to save energy. Inspired by the promising performance of the ant colony system (ACS) algorithm for combinatorial problems, an ACS-based approach is developed to achieve the VMP goal. Coupled with order exchange and migration (OEM) local search techniques, the resultant algorithm is termed an OEMACS. It effectively minimizes the number of active servers used for the assignment of virtual machines (VMs) from a global optimization perspective through a novel strategy for pheromone deposition which guides the artificial ants toward promising solutions that group candidate VMs together. The OEMACS is applied to a variety of VMP problems with differing VM sizes in cloud environments of homogenous and heterogeneous servers. The results show that the OEMACS generally outperforms conventional heuristic and other evolutionary-based approaches, especially on VMP with bottleneck resource characteristics, and offers significant savings of energy and more efficient use of different resources.**

***Index Terms*—Ant colony system (ACS), cloud computing, virtual machine placement (VMP).**

I. Introduction

CLOUD computing is a large-scale distributed computing paradigm, driven by an increasing demand for various levels of pay-per-use computing resources [1]. Cloud facilitates three major types of services to the customer via the Internet. Infrastructure as a service for hardware resources, such as Amazon Elastic Compute Cloud. Platform as a service for a runtime environment, such as Google App Engine. Software as a service, such as Salesforce.com [2], [3]. These services are offered mainly through virtualization [4]. This way, the physical resources are virtualized as uniform resources and therefore are efficient for parallel and distributed computing [5], [6]. Virtual machines (VMs) are created according to the type of operating system and the amount of required resources such as CPU, memory, and storage, specified by the customers and then run on a physical server to host application to meet requirements of customers [7], [8]. On the other hand, virtualization allows multiple VMs to be executed on the same physical server and share hardware resources. This enables VM consolidation, which allocates the maximum number of VMs in the minimum number of physical servers [9]. The unused servers can be switched off to cut the cost for cloud provider and customers.

With a rapid growth in the number and size of cloud data centers [10], the energy consumption, as well as equipment cooling costs has risen to new highs [10]. Studies have shown that data centers around the world consumed 201.8 TWh of electricity in 2010, enough to power 19 million average U.S. households [12]. This consumption accounted for 1.1%–1.3% of the worldwide total and the rate was expected to increase to 8% by 2020 [13]. At present, most cloud servers utilize between 11% and 50% of their total resources most of time [14]. The power consumed by an active but idle server is at the ratio between 50% and 70% of a fully utilized server [15]. Therefore, placing the VMs of a lowly utilized server onto other servers and gracefully schedule down the lowly utilized server will efficiently reduce the power consumption.

The consolidation of VMs has an implication in energy efficiency. This leads to a VM placement (VMP) problem, a computational problem that seeks to obtain an optimal deployment of VMs onto physical servers [16], [17]. Various methods have been reported in the literature for VMP according to different objectives, such as energy efficiency of the physical servers that are used to host the VMs by optimizing the assignment of VMs [15], maximization of the resource utilization ratio of the physical servers through VM consolidation [19], and load balancing on different physical servers to improve the overall system efficiency [20], [21]. Further, a guideline of

Manuscript received November 17, 2015; revised March 10, 2016, June 22, 2016, and October 5, 2016; accepted October 25, 2016. Date of publication November 21, 2016; date of current version January 26, 2018. This work was supported in part by the National Natural Science Foundations of China (NSFC) under Grant 61402545, in part by the Natural Science Foundations of Guangdong Province for Distinguished Young Scholars under Grant 2014A030306038, in part by the Project for Pearl River New Star in Science and Technology under Grant 201506010047, in part by the GDUPS (2016), and in part by the NSFC Key Program under Grant 61332002. *(Corresponding authors: Zhi-Hui Zhan; Jun Zhang.)*

X.-F. Liu, Z.-H. Zhan, and J. Zhang are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: zhanapollo@163.com; junzhang@ieee.org).

X.-F. Liu is also with the Department of Computer Science, Sun Yat-sen University, Guangzhou 510006, China.

J. D. Deng is with the Department of Information Science, University of Otago, Dunedin 9054, New Zealand.

Y. Li is with School of Computer Science and Network Security, Dongguan University of Technology, Dongguan 523808, China.

T. Gu is with the School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China.

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.orgprovidedbytheauthors.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2016.2623803

VMP mechanisms for backup (snapshots of each VM) and working VMs to support a disaster-resilient cloud has been proposed in [22].

For the energy efficiency objective, the VMP problem is an NP-hard problem [23]–[25]. This VMP problem was first solved as a linear programming (LP) problem. For example, stochastic integer programming was used to minimize the cost for hosting VMs in a multiple cloud provider environment [26]. In [27], a server consolidation problem is also formulated as an LP problem, solved with heuristics for a minimized server cost. Using a VM mixed integer LP model, Lawey *et al.* [4] proposed a framework for designing energy efficient cloud computing services over nonbypass IP/wavelength division multiplexing core networks. They adopted an approach slicing the VMs into smaller VMs and placing them in a proximity to their users so as to minimize the total power consumption.

In comparison, heuristic methods have offered higher efficiency in solving the VMP problem. In particular, evolutionary computation (EC) algorithms [28] such as genetic algorithm (GA) [29] have been used to improve resource utilization and reduce energy consumption. A modified GA with fuzzy multiobjective evaluation was developed for the VMP in [30]. Wang *et al.* [31] designed an improved GA to maximize resource utilization, balance multidimensional resources, and minimize communication traffic. Wilcox *et al.* [32] modeled the VMP problem as a multicapacity bin packing problem so as to find an optimal assignment homogeneously problem to simplify the VMP with the often heterogeneous servers in cloud data centers. Foo *et al.* [33] proposed to use a GA to optimize the neural network, so as to forecast and reduce energy consumption in cloud computing.

As the VMP problem can be regarded as a combinatorial optimization problem (COP) [44], [45], many EC algorithms may be applicable [28]. EC algorithms have been successfully applied to many COPs, such as protein structure prediction [46], music composition [47], multiple sequence alignment [48], distribution network restoration [49], constrained optimization [50], scheduling problems [51], and haystack problem [52], and have shown promising performance. However, among the EC algorithms, the ACO paradigm [53], [54], especially its ant colony system (ACS) variant, fits COPs better and has shown particular strengths in solving real-world COPs. Compared with other EC algorithms, such as a GA and particle swarm optimization (PSO) [55], [56], the adoption of global shared pheromone in ACS allows the experience information to be spread rapidly among the colony and thus help the cooperation among multiple ants. Moreover, the introduction of heuristic information enhances the exploration capacity. The balance of exploration of new solution and exploitation of accumulated experience about the problem ensure fast convergence and good performance of ACS. Therefore, the ACS-based algorithm for VMP optimization is extensively studied in this paper. In cloud computing domain, Feller *et al.* [11] applied an ACO-based approach to minimize the number of cloud servers to support current load. However, this method has a high computing cost and consolidates VMs only on a single resource. In this paper, we consolidate VMs according to multiple resources (i.e., both CPU and memory), being more applicable in cloud computing, but more challenging [34]. There also exist some reports on the use of multiobjective algorithms to minimize the total resource wastage and power consumption. These algorithms include multiobjective ACS [35], multiobjective ACO (MACO) [36], and hybrid ACO with PSO, termed HACOPSO [37]. In [38], an ACS is employed for VMs consolidation in dynamic environment to reduce energy consumption but not directly to reduce the number of servers.

In this paper, we develop an ACS-based approach to allocating the VMs in minimum number of physical servers to reduce energy consumption for cloud computing. This is a substantially extension of a feasibility study in our early work [39] on consolidating VMs and has significant differences, not only from the algorithm design aspect, but also from the experimental environment aspect. To handle both homogeneous and heterogeneous server environment, we also develop an order exchange and migration (OEM) mechanism for the ACS, resulting in an OEMACS algorithm. Further, the OEMACS algorithm incorporates a new solution evaluation method with a hierarchical structure.

The remainder of this paper is organized as follows. Section II outlines a model of the VMP problem and an ACS approach. Section III develops the OEMACS algorithm in detail. It is then applied in Section IV to various cloud computing environments, including 17 homogeneous and five heterogeneous server VMP scenarios with bottleneck resources. Experiments are undertaken to validate the effectiveness and efficiency of OEMACS, by comparing not only with heuristic approaches such as first-fit decreasing (FFD) [40], but also with EC-based approaches such as reordering grouping GA (RGGA) [32], ACO-based method [11], multiobjective MACO [36], and HACOPSO that hybrids ACO and PSO [37]. Finally, the conclusions are drawn in Section V.

II. Background

A. Virtual Machine Placement

Virtualization lies in the heart of cloud computing. Once the cloud data center receives an application request from a customer, a VM is created to host the application based on the required resources (CPU, memory, and storage) and the type of operating system specified by the customer. Then, the VM is assigned to one available server according to the placement strategy. How to assign the VMs to suitable servers to reduce the energy consumption is an important problem. This paper tracks the VMP for minimizing the number of active servers and presents a special case in which VMs are consolidated with respect to two resources: 1) CPU and 2) RAM, referring to the fact that many cloud applications such as Google AppEngine hosts mostly e-commerce applications and charges the users based on their CPU consumption [19], [41]. Assume that there are N VMs and M servers. Let $V = \{1, 2, \ldots, N\}$ be a set of VMs, and $P = \{1, 2, \ldots, M\}$ a set of servers. As we consider the two most representative resources (i.e., CPU and RAM) for a VM, we use symbols

vc_j and vm_j to represent the CPU and RAM requirements of VM$j \in V$, respectively. Similarly, CPU and RAM capacities of server $i \in P$ are denoted as PC_i and PM_i, respectively. Since VM is not allowed to be assigned across servers, we assume that none of the VMs requires more resources than the capacity of a single server. We aim to achieve an assignment with a minimum number of active servers to improve resource utilization and energy efficiency. In a placement solution S, each VM is placed onto one and only one server. The assignment is attributed by a zero-one adjacency matrix $\mathbf{X}$, where element x_{ij} represents whether VMj is assigned to server i. If VMj is placed on server i, then $x_{ij} = 1$, otherwise $x_{ij} = 0$. Each server must have enough resources to satisfy the demand of all VMs on it. The VMP problem for minimizing the number of active servers is formulated as

$$\text{minimize } f(S) = \sum_{i=1}^{M} y_i \tag{1}$$

subject to

$$x_{ij} = \begin{cases} 1, \text{ if VM}j \text{ is assigned to server } i \\ 0, \text{ otherwise} \end{cases} \forall i \in P \text{ and } \forall j \in V \tag{2}$$

$$y_i = \begin{cases} 1, \text{ if } \sum_{j=1}^{N} x_{ij} \geq 1 \\ 0, \text{ otherwise} \end{cases} \quad \forall i \in P \tag{3}$$

$$\sum_{i=1}^{M} x_{ij} = 1 \qquad \forall j \in V \tag{4}$$

$$\sum_{j=1}^{N} \text{vc}_j \cdot x_{ij} \leq \text{PC}_i \cdot y_i \quad \forall i \in P \tag{5}$$

$$\sum_{j=1}^{N} \text{vm}_j \cdot x_{ij} \leq \text{PM}_i \cdot y_i \quad \forall i \in P. \tag{6}$$

Constraint (3) shows whether server i is used ($y_i = 1$) or not ($y_i = 0$). Constraint (4) helps ensure that a VM is assigned to one and only one of the servers. Constraints (5) and (6) help guarantee that each server satisfies the resource requirement of VMs on it.

