Journal of Computer and System Sciences ••• (••••) •••–•••

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

A multi-objective ant colony system algorithm for virtual machine placement in cloud computing

Yongqiang Gao [a], Haibing Guan [a,*], Zhengwei Qi [a], Yang Hou [b], Liang Liu [c]

[a] *Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*
[b] *UM-SJTU Joint Institute, Shanghai Jiao Tong University, Shanghai 200240, China*
[c] *IBM Research – China, Beijing 100193, China*

ARTICLE INFO

Article history:
Received 6 December 2011
Received in revised form 26 April 2012
Accepted 22 February 2013
Available online xxxx

Keywords:
Multi-objective optimization
Ant colony optimization
Virtual machine placement
Cloud computing

ABSTRACT

Virtual machine placement is a process of mapping virtual machines to physical machines. The optimal placement is important for improving power efficiency and resource utilization in a cloud computing environment. In this paper, we propose a multi-objective ant colony system algorithm for the virtual machine placement problem. The goal is to efficiently obtain a set of non-dominated solutions (the Pareto set) that simultaneously minimize total resource wastage and power consumption. The proposed algorithm is tested with some instances from the literature. Its solution performance is compared to that of an existing multi-objective genetic algorithm and two single-objective algorithms, a well-known bin-packing algorithm and a max–min ant system (MMAS) algorithm. The results show that the proposed algorithm is more efficient and effective than the methods we compared it to.

1. Introduction

In recent year, cloud computing has become a popular computing paradigm for hosting and delivering services over the Internet [1]. There are three major types of cloud computing: infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). The adoption and deployment of cloud computing platforms have many attractive benefits, such as reliability, quality of service and robustness [2]. To the consumer, the cloud appears to be infinite, and the consumer can purchase as much or as little computing power as they need. From a provider's perspective, the key issue is to maximize profits by minimizing the operational costs. In this regard, power management in cloud data centers is becoming a crucial issue since it dominates the operational costs. Moreover, power consumption in large-scale computer systems like clouds also raises many other serious issues including carbon dioxide and system reliability. The emergence of cloud computing has made a tremendous impact on the information technology (IT) industry over the past few years, where large companies such as Amazon, Google, Salesforce, IBM, Microsoft, and Oracle have begun to establish new data centers for hosting cloud computing applications in various locations around the world to provide redundancy and ensure reliability in case of site failures.

There are a number of key technologies that make cloud computing possible. One of the most important is virtualization. Virtualization provides a promising approach through which hardware resources on one or more machines can be divided through partial or complete machine simulation, time-sharing, hardware and software partitioning into multiple execution environments, each of which can act as a complete system. Virtualization enables dynamic sharing of physical resources

* Corresponding author. Fax: +86 21 3420 7150.
E-mail addresses: gaoyongqiang@sjtu.edu.cn (Y. Gao), hbguan@sjtu.edu.cn (H. Guan), qizhwei@sjtu.edu.cn (Z. Qi), yang8844@sjtu.edu.cn (Y. Hou), liuliang@cn.ibm.com (L. Liu).

0022-0000/$ – see front matter
http://dx.doi.org/10.1016/j.jcss.2013.02.004

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

in cloud computing environments, allowing multiple applications to run in different performance-isolated platforms called virtual machines (VMs) in a single physical server. This technology also enables on-demand or utility computing—a just-in-time resource provisioning model in which computing resources such as CPU, memory, and disk space are made available to applications only as needed and not allocated statically based on the peak workload demand [3]. Through virtualization, a cloud provider can ensure the quality of service (QoS) delivered to the users while achieving a high server utilization and energy efficiency.

Virtual machine placement is a process of mapping virtual machines to physical machines. As virtualization is a core technology of cloud computing, the problem of virtual machine (VM) placement has become a hot topic recently. This VM placement is an important approach for improving power efficiency and resource utilization in cloud infrastructures. Several research works [4,5] addressed the importance of placing VMs appropriately. Vogels [6] quoted the benefit of packing VMs efficiently in server consolidation. The proxy placement [7–9] and object placement/replacement [10,11] for transparent data replication bear some resemblance to the issues we face since they all attempt to exploit the flexibility available in determining proper placement. The following are some of the approaches that have been used to solve the virtual machine placement problem.