Recent studies [42] have shown that the power consumed by servers can be assumed to be linear with the CPU utilization. An active but idle server burns between 50% and 70% of the power consumed by the server working at full load [15]. Therefore, we defined the power model as

$$P(u) = k \cdot P_{\max} + (1 - k) \cdot P_{\max} \cdot u \tag{7}$$

where $P_{\max}$ is the maximum power consumed by a full-loaded server; k is the fraction of power consumption in idle state; and $u (u \in [0, 1])$ is the CPU utilization of the server. Since power consumed by the idle state is the major source of wasted energy, reducing the number of active servers can save energy significantly. In the experiments, the consumed power is calculated according to the power model in (7).

B. Ant Colony System

With major improvements over the original ACO algorithms, the ACS was proposed by Dorigo and Gambardella [43] initially for solving the traveling salesman problem (TSP). Real ants are capable of finding the shortest path between the nest and the food source according to information passed via pheromone. Inspired by the foraging behavior of ants, ACS uses pheromone to record historical searching experience. Moreover, heuristic information is introduced to provide greedy information to guide the search. In solving a TSP, ants construct a solution by visiting the cities one by one until all are visited. During the construction process, the next city to visit is selected according to the pheromone and heuristic information. Similarly, a VMP can also use such a step-by-step construction as shown in Fig. 1(a), assigning the VMs one by one to suitable servers. Thus, ACS is applicable to VMP problems.

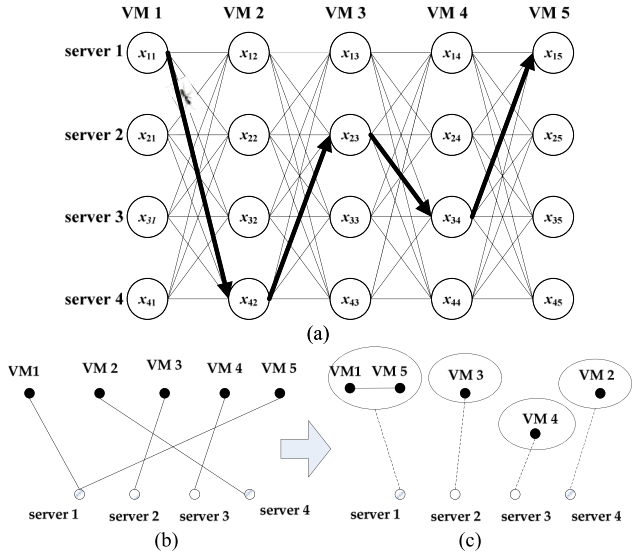

Fig. 1. (a) Example of a construction scheme, with $N = 5$ and $M_t = 4$. (b) Simplified bipartite graph of VMP, with solid lines representing VMs assigned to servers. (c) Grouping relationship graph of VMs in VMP. The ellipsoid represents a VM group onto a server. The solid line represents the link between VMs which are assigned on the same server. The dotted lines represent an assignment.

C. Pheromone in ACS

Pheromone records accumulated historical experience and helps the search more directed. Under the guidance of pheromone, ants search in the neighborhood of the best solution that has been found. Since VMP can be described as a bipartite graph with links between VM and server as shown in Fig. 1(b), an intuitive idea is to deposit the pheromone on the edge between a VM and a server. However, for two homogeneous servers, e.g., servers 1 and 2, there is no difference for a VM to be assigned to either servers 1 or 2. What influence the assignment are the VMs that have been assigned onto the same server. From this angle, an assignment can be described as a graph with links between VMs as shown in Fig. 1(c). The interconnected VMs are assigned to the same server. Since each group corresponds to a server, the links between VM and server [dotted line in Fig. 1(c)] is implied in

the link between VMs. Based on this consideration, we introduce pheromone between VM pairs to record the grouping relationship of VMs.

III. OEMACS Algorithm for Solving the VMP

A. Initialization State Configurations

To minimize the number of active servers and then reduce the energy consumption for cloud computing, we develop an ACS-based approach here, with OEM operations. The resultant OEMACS algorithm searches for a solution with minimal number of servers to host all the VMs. We denote this minimal number as $M^{\min}$ in the feasible globally best solution S^{gb}. Since $M^{\min}$ is our optimization objective which is unknown in advance, we begin with $M^{\min} = N$ in the initialization state, where N is the number of VMs. This means that the initial feasible globally best solution S^{gb} is to place the N VMs on N servers with one VM mapping to one server. For each pair VM k and VM $j(j \neq k)$, we introduce a pheromone value $\tau(k, j)$ and initialize it as $\tau_0 = (N)^{-1}$. The pheromone value indicates the preference of two VMs to be assigned to the same server according to the historical experience.

B. Solution Construction

After the initialization, OEMACS goes to construct solutions iteration by iteration so as to find better feasible solutions with fewer servers. In each iteration $t(t \geq 1)$, OEMACS aims to find a feasible solution with one server less than $M^{\min}$. Therefore, the ant tries to place the N VMs to the $M_t = M^{\min} - 1$ servers.

In each iteration, multiple ants construct their own solutions (assignment) with the guidance of the construction rule. Each ant maintains a construction process by choosing vertices from a construction scheme. Fig. 1(a) shows an example of the construction scheme with $N = 5$ VMs and $M_t = 4$ available servers. It can be observed that the vertices of the construction scheme are arranged into an $M_t \times N$ matrix. Each vertex x_{ij} denotes a VM assignment to a server. The undirected arc between two vertices in the adjacent columns indicates the potential route of ants. Taking Fig. 1(a) as an example, a solution $S = \{s_1, s_2, s_3, s_4\}$, where s_i indicates a set of VMs assigned to server i, is constructed by the ant according to the path $(x_{11}, x_{42}, x_{23}, x_{34}, x_{15})$ (denoted by black arrows line). This solution shows that all the four servers are active to host the five VMs, with the consolidation as $s_1 = \{1, 5\}$, $s_2 = \{3\}$, $s_3 = \{4\}$, and $s_4 = \{2\}$.

Each ant adopts similar process to construct a solution according to the construction scheme like Fig. 1. Note that the VMs are randomly shuffled before each construction process. Then the ant constructs a solution by assigning VMs one by one to the servers. Therefore, the VMs are not in particular order for solution construction in the evolutionary process. We describe the solution construction process based on one ant in the follows. It should be noted that the following description is based on a partial solution under construction. There are totally N steps for an ant to construct a solution, with each step to select a proper server for the corresponding VM. In the lth $(1 \leq l \leq N)$ step for placing VMj, a set of available servers I_j is first defined as

$$I_j = \left\{ i \left| \sum_{n=1}^{N} x_{\text{in}} \cdot \text{vc}_n + \text{vc}_j \leq \text{PC}_i \text{ and } \sum_{n=1}^{N} x_{\text{in}} \cdot \text{vm}_n + \text{vm}_j \leq \text{PM}_i, 1 \leq i \leq M_t \right. \right\} \tag{8}$$

whose element i stands for that server i has enough remaining resources (both CPU and memory) to host the unassigned VMj. Then the ant uses a state transition rule to select a proper server i from the set I_j. The pheromone and heuristic values in the state transition rule are described as follows.

For the pheromone, OEMACS deposits the pheromone between VMs rather than between a VM and a server. So we design a method to translate the pheromone between VM pairs into the preference between VM and server (the server here is also the existing VMs group which includes the links between VMs). The preference between VMj and server i represents the historical experience of packing the VMj together with those VMs (the set of s_i) that have already been placed on server i. Suppose the pheromone between two VMs, VM k and VMj, is denoted by $\tau(k, j)$. We calculate the preference value $T(i, j)$ of VMj to be assigned to server i as the average of the pheromone between VMj and the VMs that have been placed on server i. If there is no any VM deployed on server i, the value of $T(i, j)$ is set as τ_0. Therefore, we have

$$T(i,j) = \begin{cases} \frac{1}{|s_i|} \sum_{k \in s_i} \tau(k,j), & \text{if } |s_i| \neq 0 \\ \tau_0, & \text{otherwise} \end{cases} \tag{9}$$

where s_i is the existing VM set on server i and $|s_i|$ is the number of VMs deployed on server i.

Unlike the pheromone that provides historical information, the heuristic information is for deriving a better selection at the current local situation by a greedy strategy. In order to use a smaller number of active servers, each active server needs to host more VMs, which results in an increase of resource utilization of the server. On the other hand, the balance use of resources in all dimensions helps avoid the situation that some resources are highly utilized while other kinds are lowly utilized, which is not beneficial to the full use of the servers. Based on these, the heuristic information is designed to improve the utilization of different resources as well as to balance the usage of different resources in the servers (the utilization of different resources being high and similar). The heuristic information is associated to each VM assignment for measuring the utilization improvement that VM j can bring to server i, whose value is calculated as

$$\eta(i,j) = \frac{1.0 - \left| \frac{\text{PC}_i - \text{UC}_i - \text{vc}_j}{\text{PC}_i} - \frac{\text{PM}_i - \text{UM}_i - \text{vm}_j}{\text{PM}_i} \right|}{\left| \frac{\text{PC}_i - \text{UC}_i - \text{vc}_j}{\text{PC}_i} \right| + \left| \frac{\text{PM}_i - \text{UM}_i - \text{vm}_j}{\text{PM}_i} \right| + 1.0} \tag{10}$$

where UC_i and UM_i represent the usage of CPU and memory of server i before joining VMj. OEMACS considers two factors when designing the heuristic message: one is the resource utilization in all dimensions (e.g., both CPU and memory), as

the denominator of (10); another is the balance of the remaining resources on the server, as the numerator of (10). The increase of the resource utilization and the balanced use of different resources benefit VM consolidation and then reduce the number of active servers.

With the design of pheromone and heuristic information, the probability for assigning an unassigned VMj to server i is calculated by

$$p(i,j) = \frac{T(i,j)\eta(i,j)^{\beta}}{\sum_{k\in I_j} T(k,j)\eta(k,j)^{\beta}}, \quad \forall i \in I_j \tag{11}$$

where $\beta(\beta > 0)$ is a predefined parameter that controls the relative importance of heuristic information.

In the OEMACS algorithm, the state transition rule is as follows: for VMj, it chooses server i from the servers set I_j by applying the rule given by

$$i = \begin{cases} \arg\max_{k\in I_j} T(k,j)\eta(k,j)^{\beta}, & \text{if } q \le q_0 \\ I, & \text{otherwise} \end{cases} \tag{12}$$

where q is a random number uniformly distributed in [0, 1], I is a random number selected from I_j by a roulette wheel selection according to probability distribution in (11), and q_0 is a predefined parameter ($0 \le q_0 \le 1$) and is used to control the exploitation and exploration behaviors of the ant. If q is not larger than q_0, then the ant chooses the server whose preference value T and heuristic η are maximal, measured by $T(i,j)\eta(i,j)^{\beta}$. Otherwise, the ant chooses server i which is probability selected according to (11).