Linear programming. A traditional analytical approach is linear programming. For example, Chaisiri et al. [12] presented a nice algorithm for optimal placement of virtual machines on physical machines. The goal is that the number of used nodes is minimum. They provided approaches based on linear and quadratic programming. In [13] and [14], the authors described linear programming formulations of server consolidation problems. They also designed extension constraints for allocating virtual machines to a specific set of physical servers that contain some unique attribute, restricting the number of virtual machines in a single physical server, ensuring that some virtual machines are assigned to different physical servers and limiting the total number of migrations. In addition, they developed an LP-relaxation-based heuristic for minimizing the cost of solving the linear programming problem.

Genetic algorithm. Another approach to this problem is to use a genetic algorithm. In [15], the authors proposed a genetic algorithm based approach, namely GABA, to adaptively self-reconfigure the VMs in cloud data centers consisting of heterogeneous nodes. GABA can efficiently decide the optimal physical locations of VMs according to time-varying requirements and the dynamic environmental conditions. In [16], the VM placement problem is formulated as a multi-objective optimization problem of simultaneously minimizing total resource wastage, power consumption and thermal dissipation costs. A modified genetic algorithm with fuzzy multi-objective evaluation was proposed for efficiently searching the large solution space and conveniently combining possibly conflicting objectives.

Constraint programming. Constraint programming methods have also been applied for VM placement in various environments. Van et al. [17] proposed a resource management framework combining a utility-based dynamic virtual machine provisioning manager and a dynamic VM placement manager. The VM provisioning and placement problems were expressed as two constraint satisfaction problems. In [18], the authors proposed the Entropy resource manager for homogeneous clusters, which performs dynamic consolidation based on constraint programming and takes into account both the problem of allocating the VMs to the available nodes and the problem of how to migrate the VMs to these nodes.

Bin packing. The problem of VM placement in a data center is often formulated as a variant of the vector bin-packing problem, which is an NP-hard optimization problem [19]. Various heuristics [20–24] have been proposed for this problem. For example, the pMapper system [21] tackled power-cost tradeoffs under a fixed performance constraint by minimizing migration costs while packing VMs in a small number of machines. The packing algorithm is an extension of the first fit decreasing (FFD) heuristic. Moreover, Feller et al. [24] proposed a single-objective algorithm based on the MMAS metaheuristic to minimize the number of physical machines required to support the current load.

The majority of the studies on virtual machine placement focus on a single criterion. However, many real-world problems require taking multiple criteria into account. For this reason, recent research tends to look at the multiple-objective situation. Therefore, in this paper the problem of VM placement is formulated as a multi-objective combinatorial optimization problem aiming to simultaneously optimize total resource wastage and power consumption. A modified version of the ant colony system (ACS) algorithm is proposed and designed to deal effectively with the potential large solution space for large-scale data centers. To the best of our knowledge, this study is the first application of the ACS metaheuristic to a multi-objective virtual machine placement problem where both power consumption and resource wastage should be minimized. The performance of the proposed algorithm is compared to that of a multi-objective genetic algorithm and two single-objective algorithms, a well-known bin-packing algorithm and an MMAS algorithm. Computational experiments on benchmark problems are carried out. The results show that the proposed algorithm can compete efficiently with other promising approaches to the problem.

The remainder of this paper is organized as follows. In Section 2, we provide an overview of both ant colony optimization and evolutionary multi-objective optimization, and present a simple procedure to perform VM placement in a virtualized cloud environment. Section 3 formulates the virtual machine placement problem. In Section 4, the proposed algorithm is presented. The computational results on benchmark problems are given in Section 5. We conclude in Section 6.

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

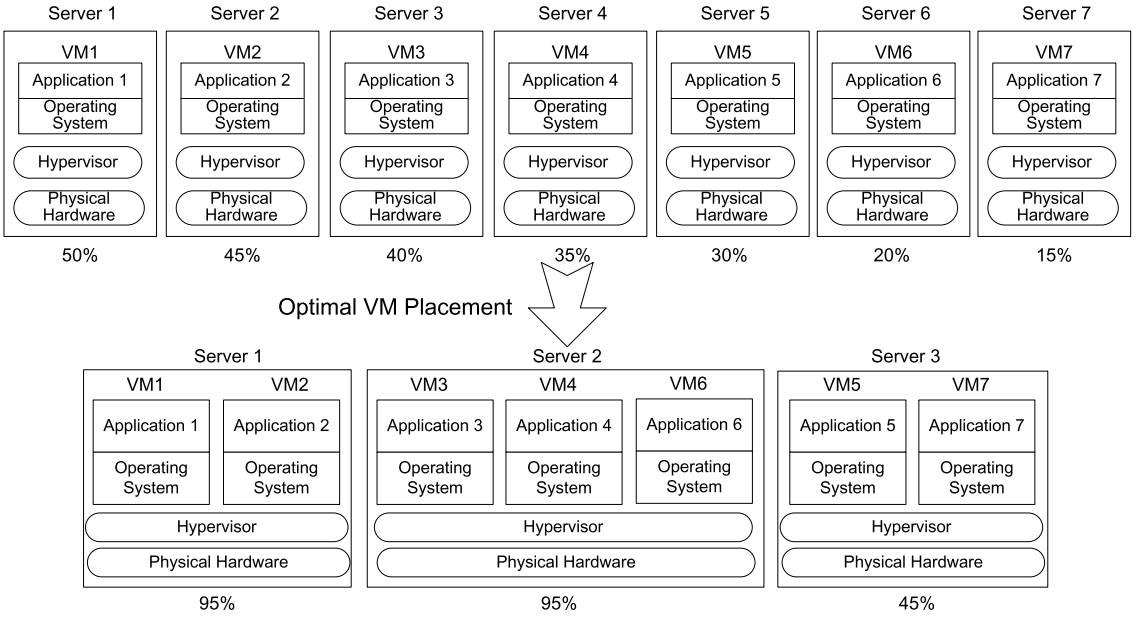

Fig. 1. An example of VM placement in a virtualized environment.

2. Background

2.1. An example of VM placement in a virtualized cloud environment

For example, let us consider the situation depicted in Fig. 1. We have seven servers, each of which has a quad-core processor which is capable of executing four VMs. The system is currently hosting seven virtualized application labeled Application 1 to Application 7. A simple process for VM placement is as follows [25].

(1) For each server, compute application resource requirement using server's resource usage statistics over a period of time (e.g., several weeks).
(2) Choose a target server with compatible virtualization software, comparable CPU types, similar network connectivity, and usage of shared storage.
(3) Place the first virtual machine on the first server in step 2. Place the second virtual machine on the same server if it can satisfy the resource requirements. If not, add a new physical machine and place the VM on this new machine. Continue this step until each of the VMs has been placed on a physical machine, adding a new physical machine when required.
(4) The set of resulting hosts at the end of step 3 comprises the consolidated server farm. Finally the number of servers required in the cluster is reduced from 7 down to 3.

2.2. Ant colony optimization

Ant Colony Optimization (ACO) is a metaheuristic inspired by the observation of real ant colonies and based upon their collective foraging behavior [26]. Ants are social insects and live in colonies. Their behavior is governed by the goal of colony survival. When searching for food, ants frequently travel between their nest and food sources. At the beginning, ants explore the area surrounding their nest in a random manner. While moving, ants deposit special substances called pheromones along their paths. Ants can smell pheromones. When choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromones that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails will guide other ants to the food source. The indirect communication between the ants via pheromone trails enables them to find the shortest paths between their nest and food sources.