A special case is that all servers are overloaded after joining VM j (i.e., $I_j = \varphi$). To address this issue, we design a complementary rule to assign VMj to server i as

$$i = \begin{cases} \arg\min_{1\le k\le M_t} \text{over}(k), & \text{if } q \le q_0 \\ R, & \text{otherwise} \end{cases} \tag{13}$$

where R is a random integer in $[1, M_t]$ selected by a roulette wheel selection according to the probability distribution in (15). If a randomly generated number q is not larger than q_0, then the ant chooses the server whose overload rate after joining VMj is minimal. The overload rate describes the difference between the usage and the resource capacity after joining VMj. We calculate the overload rate over(i) of server i as

$$\text{over}(i) = \frac{\left|\text{PC}_i - \text{UC}_i - \text{vc}_j\right|}{\text{PC}_i} + \frac{\left|\text{PM}_i - \text{UM}_i - \text{vm}_j\right|}{\text{PM}_i}. \tag{14}$$

Otherwise, VMj will be assigned to server $i(1 \le i \le M_t)$ which is selected according to the probability distribution

$$r(i,j) = 1 - \frac{\text{over}(i)}{\sum_{k=1}^{M_t} \text{over}(k)}. \tag{15}$$

C. Objective Function

After an ant has finished constructing a solution, we must evaluate its fitness value. Assume that S is an assignment solution to the VMP problem. We evaluate the solution in a two hierarchical structure as

$$f_1(S) = \begin{cases} \sum_{i=1}^{M_t} y_i, & \text{if } S \text{ satisfies the capacity constraint} \\ M_t + 1, & \text{otherwise} \end{cases} \tag{16}$$

$$f_2(S) = \sum_{i=1}^{M_t} \left(\left(\frac{|\text{PC}_i - \text{UC}_i|}{\text{PC}_i} + \frac{|\text{PM}_i - \text{UM}_i|}{\text{PM}_i} \right) \cdot y_i \right) \tag{17}$$

where M_t is the number of servers provided in current iteration t, y_i means that whether server i is used in the solution S. In (16), if the solution is feasible, $f_1(S)$ is the number of active servers, which is not larger than M_t. Otherwise, $f_1(S)$ is set to $M_t + 1$ to distinguish from the feasible solutions, which means that the number of servers we really need to use is the same as the best feasible solution in the last iteration, that is, equals to $M^{\min}$ because $M_t = M^{\min} - 1$. Equation (17) calculates the approximation of the placement to fill up the servers, to evaluate the resource utilization or distinguish the easiness of being transformed into a feasible solution. For two solutions, we compare their f_1 values first, and select the solution with a smaller f_1 value. If two solutions have the same f_1 value, we compare their f_2 values, then select the one with a smaller f_2 value. That is, the solution with fewer servers and higher utilization is preferred.

D. Pheromone Management

The pheromone records the historical preference information. A local pheromone updating and a global pheromone updating rule are implemented in the optimization process. After a solution has been constructed by each ant, the local pheromone updating operation is performed on each VM-pair (k, j) on the same server. The updating rule is

$$\tau(k,j) = (1-\rho)\cdot\tau(k,j) + \rho\cdot\tau_0 \tag{18}$$

where $0 < \rho < 1$ is the pheromone decay parameter.

In contrast, only is the best solution of the current iteration allowed to perform the global pheromone updating operation at the end of each iteration. When all the ants have built their solutions, the best solution of the current iteration can be found and denoted as S^b. The solution S^b, however, can be either feasible or infeasible. If S^b is feasible, it means that OEMACS has found a new feasible solution with servers no more than M_t, then we update the feasible globally best solution S^{gb} to S^b and update $M^{\min}$ to $f_1(S^b)$. That is, the active servers used in the solution S^b. On the other hand, if S^b is infeasible, it means that OEMACS cannot find a feasible solution with only M_t servers. In this case, OEMACS carries out the OEM local search on S^b, which is described in Section III-E. No matter the S^b is feasible or not, the global pheromone updating rule is carried out on S^b after the above operations, to increase the pheromone on the VM-pair of the same server of S^b as

$$\tau(k,j) = (1-\varepsilon)\cdot\tau(k,j) + \varepsilon\cdot\Delta\ \tau_i, \text{if}(k,j) \in s_i, \forall s_i \in S^b \tag{19}$$

$$\Delta\tau_i = \frac{1}{f_1(S^b)} + \frac{1}{\text{LC}_i + \text{LM}_i + 1}. \tag{20}$$

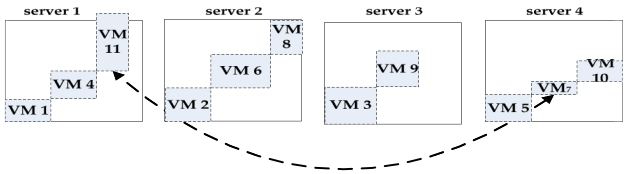

Fig. 2. Example of the ordering exchange operation on a solution where server 1 is overloaded while servers 2–4 are not overloaded, and the solution becomes feasible after exchanging VM 11 on server 1 with VM 7 on server 4.

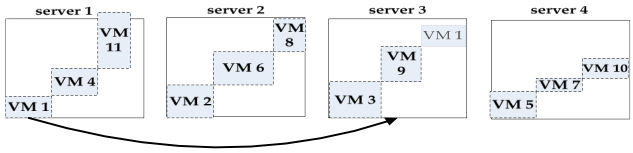

Fig. 3. Example of the migration operation where VM 1 on server 1 is moved to server 3. The dotted rectangular with VM number represents a VM, and the large solid rectangular represents a server.

where $\varepsilon(0 < \varepsilon < 1)$ is the pheromone enhance parameter, $f_1(S^b)$ is the number of servers used in S^b, LC_i and LM_i represent the normalized remaining CPU and memory resource of server i (the ratio of remaining resource to the total resource). The key idea behind the above equation is to record an almost "full" VM group (the utilization of server is high) and a better solution. The pheromone of VM pairs in the more "occupied" server will increase more.

The local and global pheromone updating rules play different roles in guiding the search act of the ants. The local pheromone updating rule reduces the appeal of grouping VM pair which has been found by the last ant, and to help the other ants explore new assignment space. It is able to avoid rapid convergence toward a narrow neighbor region of the best previous route and enhance the population diversity. Global pheromone updating rule is used to strengthen the ties between VM pairs in the good assignments and guides the ants to construct better solutions in a more promising direction.

E. Local Search Procedure

Local search is significant in transforming an infeasible solution into a feasible solution. At the end of each iteration, before the global pheromone updating, the OEM local search is carried out on the current best solution S^b if S^b is infeasible. The OEM includes two procedures. First, an ordering exchange operation and then a migration operation. Both operations try to adjust the VM assignments to ease or eliminate the overloaded servers. If S^b becomes feasible after the local search, we update the feasible globally best solution S^{gb} to S^b and update $M^{\min}$ to the server number of the new feasible solution.

1) Ordering Exchange Operation: The ordering exchange operation swaps VMs between different servers. The operation is used to enhance the resources utilization of nonoverloaded servers and to avoid unbalance resources utilization of overloaded servers. For each overloaded server, we can try to exchange every VM on it with every VM on a nonoverloaded server, to observe whether the overloaded server can become nonoverloaded while the nonoverloaded server is still nonoverloaded. In order to reduce the computational burden of this process, we design the following exchange strategy.

For every overloaded server i, we sort the VMs on it according to the absolute difference between CPU and RAM requirements and give preference to the VM with higher difference for exchange (i.e., sort to front). Then we select one nonoverloaded server k for exchange. The VMs on server k are sorted by the rule contrary to server i (i.e., VM with smaller resource difference is sorted to front). Subsequently, we check the VMs on server i one by one with the VMs on server k one by one to determine whether the two VMs can be exchanged. The process ends until the exchange makes server i become nonoverloaded while the server k is still nonoverloaded, or all VMs on server k have been checked whether to exchange. If server i cannot turn to nonoverloaded after all exchange attempts, another nonoverloaded server is selected for exchange.

Fig. 2 presents possible transition directions of the exchange operation in an example. The dotted rectangular with VM number represents a VM, and the large solid rectangular represents a server. The length of horizontal line of rectangular represents CPU size, and the length of vertical line represents the RAM size. Two VMs connected by arrow line are the exchange objects. Server 1 is overloaded because the required RAM exceeding the size. Servers 3 and 4 are not overloaded. After the ordering exchange operation, VM 11 on server 1 is exchanged with VM 7 on server 4. Then the solution becomes feasible. The exchange makes the resource utilization be less than or get near to the resource upper bound of servers.

2) Migration Operation: The migration operation schedules a VM from an overloaded server to a nonoverloaded server that has enough remaining resource to satisfy the resource requirement of this VM. The operation is similar to Falkenauer's remove mutation [57]. Check the VMs on the overloaded servers one by one with the nonoverloaded servers one by one to determine whether the VM can move to the server. The operation terminates when all servers become nonoverloaded or there is no VM movement that can be carried out. A possible transition scenario of migration operation is illustrated in Fig. 3. The length of horizontal line of rectangular represents CPU size, and the length of vertical line represents the RAM size. The pale VM 1 on server 3 shows that VM 1 fits server 3. The arrow line connects a VM and a server, which highlights the direction of VM migration.

Local search is based on the idea of lowering resource utilization of overloaded servers. The migration operation is performed after the ordering exchange operation. In an early stage, the number of servers is large enough to find a feasible solution, so local search does not work. In a later stage, the searching approaches the optima and becomes harder. Local search can improve searching by converting the infeasible solution into a feasible one.

F. Complete OEMACS Algorithm

The overall flowchart of the OEMACS algorithm is shown in Fig. 4 and is described in the following six steps.

Step 1: Initialization. Set the parameters τ_0. Set the feasible globally best solution S^{gb} as placing the N VMs

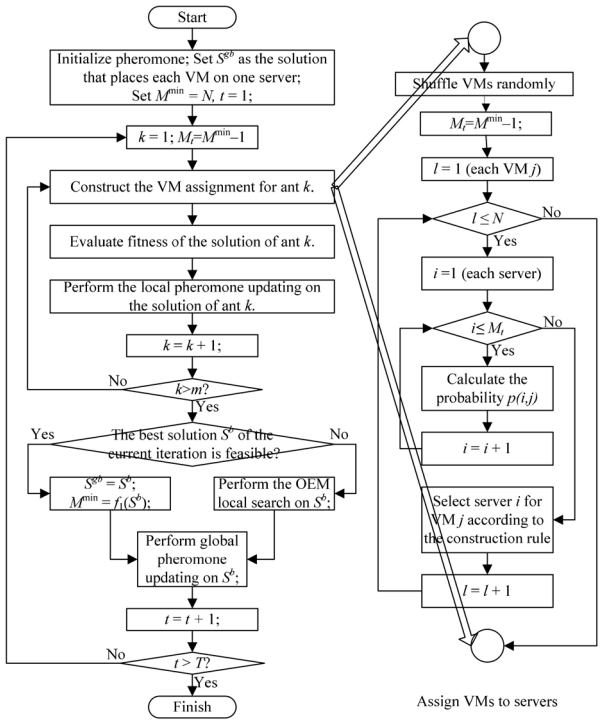

Fig. 4. Flowchart of the OEMACS algorithm.

on N servers with one VM mapping to one server. Therefore, set the number of minimum servers $M^{\min}$ as N. Set the iteration $t = 1$.

Step 2: Set $M_t = M^{\min} - 1$. Let m ants construct m solutions according to the construction rule, and perform local pheromone updating on each solution.

Step 3: Evaluate the fitness values of the m solutions.

Step 4: Find out the current iteration best solution S^b; if S^b is feasible, update S^{gb} as S^b and set $M^{\min} = f_1(S^b)$; otherwise, perform OEM local search on S^b. Update S^{gb} and $M^{\min}$ if the local search successes.

Step 5: Perform global pheromone updating on S^b.

Step 6: Termination detection. When the maximum number of iterations is reached, the algorithm terminates. Otherwise, set $t = t + 1$ and move to step 2 for the next iteration.

Throughout the procedure, step 2 is a main process of the OEMACS algorithm. A detail description is shown in the subflowchart in Fig. 4.

IV. Experiments and Comparisons

Experimental tests are carried out in this section to verify the performance of OEMACS. All the algorithms have been implemented in C++, and ran on a PC with a Pentium Dual CPU i7 and 4.0GB RAM.