Ant colony optimization has been successfully applied to solve numerous optimization problems such as the traveling salesman problem [27], the flow shop scheduling problem [28] and the quadratic assignment problem [29]. Besides its original domain of combinatorial optimization, ACO is also now used to solve continuous optimization problems [30]. Some extensions of ACO algorithms have been proposed in the literature such as ACS [27], Ant System (AS) [31] and MMAS [32]. Recently there have also been a number of studies extending ACO to the field of multi-objective optimization [33]. These algorithms mainly differ with respect to the three following points.

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

Pheromone update. When updating pheromone trails, one has to decide on which of the constructed solutions to lay pheromones. There are usually two strategies to update the pheromone trails. A first strategy is to select the iteration-best or best-so-far solutions to update the pheromone matrices, with respect to each objective. A second strategy is to collect and store the non-dominated solutions in an external set. Only the solutions in the non-dominated set are allowed to update the pheromones.

Definition of pheromone and heuristic information. At each step of the construction of a solution, a candidate is chosen relative to a transition probability which depends on two factors: a pheromone factor and a heuristic factor. There are two approaches to define the pheromone/heuristic information: using one or multiple matrices. When only one matrix is utilized, the pheromone information associated with each objective is combined to reduce the multiple objectives into a single one. If multiple matrices are used, usually each matrix corresponds to one objective. With respect to the pheromone information, each matrix may contain different values depending on the implementation strategy applied. The same applies to the heuristic information.

Pheromone and heuristic aggregation. Whenever multiple matrices are used, one must use some form of aggregation procedure to aggregate the pheromone/heuristic matrices. There are three common strategies for this: (1) the weighted sum, where matrices are aggregated by a weighted sum; (2) the weighted product, where matrices are aggregated by a weighted product; and (3) random, where at each step a random objective is selected to be optimized. Whenever weights are used for aggregating multiple matrices, two strategies can be applied for setting the weights used at each iteration of the algorithm: (a) dynamically, where each ant may be assigned a different weight from the other ants at each iteration; (b) fixed, where we can assign to all ants the same weight and each objective has the same importance during the entire algorithm run.

2.3. Evolutionary multi-objective optimization

Multi-objective evolutionary algorithms (MOEAs) are stochastic optimization methods, which usually use a population-based approach to find Pareto optimal solutions [34]. The majority of existing MOEAs use the concept of dominance during selection. Therefore, we focus here on the class of dominance-based MOEAs only. The formal definition of the dominance concept is as follows. Let us consider, without loss of generality, a multi-objective minimization problem with m parameters (decision variables) and n objectives:

$$\text{Minimize } \vec{f}(\vec{x}) = \left[f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m)\right]$$

where

$$\vec{x} = (x_1, \ldots, x_m) \in X$$
$$\vec{f} = (f_1, \ldots, f_n) \in Y$$

where $\vec{x}$ is called the decision (parameter) vector, X parameter space, $\vec{f}$ objective vector and Y objective space. Here we consider the term "solution" as a decision vector and the term "point" as the corresponding objective vector. A solution $\vec{x}_1$ is said to dominate the other solution $\vec{x}_2$, if the both following conditions are true (see [35,36]):

(1) The solution $\vec{x}_1$ is not worse than $\vec{x}_2$ in any objective.
(2) The solution $\vec{x}_1$ is strictly better than $\vec{x}_2$ in at least one objective.

All points which are not dominated by any other point are called the non-dominated points. Usually the non-dominated points together constitute a front in the objective space and are often visualized to represent a non-domination front. The points lying on the non-domination front, by definition, do not get dominated by any other point in the objective space, hence they are Pareto optimal points (together they make up the Pareto optimal front), and the corresponding variable vectors are called Pareto optimal solutions.