The experiment instances in [32] are used to test the performance of the OEMACS algorithm. Moreover, we design two other test sets, tests B and C, with a bottleneck resource (one resource demands more) under homogeneous and heterogeneous server environments, respectively. The efficiency of OEMACS is also evaluated by comparing with the corresponding results obtained by the FFD heuristic approach [40], the RGGA approach [32], the ACO-based approach [11], multiobjective MACO [36], and HACOPSO [37]. FFD sorts the VMs in descending order by first considering CPU requirement and then RAM requirement and assigns each VM to the first server with enough remaining resource. FFD gives the result which is less than or equal to $11/9 * \text{OPT} + 1$, and performs well compared with other deterministic algorithms [40]. Therefore, FFD can be regarded as a representative of heuristic and deterministic algorithms. The RGGA is compared because of its superiority over traditional approaches [32]. In an RGGA, a crossover operator is performed to generate a candidate solution. The servers in two parents are combined together and sorted by resource utilization. Then, the more full servers with different VMs are selected. The remaining unassigned VMs are sorted in a decreasing order according to the requirement of CPU first and then memory, and are placed by the FFD strategy. The ACO algorithm based on max-min ant system in [11] is compared to verify the efficiency of the proposed ACS approach. The MACO and HACOPSO are compared because they are the most recent well-performed approaches that are reported to have better performance than other algorithms, so as to evaluate the performance and advantages of OEMACS.

The OEMACS related parameters are $m = 5$, $q_0 = 0.7$, $\rho = 0.1$, $\epsilon = 0.1$, $\beta = 2.0$, $M_1 = N - 1$, $\tau_0 = (N)^{-1}$, and maximal iteration $T = 50$. We assume that the resource utilization can reach 100%. Parameter settings of RGGA, ACO, MACO, and HACOPSO are consistent with their original literatures, with their population sizes being 75, 5, 8, and 20, respectively, while their maximal iterations are 100, 50, 100, and 500, respectively. Therefore, the maximal function evaluations (FEs) for RGGA, ACO, and MACO are 7500, 250, 800, and 10 000, respectively. Therefore, RGGA, MACO, and HACOPSO all have much larger FEs than the one used by OEMACS, which is only $5 \times 50 = 250$, because we anticipate that OEMACS converges faster. As EC algorithms all contain certain randomness in the search process, they perform 30 independent runs on each instance in all the test environments for fair comparison. The runtime of FFD is a few micro seconds in all instances. So it is not listed. The best results are marked in bold face.

A. Test A: Large-Scale Homogenous Environment

The data setproposed in [32] were created so that every VM would fit the set of servers perfectly (i.e., there is no redundant resources in any server), based on server consolidation studies by VMware. It includes nine different two-capacity problem instances of differing size from 1000 to 6000 VMs, numbered from A1 to A9. Each server has 500 CPUs and 500GB RAM. The CPU requirement of VM is an integer within the range of [1,128], and the memory requirement is an integer within the range of [0,100]. Therefore, the ratio of total requirement of CPU and memory is around 1:1. As the optimal placement has no remaining resources in any server, these problems are meant to be very hard. The OPT column

TABLE I
EXPERIMENTAL RESULT COMPARISONS IN TEST A WITH DIFFERENT SIZE OF VM

No.	N	OPT	OEMACS			RGGA [32]			ACO [11]			MACO [36]			HACOPSO [37]			FFD [40]
			Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	
A1	1363	112	**113.00**	**1.366**	**39.6**	**113**	3.300	2387.5	116.5	1.836	45.8	130.26	11.200	285.6	117.56	7.233	914	133
A2	2314	191	**192.00**	**4.097**	42.6	192.53	10.667	4732.5	199.03	4.100	**35**	222.16	31.633	277.3	204.03	21.233	949.3	229
A3	2639	216	**217.00**	**5.662**	**43.8**	218.13	12.367	4602.5	225.33	10.000	48.3	251.13	43.100	279.7	238.93	27.800	964.6	258
A4	2972	241	**242.00**	**7.441**	**45.6**	243.1	18.600	5152.5	251.43	15.700	79.5	282	39.566	226.4	263.93	34.967	969.3	288
A5	3289	267	**268.00**	**8.738**	**45.8**	269.43	23.733	5700	278.76	20.200	85	312.26	52.533	264.8	322.83	26.900	940	320
A6	3958	320	**321.00**	**13.067**	**46.8**	323	31.067	5337.5	334.03	25.800	74	374.83	57.033	194.4	387.93	41.400	984	385
A7	4604	371	**372.00**	**17.840**	**46.8**	374.4	38.400	5615	387.46	34.000	71	435.7	72.466	181.3	522.13	51.500	949.3	447
A8	5268	425	**426.00**	**30.300**	**48**	429.96	45.667	5880	443.86	39.100	62	499.7	120.800	173.6	540.5	89.500	967.3	515
A9	5949	481	**482.00**	**28.933**	**47.8**	486.7	59.233	5890	502.2	90.000	112	564.23	198.900	294.9	722.4	138.600	952	583

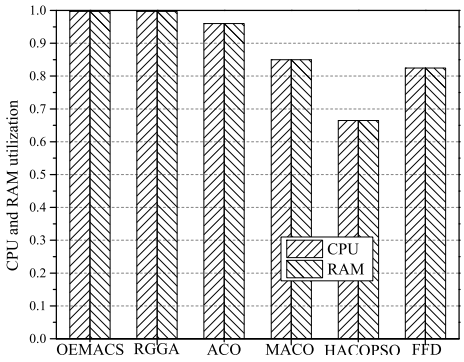

Fig. 5. Average utilization of CPU and RAM of all active servers on A9.

in the table represents the minimal number of servers used for deploying all the VMs.

It can be observed from Table I that OEMACS obtains the best solution. More significantly, OEMACS obtains the same number of active servers in 30 runs, indicating its stable performance. The FFD yields very poor solutions using the maximal number of servers. ACO, and MACO always perform worse than OEMACS and RGGA but better than FFD. HACOPSO performs better than FFD on some cases. Moreover, as the problem size increases, the advantages of EC-based algorithms become more significantly when compared with the traditional algorithm FFD. For example, in A1, ten physical servers are less in OEMACS than in FFD, while in A9, more than 100 physical servers are less. Fig. 5 displays the utilization of CPU and RAM of active servers for OEMACS, RGGA, ACO, MACO, HACOPSO, and FFD on A9. OEMACS and RGGA obtain the highest CPU and RAM utilization which is near to 100%. More significantly, the utilization rates of the CPU and RAM resources are balanced.

Another advantage of OEMACS is that it can obtain better solution than other approaches do with fewer FEs in much shorter computational time. Table I records the average running time and FEs when the algorithm cannot improve the solution quality any more. For example, in A1, OEMACS uses average 39.6 FEs (with average 1.366 s) to obtain the final result 113. However, when RGGA stops in the result 113, it needs average 2387.5 FEs (with average 3.300 s), and ACO needs 45.8 FEs (with 1.836 s) to obtain the result 116.5, MACO needs 285.6 FEs (with 11.200 s) to obtain the result 130.26, and HACOPSO needs 914 FEs (with 7.233 s) to obtain the result 117.56. The convergence speed advantage to obtain good result is very significant when compared OEMACS with RGGA, MACO, and HACOPSO, although ACO sometimes stops improving the solution early than OEMACS. However, the early stop makes ACO result in very poor solutions, especially in large-scale problems. The detailed convergence of OEMACS will be further analyzed and discussed in Section IV-E.

The results in test A show that OEMACS is able to solve large-scale problems (even with number of VMs up to about 6000) under homogenous server environments where the relative overall demands of CPU and RAM are comparable (the ratio being 1:1). Moreover, OEMACS has general better performance in obtaining better solution with fast convergence speed when compared with the compared FFD, RGGA, ACO, MACO, and HACOPSO algorithms.

B. Test B: Bottleneck Resource Homogenous Environment

To further test the effectiveness and efficiency of OEMACS, we designed a set of data that models the VMs and servers in a cloud computing environment that has bottleneck resource. We created eight different problem instances with different sizes from 100 to 2000, numbered sequentially from B1 to B8. Each server has a 16-core CPU and 32GB RAM. Each VM has CPU requirement of 1–4 cores and memory requirement of 1–8GB, which is generated randomly from discrete uniformly distributions. CPU is the bottleneck resource in this case because the probability of 4-core VM is 0.25 but 7 or 8GB VM is 0.125. Therefore, the overall ratio of requirement of CPU to memory utilization is nearly 10:9. Since the instances are generated randomly, we do not know the optimal solution. We estimate the lower bound of the optimum as $\tilde{M}$ and calculate its value as

$$\breve{M} = \max\left\{\left\lceil \frac{\sum_{j=1}^{N} \mathrm{vc}_j}{\mathrm{PC}_i} \right\rceil, \left\lceil \frac{\sum_{j=1}^{N} \mathrm{vm}_j}{\mathrm{PM}_i} \right\rceil\right\} \tag{21}$$

where PC_i and PM_i are the CPU and RAM capacities of any server because the servers are homogenous, while $\sum_{j=1}^{N} \mathrm{vc}_j$ and $\sum_{j=1}^{N} \mathrm{vm}_j$ are the sum of CPU and RAM requirements of all the VMs, respectively.

The results obtained by OEMACS, FFD, RGGA, ACO, MACO, and HACOPSO are given and compared in Table II. Similar to the results in test A, results presented in Table II also reveal the effectiveness and efficiency of OEMACS. It can

TABLE II
EXPERIMENTAL RESULT COMPARISONS IN TEST B

No.	N	$\breve{M}$	OEMACS			RGGA			ACO			MACO			HACOPSO			FFD
			Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	
B1	100	16	**16.00**	0.0070	12.6	**16.00**	0.0666	75.0	**16.00**	**0.0010**	**5.0**	**16.00**	0.0073	32.2	16.93	0.0063	133.3	18
B2	200	33	**33.00**	0.0226	16.0	**33.00**	0.0333	75.0	**33.00**	**0.0076**	**9.6**	**33.00**	0.0173	21.6	34.00	0.0436	331.3	37
B3	300	46	**46.00**	**0.0516**	**17.8**	46.03	0.4000	895.0	46.63	0.0533	27.3	47.00	0.1840	99.7	47.00	0.1186	426.0	53
B4	400	64	**64.00**	**0.1170**	**19.0**	64.03	0.1333	395.0	64.33	0.1700	49.1	65.90	0.5306	167.4	65.10	1.0540	2343.3	74
B5	500	77	**77.00**	**0.1436**	**21.1**	77.63	1.7666	3417.5	78.93	0.2573	47.0	81.93	0.7316	141.3	80.50	2.2123	3185.3	91
B6	600	95	**95.00**	**0.2163**	**19.5**	95.77	0.7333	1037.5	96.73	0.2853	35.1	99.16	2.1593	302.9	99.20	5.2763	5397.3	111
B7	1000	159	**159.00**	**0.6493**	**21.6**	161.23	2.3666	2787.5	161.86	1.1546	50.5	168.86	7.2600	333.6	174.66	15.498	5798.6	184
B8	2000	317	**317.00**	**2.0333**	**24.1**	319.64	7.1333	4090.0	322.73	4.8053	51.1	339.33	18.5333	206.4	345.66	15.0666	924.6	366

TABLE III
EXPERIMENTAL RESULT COMPARISONS IN TEST C

No.	N	$\breve{M}$	OEMACS			RGGA			ACO			MACO			HACOPSO			FFD
			Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	
C1	100	18	**19.00**	0.0240	**53.5**	19.33	0.466	1420.0	23.13	**0.0136**	58.6	28.267	0.0510	261.0	24.56	0.0036	101.3	32
C2	200	45	**45.00**	0.0773	**64.6**	45.26	1.133	2652.5	55.26	**0.0643**	77.3	68.26	0.2583	329.6	57.76	0.1376	1121.3	75
C3	300	68	**68.00**	0.1743	**77.5**	**68.00**	0.900	1392.5	79.13	**0.1673**	88.6	106.86	0.4773	266.1	79.36	0.2230	812.6	102
C4	400	80	**81.00**	0.2873	79.1	82.36	2.900	5512.5	103.06	**0.2260**	**66.8**	137.16	0.7850	244.5	114.40	1.7710	3896.6	131
C5	500	107	**107.00**	**0.4526**	**82.0**	107.96	2.500	3815.0	127.06	0.5100	89.0	178.36	1.3660	289.3	133.66	4.8000	6910.0	167

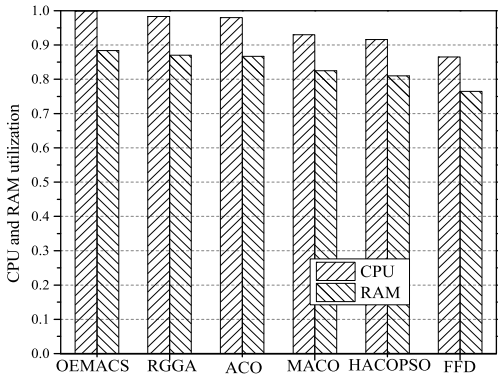

Fig. 6. Average utilization of CPU and RAM of all active servers on B8.

be observed that FFD cannot find the optima in all instances and performed the worst compared with other five algorithms. OEMACS not only outperforms FFD, but also does better than RGGA, ACO, MACO, and HACOPSO. From Table II, we can see that RGGA and ACO achieve results that are equal to $\breve{M}$ in only two out of the eight instances whose number of VMs is not larger than 200. However, the proposed OEMACS obtains results that are equal to $\breve{M}$ in all of the eight instances within only 30 FEs. When the problem size increases, the superiority of OEMACS is more apparent. In comparison with ACO, the advantage of OEMACS is obvious. ACO cannot always obtain the optima within the predefined maximum number of FEs except for very small size instances like B1 and B2. For instance B1, ACO is very fast to find the optima in the second iteration while OEMACS in the fourth iteration. This may be due to that ACO selects VM to be assigned to the current server and open another server while no VM can be assigned to it. ACO acts like the greedy algorithm FFD and can find good solution in early stages to converge quickly in very small size problems. OEMACS constructs solutions from VM's view and selects a server for each VM, so it converges a litter slow in early stage and but jumps out of local optimal rapidly. The results show that OEMACS outperforms not only traditional FFD and RGGA, but also other ACO-based algorithms, in terms of both solution quality and the optimization speed, especially in large-scale homogeneous server problems with bottleneck resource.

Fig. 6 shows the average CPU and RAM utilization of the active servers in the assignment obtained by OEMACS, RGGA, ACO, MACO, HACOPSO, and FFD on test B8. OEMACS obtains the largest CPU and memory utilization. The CPU (the most critical resource) utilization is near to 100% and RAM is near to 90%. It shows that different resource-intensive VMs are balanced, so that both resources are better exploited. In general, OEMACS is able to obtain the best assignment.

C. Test C: Hard Heterogeneous Environment

In tests A and B, all the servers are homogeneous, as in real cloud environment servers are often heterogeneous. So we designed another test environment with heterogeneous servers and CPU-intensive and RAM-intensive VMs. Two kinds of servers (type $s0$ 16-core CPU, 32GB RAM, $P_{\max} = 215$ W and type $s1$ 32-core CPU, 128GB RAM, $P_{\max} = 300$W) are provided. The number of servers of type $s0$ is set as $M_{s0} = 9N/10$ (N is the number of VMs) and type $s1$ as $M_{s1} = N/10$. This makes that using only the large servers (e.g., type $s1$) cannot host all the VMs, so that both types of servers have to be used. We created five problem instances of different sizes from 100 to 500, which are numbered sequentially from C1 to C5. In every problem instance, VMs are generated by discrete uniform distribution of [1, 8] for CPU and [1, 32] for memory. The memory is the bottleneck resource. The lower bound of the optimum $\tilde{M}$ is estimated as

$$\breve{M} = M_{s1} + \max\left\{\left\lceil \frac{\sum_{j=1}^{N} \mathrm{vc}_j - \mathrm{PC}_{s1} \cdot M_{s1}}{\mathrm{PC}_{s0}} \right\rceil, \left\lceil \frac{\sum_{j=1}^{N} \mathrm{vm}_j - \mathrm{PM}_{s1} \cdot M_{s1}}{\mathrm{PM}_{s0}} \right\rceil\right\} \quad (22)$$

which is similar to (21) except that both types of servers are considered because of the heterogeneous environment.

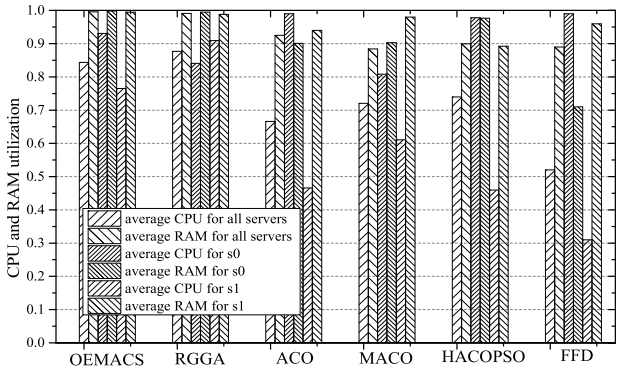

Fig. 7. Average utilization of CPU and RAM of active servers of all, types $s0$ and $s1$ on C5.

The results are given in Table III. As shown in the table, FFD performs poorly. Neither can RGGA, ACO, MACO, nor HACOPSO find the optima in all instances. In contrast, OEMACS can obtain the best results among all the six algorithms in all instances and can even find the optima in C2, C3, and C5. In the instances C1 and C4, the results obtained by OEMACS are only with one server away from the optima. However, the results obtained by other algorithms are far away from the optima. Moreover, OEMACS finds the optima within 250 FEs while RGGA and other ACO-based algorithms cannot find even using more FEs. In summary, OEMACS outperforms FFD significantly in terms of solution quality and outperforms RGGA and other ACO-based algorithms in terms of both solution quality and search speed.

Fig. 7 shows the average CPU and memory utilization of the active servers of all, types $s0$ and $s1$, respectively, in the assignment obtained by different algorithms. Both average CPU and RAM utilization of all active servers of FFD's assignment are the lowest. There is a significant gap between the utilization of CPU and RAM both for types $s0$ and $s1$, which demonstrates that FFD cannot efficiently balance different resource. For OEMACS, the CPU utilization of the active servers of type $s0$ is the highest, while type $s1$ is smaller than the RGGA. It shows that the OEMACS prefers to highly integrate the VMs on high configuration servers while RGGA prefers low-profile servers. Memory is the bottleneck resource and limits the consolidation level. The average memory utilization of OEMACS is the highest, near to 100%, which shows that OEMACS can obtain the highest consolidation of VMs with high resource utilization.

Compared with test B, OEMACS needs more FEs and time to obtain the optima of the same problem size. This is because the VMP problem becomes harder in heterogeneous server environments. However, OEMACS can still find the optima by using more computing power. Take C5 for example, OEMACS can still achieve the optima of 500 VMs within 0.5 s. OEMACS can find the optima in acceptable time. FFD does not seem suitable for large-scale problems as it performs poorly. OEMACS performs well in all three tests. Tests A and B are in homogeneous server environments with different correlation ratios for CPU and memory utilization, and test C is of a heterogeneous server environment. Therefore, our results show that OEMACS can find optima or quasi-optimal solutions for large-scale homogeneous and heterogeneous server data center environments with resource-intensive VMs.

TABLE IV
COMPARISON RESULTS ON STATISTICALLY SIGNIFICANT DIFFERENCES OF OEMACS WITH ALL CONTENDERS AND ON RUNTIME OF ALL ALGORITHMS IN TESTS A, B, AND C

Comparison	OEMACS	RGGA	ACO	MACO	HACOPSO
Wilcoxon's signed rank test	+	0	0	0	0
	=	6	2	2	0
	−	16	20	20	22
Runtime rank	1	3	2	5	4

A nonparametric statistical test called Wilcoxon's signed rank test is conducted between the compared algorithm and OEMACS at a 5% significance level to judge whether the results obtained with the best performing algorithm significantly exhibits superior performance. The null hypothesis in each test is that no difference exists between the compared algorithm and OEMACS. We mark the cases with "+" and "−" when the null hypothesis is rejected to indicate that the compared algorithm performs significantly better or worse than OEMACS. The cases marked "=" means that there is no statistically significant difference between the performance of the two algorithms. The numbers of the three kinds of statistical significance cases $(+/=/-)$ are reported in Table IV in this paper and the details in each test are listed in Table S.I in the supplementary file. From the table, we can see that OEMACS performs significantly better than RGGA, ACO, MACO, and HACOPSO on 16, 20, 20, and 22 out of 22 cases, respectively. OEMACS is seen outperforming the algorithms compared.

The runtime required to obtain the best solution for each algorithm (whose mean value has been reported in Tables I–III) is also performed with a t-test at a 5% significance level to compare with the one of OEMACS. The results of t-value and p-value are reported in Table S.II in the supplementary file. From Table S.II, we can see that OEMACS can find better or similar solutions in significantly shorter time in most cases. Although both number of FEs and runtime that are required to obtain best solutions can reflect the convergence speed of the algorithm, it is still interesting to investigate the total runtime of the algorithm until it terminates. Therefore, the mean total runtime of the 30 runs are presented and compared in Table S.III in the supplementary file, for each problem instance and for each algorithm. Table IV also reports the total runtime ranks of OEMACS, RGGA, ACO, MACO, and HACOPSO. From the table, we can observe that the runtime of OEMACS ranks the first and is the shortest among the compared algorithms.

The OEMACS is also compared with CPLEX, which is a linear program solver. The IBM ILOG CPLEX Optimization Studio 12.6.1.0 using a mixed integer programming solver in a parallel mode with four threads is adopted. The results in Table V show that both CPLEX and OEMACS can obtain similar good solutions on small scale instances, except that CPLEX does slightly better than OEMACS on C1 which is concerned with only 100 VMs, while performing worse than OEMACS on B4 and C4 which are concerned with

TABLE V
COMPARISON RESULTS BETWEEN OEMACS AND CPLEX

No.	OEMACS Mean(Time)	CPLEX Result(Time)	No.	OEMACS Mean(Time)	CPLEX Result(Time)
B1	**16.00 (0.0070)**	**16** (2.95)	C1	19.00 (**0.0240**)	**18** (13)
B2	**33.00 (0.0226)**	**33** (52.81)	C2	**45.00 (0.0773)**	**45** (1607.43)
B3	**46.00 (0.0516)**	**46** (688.83)	C3	**68.00 (0.1743)**	**68** (209.88)
B4	**64.00 (0.1170)**	65 (607.19)	C4	**81.00 (0.2873)**	83 (54860)

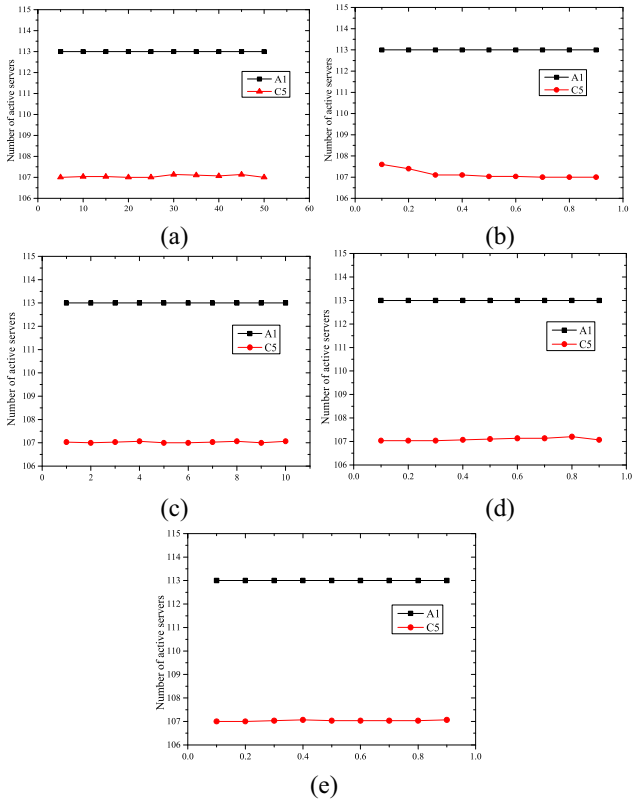

Fig. 8. Influence of the parameters in OEMACS on A1 and C5. (a) m. (b) q_0. (c) β. (d) ρ. (e) ϵ.