The above concept can also be extended to find a non-dominated solution set. Let us consider a set of N solutions, each having M ($M > 1$) objective function values. In our work, the following procedure is used to find the non-dominated solution set [37]:

Step 1 Begin with $i = 1$.
Step 2 For all $j \neq i$, compare solutions $\vec{x}_i$ and $\vec{x}_j$ for domination using the above two conditions for all M objectives.
Step 3 If for any j, $\vec{x}_i$ is dominated by $\vec{x}_j$, mark $\vec{x}_i$ as "dominated". Increment i by one and Go to Step 2.
Step 4 If all solutions (that is, when $i = N$ is reached) in the set are considered, Go to Step 5, else increment i by one and Go to Step 2.
Step 5 All solutions that are not marked "dominated" are non-dominated solutions.

3. Problem statement and formulation

In a cloud environment, we have a pool of server nodes with applications running on them. Suppose that the cluster is fully virtualized and all the applications are running on VMs. The problem of VM placement across a pool of server nodes

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

is related to the multidimensional vector packing problems. Dimensions in the packing problem are resource utilizations. In our work we use two dimensions to characterize a VM and a server node—CPU and memory. We do not consider the disk size dimension because we assume that network-attached storage (NAS) is used as main storage across the cluster. If two VMs are running on the same server, the CPU utilization of the server is estimated as the sum of the CPU utilizations of the two VMs. This is the case with memory resources. For example, let (20%, 30%) be a pair of the CPU and memory requests of a VM, and (35%, 40%) be that of another VM. Then, the utilizations of a server accommodating the two VMs are estimated at (55%, 70%), i.e., the sum of the vectors. To prevent CPU and memory usage of a server from reaching 100%, we have to impose an upper bound on resource utilization of a single server with some threshold value. The main idea behind this is that 100% utilization can cause severe performance degradation and VM live migration technology consumes some amount of CPU processing capability on the migrating node.

3.1. Resource wastage modeling

The remaining resources available on each server may vary greatly with different VM placement solutions. To fully utilize multidimensional resources, the following equation is used to calculate the potential cost of wasted resources:

$$W_j = \frac{|L_j^p - L_j^m| + \varepsilon}{U_j^p + U_j^m}$$

where W_j denotes the resource wastage of the j-th server, U_j^p and U_j^m represent the normalized CPU and memory resource usage (i.e., the ratio of used resource to total resource). L_j^p and L_j^m represent the normalized remaining CPU and memory resource. ε is a very small positive real number and its value is set to be 0.0001. The key idea behind the above equation is to make effective use of the resources in all dimensions and balance the resources left on each server along different dimensions.

3.2. Power consumption modeling

Recent studies show that the power consumption of servers can be accurately described by a linear relationship between the power consumption and CPU utilization [38]. This linear relationship is also confirmed by our profiling conducted on a Dell server. In order to save energy, servers are turned off when they are idle. Hence, their idle power is not part of the total energy consumption. Finally, we defined the power consumption of the j-th server as a function of the CPU utilization as shown in Eq. (1).

$$P_j = \begin{cases} (P_j^{busy} - P_j^{idle}) \times U_j^p + P_j^{idle}, & U_j^C > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where P_j^{idle} and P_j^{busy} are the average power values when the j-th server is idle and fully utilized, respectively. In our simulation experiments, the values have been fixed to 162 and 215 Watt according to the measurements performed on a Dell server.

3.3. Optimization formulation

Next, we formalize the VM placement optimization problem. Suppose that we are given n VMs (applications) $i \in I$ that are to be placed on m servers $j \in J$. For simplicity, we assume that none of the VMs requires more resource than can be provided by a single server. Let R_{p_i} be CPU demand of each VM, T_{pj} be the threshold of CPU utilization associated with each server, R_{mi} be the memory demand of each VM, and T_{mj} be the threshold of memory utilization associated with each server. We use two binary variables x_{ij} and y_j. The binary variable x_{ij} indicates if *VM i* is assigned to *server j* and the binary variable y_j indicates whether *server j* is in use or not. Our objective is to simultaneously minimize the power consumption and the resource wastage. The placement problem can therefore be formulated as:

$$\text{Minimize } \sum_{j=1}^{m} P_j = \sum_{j=1}^{m} \left[y_j \times \left(\left(P_j^{busy} - P_j^{idle}\right) \times \sum_{i=1}^{n} (x_{ij} \cdot R_{p_i}) + P_j^{idle} \right) \right]$$

$$\text{Minimize } \sum_{j=1}^{m} W_j = \sum_{j=1}^{m} \left[y_j \times \frac{|(T_{Pj} - \sum_{i=1}^{n}(x_{ij} \cdot R_{p_i})) - (T_{mj} - \sum_{i=1}^{n}(x_{ij} \cdot R_{m_i}))| + \varepsilon}{\sum_{i=1}^{n}(x_{ij} \cdot R_{p_i}) + \sum_{i=1}^{n}(x_{ij} \cdot R_{m_i})} \right]$$

Subject to:

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i \in I \tag{2}$$

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

$$\sum_{i=1}^{n} R_{p_i} \cdot x_{ij} \leqslant T_{p_j} \cdot y_j \quad \forall j \in J \tag{3}$$

$$\sum_{i=1}^{n} R_{mi} \cdot x_{ij} \leqslant T_{mj} \cdot y_j \quad \forall j \in J \tag{4}$$

$$y_j, x_{ij} \in \{0, 1\} \quad \forall j \in J \text{ and } \forall i \in I \tag{5}$$

Constraint (2) assigns a *VM i* to only one of the servers. Constraints (3) and (4) model the capacity constraint of the server. Constraint (5) defines the domain of the variables of the problem. Given a set of n virtual machines and a set of m physical machines, there are a total of m^n possible VM placement solutions. It is therefore typically impractical to make a complete enumeration of all possible solutions to find the best solutions. The following shows how to apply an ACO algorithm to efficiently search for good solutions in large solution spaces.

4. The description of the proposed multi-objective ant colony system algorithm

The algorithm proposed to solve the problems formulated in Section 3 is mainly based on an ACS. A feasible and complete solution of the formulated multi-objective VM placement problem is considered as a permutation of VM assignment. The terms "host" and "server" will be used interchangeably in this paper. An assignment of a VM to a host is called a movement and represented by *VM-Host*.

The pseudocode of the proposed multi-objective ant colony system algorithm (VMPACS) is depicted in Fig. 2. This algorithm works as follows: in an initialization phase, the parameters are initialized and all the pheromone trails are set to τ_0. In the iterative part each ant receives all VM requests, introduces a physical server and starts assigning VMs to hosts. This is achieved by the use of a pseudo-random-proportional rule, which describes the desirability for an ant to choose a particular VM as the next one to pack into its current host. This rule is based on the information about the current pheromone concentration on the movement and a heuristic which guides the ants towards choosing the most promising VMs. A local pheromone update is performed once an artificial ant has built a movement. After all ants have constructed their solutions, a global update is performed with each solution of the current Pareto set.

4.1. Definition of the pheromone trail and the heuristic information

Similarly to the general implementation of ACO algorithms, VMPACS starts with a pheromone trails matrix and a heuristic information matrix. The quality of an ACO implementation depends greatly on the definition of the meaning of the pheromone trail [31]. It is crucial to choose a definition which conforms to the feature of the problem. One may consider two different pheromone structures: one that associates a pheromone trail with every movement *VM-Host*, or one that associates a pheromone trail with every pair of VMs. In this paper, the first way of laying pheromone trails is used, i.e., the pheromone trail $\tau_{i,j}$ will be defined as the favorability of packing VM i into host j. In the initialization phase, initial pheromone level is calculated by $\tau_0 = 1/[n \cdot (P'(S_0) + W(S_0))]$, where n is the number of VMs, S_0 is the solution generated by the FFD heuristic and $W(S_0)$ is the resource wastage of the solution S_0. $P'(S_0)$ is the normalized power consumption of the solution S_0 and its value is calculated according to the following equation:

$$P'(S_0) = \sum_{j=1}^{m} \left(P_j / P_j^{Max}\right)$$

where P_j^{Max} is the peak power consumption of server j.