400 VMs (where CPLEX stops when it obtains the results due to out-of-memory). On the running time, OEMACS is much faster than CPLEX. The speed advantage becomes more evident as the problem scale increases. Moreover, CPLEX consumes too much time when dealing with VMP with more than 500 VMs, indicating that it may be not suitable for large-scale VMP. Therefore, we only give the available results on B1–B4 and C1–C4 in Table V.

D. Analysis of OEMACS Parameters

The OEMACS parameters include the population size of ants m, β, q_0, ρ, and ϵ. In order to study the influences of the parameters to the solution quality, we take A1 and C5 as an example and perform parameter analysis in this section.

The investigation begins with the parameter m. We set m from 5 to 50 with a step length of 5. All the other parameters remain the same as stated above. Both the mean results of A1 and C5 plotted in Fig. 8(a) are nearly invariant with the increase of population size m. Nevertheless, a large population size causes a high computational burden in each generation.

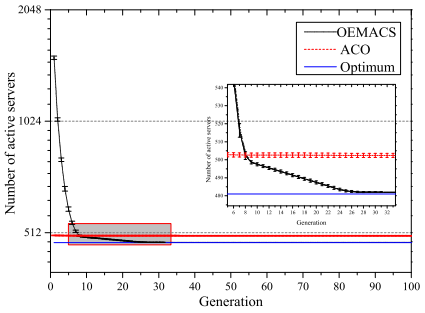

Fig. 9. Convergence curves of OEMACS and ACO on A9.

By considering both of these, this paper adopts the population size of 5 in OEMACS.

The next parameter tested is q_0. In the following investigation, q_0 varies from 0.1 to 0.9 with a step length of 0.1. The mean number of active servers in obtained solutions are plotted in Fig. 8(b). The tendency of the curves indicates that it is better to use a larger q_0 for better performance. However, too large a q_0 makes the algorithm perform poorly due to the loss of exploration ability, e.g., the results for $q_0 = 1.0$ are too bad to be plotted within Fig. 8(b). Therefore, we set q_0 0.7 in this paper.

Then parameter β is investigated. As shown in Fig. 8(c), for A1, the performance of OEMACS with different configurations is similar. For C5, when β is 2, 5, 6, or 9, OEMACS performs the best. The results for $\beta = 0$ are poor and not plotted in Fig. 8(c), which indicates that the heuristic information plays an important role on the algorithm performance.

Finally, parameter ρ for pheromone local updating and ϵ for global updating are tested. Fig. 8(d) and (e) shows the results obtained. The results are poor when parameters ρ and ϵ are set to 0 or 1.0; so they are not plotted in Fig. 8(d) and (e). For A1, the performance of OEMACS is similar with different values of ρ, while for C5, a relatively small ρ value seems to be preferred. This may be due to the fact that a smaller ρ is helpful to avoid a too rapid pheromone evaporation on the visited assignment and to use the historical experience recorded in the pheromone. The curves in Fig. 8(e) of the impact of ϵ are nearly parallel to the horizontal axis. OEMACS is seen insensitive to the value of ϵ.

From Fig. 8, the impact of β, ρ, and ϵ further confirms that these parameters are promising and OEMACS is not very sensitive to the parameters. This is also an advantage of the OEMACS algorithm.

E. Further Convergence Analysis of OEMACS

In this section, we analyze the convergence of OEMACS. Take A9 as an example. The situations on the other problems are similar. The maximum number of iterations is set as 100. Fig. 9 shows the optimization curves of OEMACS and ACO. In the first iteration, 5949 servers are provided, and OEMACS finds a feasible solution with 1519 servers in the first iteration, and obtains a feasible solution with 1034 servers in the second iteration. It can be noted that, while OEMACS initially finds worse solutions, it quickly improves the performance following a steep curve

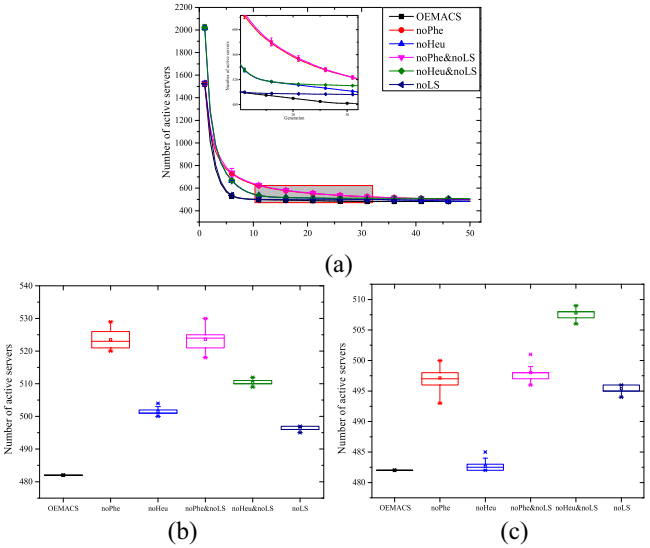

Fig. 10. Experimental results of OEMACS and its variants, noPhe, noHeu, noPhe&noLS, noHeu&noLS, and noLS on A9. (a) Convergence curves with error bars. Box plots of results at (b) 30 generations and (c) 50 generations.

and surpasses ACO, before slowing down when approaching the optima (with 481 servers) from the fifth iteration, and then converging to 482 at the 30th iteration. Conversely, starting on lower server numbers initially, ACO undergoes an incremental progress, and converges prematurely to a higher level (resulting in 502 servers).

The above results demonstrate that OEMACS can converge to a near-optima more effectively and more quickly.

F. Benefits of Pheromone and Heuristic Components

We are interested in identifying the benefit of the three components of OEMACS: 1) pheromone; 2) heuristic information; and 3) local search. For this purpose, we consider five OEMACS variants, i.e., noPhe, noHeu, noPhe&noLS, noHeu&noLS, and noLS. They differ from OEMACS only in that noPhe does not use the pheromone (and "preference"), noHeu does not adopt the heuristic information ($\beta = 0$), and noLS does not perform local search. The A9 is taken as an example. The convergence curves and box plots of results are illustrated in Fig. 10. Experimental results have shown that both the pheromone and heuristic information are fundamental in helping the OEMACS algorithm find good solutions within a reasonable period of time. As seen from Fig. 10(b) and (c), the noPhe variant and noPhe&noLS variant have similar but poor performance. This is because that heuristic information and local search are greedy strategies and therefore the OEMACS variants without pheromone are actually reduced to a stochastic restart greedy algorithm that is easy to be trapped into local optima. Moreover, noHeu performs better than noPhe, likely due to that it is guided by reinforcements provided by the global updating rule in the form of pheromone although noHeu is not helped by heuristic information. Therefore, pheromone plays a significant role in enhancing the algorithm performance, helping OEMACS perform better than a randomized restart greedy algorithm.

When referring to the heuristic component, the results show that the performance of noHeu&noLS is worse than both noLS and noHeu, while all these three variants perform significantly worse than OEMACS. These indicate the importance of both the heuristic information and local search in enhancing the algorithm performance. From Fig. 10(a) and (b), we can see that OEMACS converges to solution with 482 servers within 30 generations while its noHeu variant gets significantly worse solution with 504 servers. Even given more computational budget (50 generations), noHeu still cannot obtain solution of 482 as illustrated in Fig. 10(c). Along the whole convergence process, OEMACS always keeps its result below noHeu, especially with a large difference in the early stage. Similarly, noHeu&noLS also performs worse than noLS. Thus, the heuristic information helps speed up the convergence and improves the performance of the algorithm.

G. Effectiveness of Local Search

Local search is an important feature in the proposed OEMACS. Two operations of local search contribute in different ways to the optimization process. In this section, we validate the effectiveness by comparing the results of OEMACS with its variants without local search or with only one operation of local search. The variants without local search, ordering exchange operation or migration operation are termed OEMACS-noLS, OEMACS-noE, and OEMACS-noM, respectively. The settings of these variants are exactly the same as OEMACS except that one of the three components is not applied. The following study takes A2, B8, and C5 as examples. A2 and B8 are large-scale problems of homogeneous server settings, and C5 is a complex problem in a heterogeneous server environment. The situations on the other problems are similar.

Table VI lists the results of the four algorithms averaged over 30 independent runs. It can be observed that OEMACS obtains the best solutions, followed by OEMACS-noM and OEMACS-noE, whereas OEMACS-noLS performs the worst. The OEMACS-noLS and OEMACS-noE cannot achieve the optima or approximate optima in 7500 FEs. The advantages of OEMACS over the other three algorithms confirm that the local search is indeed effective in finding high-quality solutions. Compared with the migration operation, the ordering exchange operation contributes more in improving the solutions since OEMACS-noM can find good solutions as OEMACS using more FEs while OEMACS-noE cannot within the maximum FEs.

Table VII shows the success rates of local searches in improving infeasible solutions on A2, B8, and C5 in OEMACS. If an infeasible solution is transformed into a feasible solution by the operation, we say that it is successful. Because the migration operation is carried out after the ordering exchange operation, the success rate only calculates the case improved by the migration operation when the solution is not improved by the ordering exchange operation. It can be observed that the success rate of local search is high. For A2 and B8, the infeasible solution can be improved by the ordering exchange operation but not the migration operation.

TABLE VI
EXPERIMENTAL RESULT COMPARISONS OF OEMACS, OEMACS-NOM, OEMACS-NOE, AND OEMACS-NOLS FOR PROBLEM A2, B8, AND C5

No.	OEMACS			OEMACS-noM			OEMACS-noE			OEMACS-noLS		
	Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs	Mean	Time(s)	FEs
A2	192.00	4.097	42.6	192.00	4.760	45	194.00	195.860	3610	194.03	174.8	3388.3
B8	317.00	2.033	24.1	317.00	2.200	25	321.10	17.500	319.6	321.20	49.7	948.5
C5	107.00	0.4526	82	107.00	1.000	85	112.77	10.200	2585	113.05	12.66	3376.6

TABLE VII
SUCCESS RATE OF IMPROVING THE INFEASIBLE SOLUTION OF LOCAL SEARCH ON A2, B8, AND C5

No.	Successful rate		
	Local search	Ordering exchange operation	Migration operation
A2	88.60%	88.60%	0
B8	100%	100%	0
C5	73.03%	71.68%	9.57%

For C5, the solution, which is transformed by the ordering exchange operation can be improved by the migration operation. This helps the exchange operation to improve the solution further.

For B8, the ordering exchange operation improves all the infeasible solutions on B8, so the migration operation is not executed, and OEMACS performs the same as OEMACS-noM. The time of OEMACS and OEMACS-noM is close.

For A2, the success rate of the ordering exchange operation is just 88.60%; so the migration operation is involved. The migration operation does not transform infeasible solutions into feasible solutions. However, the migration effect results in pheromone transmitting the VM group information into the next generation. Thus the migration operation improves the algorithm indirectly. OEMACS converges in 42.6 FEs with 4.097 s, while OEMACS-noM in 45 FEs with 4.760 s, which shows that the migration operation leads to shorter time.

As for C5, although the success rate of the migration operation is only 9.57%, it helps the algorithm converge more quickly. OEMACS uses only less than a half time to obtain the final results when compared with OEMACS-noM (i.e., 0.4526 versus 1.0). Moreover, the migration operation performs better when it cooperates with the ordering exchange. For all three problem instances, OEMACS takes the shortest time. Hence, both the ordering exchange operation and the migration operation are important for improving the solutions and finding the optima in shorter time.

All the results in Tables VI and VII indicate that local search is important and necessary in the proposed algorithm OEMACS.

H. Energy Saved by OEMACS

The benefit of consolidation is determined primarily by the amount of power wasted due to the resource underutilization. The power consumed by active but idle servers, that is, the quantity P_{idle}, is the major source of energy waste. The value of P_{idle} is expressed as a percentage or fraction k of the power consumed when the server operates at full capacity, $P_{\text{idle}} = kP_{\max}$ in (7). P_{idle} is an inherent part of power consumption of an active server. For the given VMs, the utilization of CPU is determinant, so the second term $u \cdot (1-k)P_{\max}$ in (7) is certain to be included in the power consumption. To reduce it, let us start with the first term $kP_{\max}$ of (7). The P_{idle} is consumed only if the server is active. By means of consolidation to increase the resource utilization and reduce the number of active severs, the power energy can be significantly reduced. The $P_{\max}$ is set to 215W in tests A and B. Fig. 11 shows the power consumption of OEMACS, RGGA, ACO, MACO, HACOPSO, and FFD on tests A, B, and C, when $k = 0.6$. We can observe that OEMACS consumed less power than other algorithms in all cases. To further analyze the benefit of consolidation, we calculate the power consumption with different values of k in two scenarios with or without bottleneck resource. The following study takes A9 and B8 as examples. Both A9 and B8 are large-scale problems of homogeneous server settings, while B8 encounters bottleneck resources. The situations encountered by the other problems are similar. Fig. 12 displays the overall amount of power consumption of various algorithms on A9 and B8.

In Fig. 12(a), the consumed power of different algorithms for A9 is illustrated. Compared with FFD, OEMACS allows the data center to reduce the consumed power significantly, from about 9–15 kW. OEMACS and RGGA consumed the least power. In A9, the overall demands on the CPU and RAM are approximate. Both the utilization of CPU and RAM of OEMACS are incredibly closed to the permitted threshold (100%) seen from Fig. 5. Different resources are utilized effectively. The consolidation of VMs allows the idle servers to be hibernated so as to reduce the consumed power. As the value of k increases, the reduced power increases.

Fig. 12(b) gives the power consumption of different algorithms for B8. The power consumed by OEMACS is the lowest compared with traditional algorithm FFD and other evolutionary algorithms RGGA, ACO, MACO, and HACOPSO. This result coincides with the highest utilization of CPU and the lowest number of active servers of OEMACS. In B8, the CPU is the pivotal resource and determinate the consolidation degree. From Fig. 6, we can see that the CPU of OEMACS utilized up to the permitted threshold 100% and RAM to 90%, while the CPU and RAM of FFD utilized up to 90% and 80%, respectively. The number of the active servers of OEMACS is minimal and is equal to the optimum, which is shown in Table II. As servers get more power efficient (k becomes smaller), the reduced power decreases, and the superiority of consolidation decreases. Nevertheless, when servers consume only 40% of the power at idle state, OEMACS can still save power about 6%, compared with

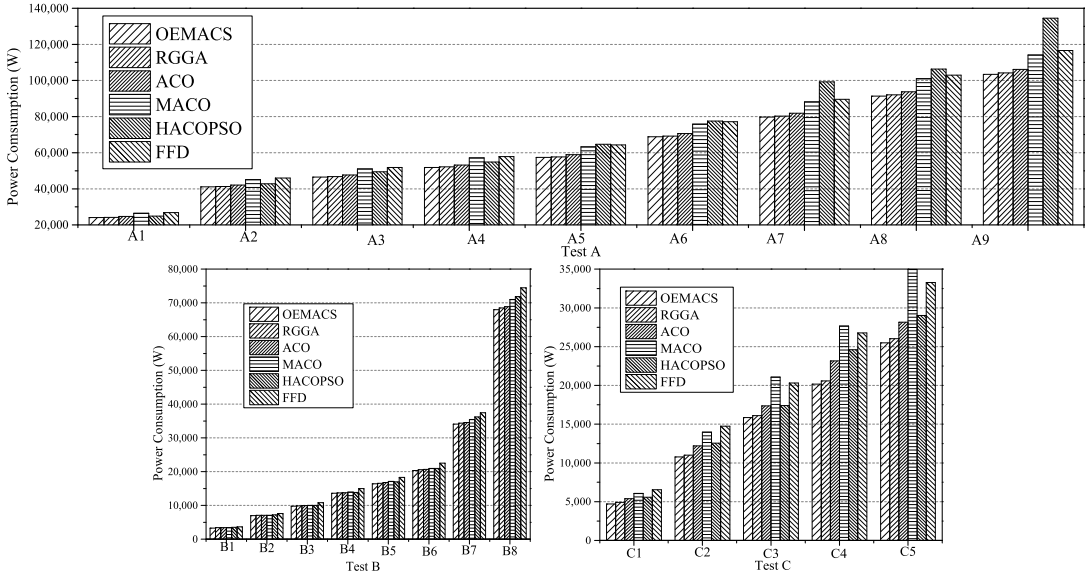

Fig. 11. Power consumption of different algorithms under different test cases.

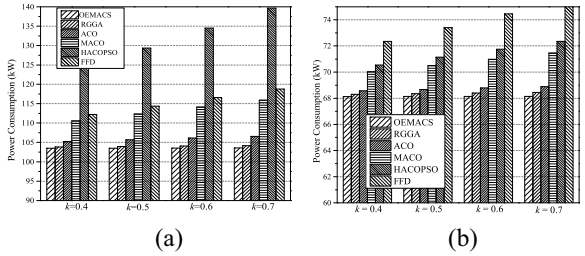

Fig. 12. Consumed power with different values of k (the ratio of power consumed at idle state to maximum utilization) on (a) A9 and (b) B8.

FFD. The results in Fig. 12 confirm that OEMACS have a substantial advantage in saving power.

V. Conclusion

Energy consumption contributes most to the total cost in a cloud system. This motivates us to have developed an energy efficient OEMACS for VMP in cloud computing. The optimal VM deployment has been achieved with the minimum number of active servers and by switching off the idle servers.

The VMP problem is a complex NP-hard problem. To solve this problem, OEMACS, an ACS-based approach, has been developed in this paper. The assignment of VMs is constructed by artificial ants based on global search information. OMEACS distributes pheromone between VM pairs, which represents a bond among the VMs on the same server and records good VM groups through learning from historical experience. To revise infeasible solutions, local search is performed, which contributes significantly to improving the solutions and speeding up global convergence of the OEMACS. Moreover, the number of servers provided for placing VMs reduces as the generation number grows, avoiding possible wastes of computation while providing guidance for further advancement of the solutions. These distinct features and the strong global search nature of an ACS make the OEMACS efficient for large-scale problems. It shows a significant advantage compared with other heuristic algorithms, which often encounter difficulties when the problem scale grows with cloud computing.

The OEMACS is applied to cloud systems of various sizes and characteristics. Experimental results show that OEMACS has achieved the objectives of minimizing the number of active servers, improving the resource utilization, balancing different resources, and reducing power consumption. Moreover, the parameter analysis shows that the performance of OEMACS is not very sensitive to the parameters, and this makes the OEMACS more competitive. In conclusion, the OEMACS is seen an effective and efficient approach to the VMP problem.

References

[1] I. Foster, Y. Zhao, I. Raicu, and S. Y. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. IEEE Grid Comput. Environ. Workshop*, Austin, TX, USA, 2008, pp. 1–10.

[2] Z.-G. Chen, K.-J. Du, Z.-H. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *Proc. IEEE Congr. Evol. Comput.*, Sendai, Japan, 2015, pp. 708–714.

[3] H.-H. Li, Y.-W. Fu, Z.-H. Zhan, and J.-J. Li, "Renumber strategy enhanced particle swarm optimization for cloud computing resource scheduling," in *Proc. IEEE Congr. Evol. Comput.*, Sendai, Japan, 2015, pp. 870–876.

[4] A. Q. Lawey, T. E. H. El-Gorashi, and J. M. H. Elmirghani, "Distributed energy efficient clouds over core networks," *J. Lightw. Technol.*, vol. 32, no. 7, pp. 1261–1281, Apr. 1, 2014.

[5] Z. H. Zhan *et al.*, "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version," *IEEE Trans. Parallel Distrib. Syst.*, 2016, to be published, doi: 10.1109/TPDS.2016.2597826.

[6] X.-F. Liu, Z.-H. Zhan, J.-H. Lin, and J. Zhang, "Parallel differential evolution based on distributed cloud computing resources for power electronic circuit optimization," in *Proc. Genet. Evol. Comput. Conf.*, Denver, CO, USA, 2016, pp. 117–118.

[7] Z.-G. Chen *et al.*, "Deadline constrained cloud computing resources scheduling through an ant colony system approach," in *Proc. Int. Conf. Cloud Comput. Res. Innov.*, Singapore, 2015, pp. 112–119.

[8] H.-H. Li, Z.-G. Chen, Z.-H. Zhan, K.-J. Du, and J. Zhang, "Renumber coevolutionary multiswarm particle swarm optimization for multi-objective workflow scheduling on cloud computing environment," in *Proc. Genet. Evol. Comput. Conf.*, Madrid, Spain, 2015, pp. 1419–1420.

[9] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 215–228, Jul./Dec. 2013.

[10] Greenpeace. (Apr. 2010). *Make It Green: Cloud Computing and Its Contribution to Climate Change. Greenpeace International*. [Online]. Available: http://www.thegreenitreview.com/2010/04/greenpeace-reports-on-climate-impact-of.html

[11] E. Feller, L. Rilling, and C. Morin, "Energy-aware ant colony based workload placement in clouds," in *Proc. IEEE/ACM Int. Conf. Grid Comput.*, Lyon, France, 2011, pp. 26–33.

[12] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav, "It's not easy being green," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 211–222, Oct. 2012.

[13] J. Koomey, *Growth in Data Center Electricity Use 2005 to 2010*, Analytics Press, Oakland, CA, USA, Aug. 2011. [Online]. Available: http://www.analyticspress.com/datacenters.html

[14] G. Dasgupta, A. Sharma, A. Verma, A. Neogi, and R. Kothari, "Workload management for power efficiency in virtualized data centers," *Commun. ACM*, vol. 54, no. 7, pp. 131–141, Jul. 2011.

[15] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2009.