Apart from pheromone trails, another important factor in an ACO application is the choice of a good heuristic, which will be used in combination with the pheromone information to build solutions. It guides the probabilistic solution construction of ants with problem-specific knowledge. The heuristic information is denoted by $\eta_{i,j}$. This information indicates the desirability of assigning *VM i* to *host j*. In order to accurately assess the desirability of each move, the heuristic information is dynamically computed according to the current state of the ant. Since the heuristic information is calculated for all movements in all ants, it may significantly affect the efficiency of the algorithm. To overcome such difficulties it therefore should be computed in an efficient manner. The proposed method to calculate the heuristic information considers the partial contribution of each move to the objective function value. Let *PL* be a list composed of all the servers. When constructing a solution, every ant starts with the set of all VMs to be placed and the list PL arranged in randomly order. It initially assigns VMs one by one to the first host in the list PL, then assigns to the second host and so on till all VMs are assigned. Therefore, while calculating the value of $\eta_{i,j}$, the permutation of VM assignments from the *host* 1 to *host j* is known. The partial contribution of assigning *VM i* to *host j* for the first objective function can therefore be calculated as follows:

$$\eta_{i,j,1} = \frac{1}{\varepsilon + \sum_{v=1}^{j} (P_v / P_v^{Max})}$$

Input: Set of VMs and set of hosts with their associated resource demand and the thresholds of resource utilization respectively, Set of parameters
Output: a Pareto set P

```
/* Initialization */
1. Set values of parameter , ρ_l, ρ_g, α, τ_0, q_0, NA (number of ants) and M (number of iteration)
2. Initialize Pareto set P as empty
3. Initialize all pheromone values to τ_0
/* Iterative loop */
4. Repeat
5.    For j=1 to NA (Number of ants) do
        /* Construct a solution*/
6.      Sort the server list PL in random order
7.      Repeat
8.        Introduce a new sever from the server list PL
9.        Repeat
10.         For each remaining VM that can be packed into the current server do
11.           Calculate the desirability of the movement according to Eq. (6)
12.           Calculate the probability of the movement according to Eq. (7)
13.         End For
            /* Choice of the virtual machine to assign */
14.         Draw q
15.         If  q ≤ q_0  Then
16.           exploitation
17.         Else
18.           exploration
19.         End if
            /* Local pheromone updating*/
20.         Apply the local updating rule (Eq. (8))
21.       Until no remaining VM fits in the server anymore
22.     Until all VMs are placed
23.   End For
        /* Evaluation */
24.   Calculate the values of the two objectives for each solution in current ant population
25.   If a solution in the current ant population is not dominated by any other solutions in the current
      population and the non-dominated solutions in the Pareto set P, this solution is added to
      the set P. Then all solutions dominated by the added one are eliminated from the set P. (see Sec. 2.3)
      /*Global pheromone updating*/
26.   For each non-dominated solution in the Pareto set P do
27.     Apply the global updating rule (Eq. (9))
28.   End For
29.Until the maximum number of iterations is reached
30.Return the Pareto set P
```

Fig. 2. The VMPACS algorithm.

Similarly to the first objective function, the partial contribution of assigning *VM i* to *host j* for the second one can be calculated as follows:

$$\eta_{i,j,2} = \frac{1}{\varepsilon + \sum_{v=1}^{j} W_v}$$

There are several ways to combine desirability in multi-objective problem to find the total desirability of each movement. In this paper we propose the following formula to calculate the total desirability of assigning *VM i* to *host j*:

$$\eta_{i,j} = \eta_{i,j,1} + \eta_{i,j,2} \tag{6}$$

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

4.2. Constructing a solution

In the process of making assignments, the ant k selects a VM i as the next one to pack into its current host j according to the following pseudo-random-proportional rule.