[16] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *Computer*, vol. 37, no. 11, pp. 68–76, Nov. 2004.

[17] W. Vogels, "Beyond server consolidation," *ACM Queue*, vol. 6, no. 1, pp. 20–26, Jan./Feb. 2008.

[18] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, "Exploiting spatio-temporal tradeoffs for energy-aware MapReduce in the cloud," *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1737–1751, Dec. 2012.

[19] Z. Xiao, Q. Chen, and H. P. Luo, "Automatic scaling of Internet applications for cloud computing services," *IEEE Trans. Comput.*, vol. 63, no. 5, pp. 1111–1123, May 2014.

[20] Y. Sahu, R. K. Pateriya, and R. K. Gupta, "Cloud server optimization with load balancing and green computing techniques using dynamic compare and balance algorithm," in *Proc. IEEE 5th Comput. Intell. Commun. Netw. Conf.*, Mathura, India, 2013, pp. 527–531.

[21] A. Amokrane, M. F. Zhani, R. Langar, R. Boutaba, and G. Pujolle, "Greenhead: Virtual data center embedding across distributed infrastructures," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 36–49, Jan./Jun. 2013.

[22] R. De S. Couto, S. Secci, M. E. M. Campista, and L. H. M. K. Costa, "Network design requirements for disaster resilience in IaaS clouds," *IEEE Commun. Mag.*, vol. 52, no. 10, pp. 52–58, Oct. 2014.

[23] R. K. Gupta and R. K. Pateriya, "Energy efficient virtual machine placement approach for balanced resource utilization in cloud environment," *Int. J. Cloud Comput. Super Comput.*, vol. 2, no. 1, pp. 9–20, 2015.

[24] K. Li, H. Zheng, and J. Wu, "Migration-based virtual machine placement in cloud systems," in *Proc. IEEE Int. Conf. Cloud Netw.*, San Francisco, CA, USA, 2013, pp. 83–90.

[25] M. A. Weiss, *Data Structures and Algorithm Analysis in Java*, 2nd ed. London, U.K.: Addison-Wesley, 2006.

[26] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Proc. IEEE Asia Pac. Services Comput. Conf.*, Kuala Lumpur, Malaysia, 2009, pp. 103–110.

[27] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Trans. Services Comput.*, vol. 3, no. 4, pp. 266–278, Oct./Dec. 2010.

[28] Z.-H. Zhan *et al.*, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surveys*, vol. 47, no. 4, Jul. 2015, Art. no. 63.

[29] H. Mi *et al.*, "Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers," in *Proc. IEEE Int. Conf. Services Comput.*, 2010, pp. 514–521.

[30] J. Xu and J. A. B. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. IEEE/ACM Int. Conf. Green Comput. Commun. Int. Conf. Cyber Phys. Soc. Comput.*, 2010, pp. 179–188.

[31] S. N. Wang, H. X. Gu, and G. Wu, "A new approach to multi-objective virtual machine placement in virtualized data center," in *Proc. IEEE 8th Int. Conf. Netw. Archit. Stor.*, Xi'an, China, 2013, pp. 331–335.

[32] D. Wilcox, A. McNabb, and K. Seppi, "Solving virtual machine packing with a reordering grouping genetic algorithm," in *Proc. IEEE Congr. Evol. Comput.*, New Orleans, LA, USA, 2011, pp. 362–369.

[33] Y. W. Foo, C. Goh, H. C. Lim, Z.-H. Zhan, and Y. Li, "Evolutionary neural network based energy consumption forecast for cloud computing," in *Proc. Int. Conf. Cloud Comput. Res. Innov.*, Singapore, 2015, pp. 53–64.

[34] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Virtual machine consolidation in cloud data centers using ACO metaheuristic," in *Euro-Par 2014 Parallel Processing*, vol. 8632, F. Silva, I. Dutra, and V. S. Costa, Eds. Cham, Switzerland: Springer, 2014, pp. 306–317.

[35] Y. Q. Gao, H. B. Guan, Z. W. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, 2013.

[36] M. A. Tawfeek, A. B. EI-Sisi, A. E. Keshk, and F. A. Torkey, "Virtual machine placement based on ant colony optimization for minimizing resource wastage," in *Advanced Machine Learning Technologies and Applications*, vol. 488, A. E. Hassanien, M. F. Tolba, and A. T. Azar, Eds. Cham, Switzerland: Springer, 2014, pp. 153–164.

[37] B. B. J. Suseela and V. Jeyakrishnan, "A multi-objective hybrid ACO-PSO optimization algorithm for virtual machine placement in cloud computing," *Int. J. Res. Eng. Technol.*, vol. 3, no. 4, pp. 474–476, 2014.

[38] F. Farahnakian *et al.*, "Using ant colony system to consolidate VMs for green cloud computing, " *IEEE Trans. Services Comput.*, vol. 8, no. 2, pp. 187–198, Mar./Apr. 2015.

[39] X.-F. Liu, Z.-H. Zhan, K.-J. Du, and W.-N. Chen, "Energy aware virtual machine placement scheduling in cloud computing based on ant colony optimization approach," in *Proc. ACM Genet. Evol. Comput. Conf.*, Vancouver, BC, Canada, 2014, pp. 41–48.

[40] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," in *Proc. Int. Conf. Comput. Meas. Group*, 2007, pp. 399–406.

[41] *Google App Engine*. Accessed on May 10, 2015. [Online]. Available: http://code.google.com/appengine/

[42] X. B. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 13–23, 2007.

[43] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.

[44] L. Hernando, A. Mendiburu, and J. A. Lozano, "A tunable generator of instances of permutation-based combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. 20, no. 2, pp. 165–179, Apr. 2016.

[45] H. Abbass, G. Greenwood, and E. Petraki, "The N-player trust game and its replicator dynamics," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 470–474, Jun. 2016.

[46] M. A. Rashid, F. Khatib, and M. T. Hoque, "An enhanced genetic algorithm for Ab initio protein structure prediction," *IEEE Trans. Evol. Comput.*, vol. 20, no. 4, pp. 627–644, Aug. 2016.

[47] E. Muñoz, J. M. Cadenas, Y. S. Ong, and G. Acampora, "Memetic music composition," *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 1–15, Feb. 2016.

[48] Á. Rubio-Largo, M. A. Vega-Rodríguez, and D. L. González-Álvarez, "A hybrid multiobjective memetic metaheuristic for multiple sequence alignment," *IEEE Trans. Evol. Comput.*, vol. 20, no. 4, pp. 499–514, Aug. 2016.

[49] E. G. Carrano, G. P. da Silva, E. P. Cardoso, and R. H. C. Takahashi, "Subpermutation-based evolutionary multiobjective algorithm for load restoration in power distribution networks," *IEEE Trans. Evol. Comput.*, vol. 20, no. 4, pp. 546–562, Aug. 2016.

[50] N. M. Hamza, D. L. Essam, and R. A. Sarker, "Constraint consensus mutation-based differential evolution for constrained optimization," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 447–459, Jun. 2016.

[51] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 110–124, Feb. 2016.

[52] V. A. Cicirello, "The permutation in a haystack problem and the calculus of search landscapes," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 434–446, Jun. 2016.

[53] T. J. Liao, K. Socha, M. A. M. de Oca, T. Stützle, and M. Dorigo, "Ant colony optimization for mixed-variable optimization problems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 503–518, Aug. 2014.

[54] Q. Yang *et al.*, "Adaptive multimodal continuous ant colony optimization," *IEEE Trans. Evol. Comput.*, 2016, to be published, doi: 10.1109/TEVC.2016.2591064.

[55] Y. H. Li, Z.-H. Zhan, S. J. Lin, J. Zhang, and X. N. Luo, "Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems," *Inf. Sci.*, vol. 293, no. 1, pp. 370–382, Feb. 2015.
[56] Z.-H. Zhan *et al.*, "Multiple populations for multiple objectives: A coevolutionary technique for solving multiobjective optimization problems," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 445–463, Apr. 2013.
[57] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *J. Heuristics*, vol. 2, no. 1, pp. 5–30, 1996.

Xiao-Fang Liu (S'14) received the B.S. degree in computer science from Sun Yat-sen University, Guangzhou, China, in 2014, where she is currently pursuing the Ph.D. degree.

Her current research interests include artificial intelligence, evolutionary computation, swarm intelligence, and their applications in design and optimization such as cloud computing resources scheduling.

Zhi-Hui Zhan (S'09–M'13) received the bachelor's and Ph.D. degrees from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively.

He is currently a Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, where he is also appointed as the Pearl River Scholar Young Professor in 2016. His current research interests include evolutionary computation algorithms, swarm intelligence algorithms, and their applications in real-world problems, and in environments of cloud computing and big data.

Dr. Zhan was a recipient of the China Computer Federation Outstanding Dissertation in 2013 for his doctoral dissertation, the Natural Science Foundation for Distinguished Young Scientists of Guangdong Province, China, in 2014, the Pearl River New Star in Science and Technology in 2015, and one of the Most Cited Chinese Researchers in Computer Science.

Jeremiah D. Deng (M'01) received the B.Eng. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 1989, the M.Eng. degree from the South China University of Technology (SCUT), Guangzhou, China, in 1992, and the Ph.D. degree from SCUT and the University of Hong Kong, Hong Kong, in 1995.

He joined SCUT, as a Lecturer, in 1995. He then joined the University of Otago, Dunedin, New Zealand, as a Research Fellow, in 1999, where he is currently an Associate Professor with the Department of Information Science. He has published over 80 technical papers in the areas of machine learning, signal processing, and mobile computing.

Yun Li (S'87–M'90) received the B.S. degree in radio electronics science from Sichuan University, Chengdu, China, in 1984, the M.Eng. degree in electronic engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, in 1987, and the Ph.D. degree in parallel processing for control engineering from the University of Strathclyde, Glasgow, U.K., in 1990.

From 1989 to 1990, he was with United Kingdom National Engineering Laboratory and Industrial Systems and Control Ltd., Glasgow. In 1991, he joined the University of Glasgow, Glasgow, as a Lecturer, where he is currently a Professor of systems engineering. He served as the Founding Director of the University of Glasgow Singapore, Singapore, from 2011 to 2013. He established the IEEE Control System Society and European Network of Excellence workgroups on computer-automated design) in 1998, and wrote the popular online courseware "GA Demo" for interactive evolutionary computation in 1997. He is currently a Professor and a Visiting Professor with the University of Glasgow, UESTC, and the Dongguan University of Technology, Dongguan, China. He has 200 publications. His current research interest includes smart design with market informatics via the cloud to complete the value chain for Industry 4.0.

Dr. Li is a Chartered Engineer and an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the *SM Journal of Engineering Sciences*.

Tianlong Gu received the M.Eng. degree from Xidian University, Xi'an, China, in 1987, and the Ph.D. degree from Zhejiang University, Hangzhou, China, in 1996.

He was a Research Fellow with the School of Electrical and Computer Engineering, Curtin University of Technology, Bentley, WA, Australia, and a Post-Doctoral Fellow with the School of Engineering, Murdoch University, Perth, WA, Australia, from 1998 to 2002. He is currently a Professor with the School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin, China. His current research interests include formal methods, data and knowledge engineering, software engineering, and information security protocol.

Jun Zhang (M'02–SM'08–F'17) received the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2002.

He is currently a Changjiang Chair Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His current research interests include computational intelligence, cloud computing, high performance computing, data mining, wireless sensor networks, operations research, and power electronic circuits. He has published over 100 technical papers in the above areas.

Dr. Zhang was a recipient of the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011 and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, and the IEEE TRANSACTIONS ON CYBERNETICS.