$$i = \begin{cases} \arg\max_{u \in \Omega_k(j)} \{\alpha \times \tau_{u,j} + (1-\alpha) \times \eta_{u,j}\}, & q \leqslant q_0 \\ s, & \text{otherwise} \end{cases}$$

where α is a parameter that allow a user to control the relative importance of pheromone trail and q is a random number uniformly distributed in $[0, 1]$. If q is greater than q_0, this process is called exploration, otherwise it is called exploitation. q_0 is a fixed parameter ($0 \leqslant q_0 \leqslant 1$) determined by the relative importance of exploitation of accumulated knowledge about the problem versus exploration of new movements. $\Omega_k(j)$ is the set of VMs that qualify for inclusion in the current host j, that is, $\Omega_k(j) = \{i \in \{1, \ldots, n\} | (\sum_{u=1}^{m} x_{iu} = 0) \wedge ((\sum_{u=1}^{n} (x_{uj} \times R_{pu}) + R_{pi}) \leqslant T_{pj}) \wedge ((\sum_{u=1}^{n} (x_{uj} \times R_{mu}) + R_{mi}) \leqslant T_{mj})\}$. $\eta_{i,j}$ is defined in Eq. (6) above. The pheromone value $\tau_{i,j}$ is given in Eq. (9) below. s is a random variable selected according to the following random-proportional rule probability distribution [39], which is the probability that ant k chooses to assign VM i to host j:

$$p_{i,j}^{k} = \begin{cases} \frac{\alpha \times \tau_{i,j} + (1-\alpha) \times \eta_{i,j}}{\sum_{u \in \Omega_k(j)} (\alpha \times \tau_{u,j} + (1-\alpha) \times \eta_{u,j})}, & i \in \Omega_k(j) \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

There are two reasons for adopting the above method to calculate the selection probability. The first is the simplicity of the approach proposed in [39] as only one control parameter, i.e. α, is used to map the relative importance of quantity of pheromone and the desirability of each movement. The second reason is the computational efficiency of this method as multiplication operations are used instead of exponentiations.

4.3. Pheromone trail update

Another vital component of VMPACS is the update of pheromone trails. The pheromone trail value can either increase, as ants deposit pheromone, or decrease, due to pheromone evaporation. The deposit of new pheromone is based on the fact that the information contained in some good solutions should be indicated by pheromone trails and the movement included in these good solutions will be biased by other ants constructing subsequent solutions. However, pheromone evaporation also implements a useful form of forgetting: it avoids a too rapid convergence of the algorithm toward a suboptimal region, therefore favoring the exploration of new areas of the search space. It is a kind of diversification strategy. In our proposed algorithm, the pheromone updating process includes two steps: a local pheromone update and a global pheromone update. While constructing an assignment of *VM i* to *host j*, an ant decreases the pheromone trail level between *VM i* and *host j* by applying the following local updating rule:

$$\tau_{i,j}(t) = (1 - \rho_l)\tau_{i,j}(t-1) + \rho_l \cdot \tau_0 \tag{8}$$

where τ_0 is the initial pheromone level and ρ_l ($0 < \rho_l < 1$) is the local pheromone evaporating parameter.

The global updating rule is applied after all ants have finished building a solution. Since all non-dominated or Pareto solutions are considered as optimal or best solutions for a multi-objective optimization problem, we suppose that all non-dominated solutions have the same and highest quality and all dominated solutions must be omitted. Therefore, the global update is performed for each solution S of the current Pareto set by applying the following rule:

$$\tau_{i,j}(t) = (1 - \rho_g)\tau_{i,j}(t-1) + \frac{\rho_g \cdot \lambda}{P'(S) + W(S)} \tag{9}$$

where,

$$\lambda = \frac{NA}{t - NI_s + 1} \tag{10}$$

In Eq. (9), ρ_g ($0 < \rho_g < 1$) is the pheromone evaporation parameter of global updating. The global non-dominated solutions, that form the Pareto set, are stored in an external set. If a solution in the current iteration is not dominated by any other solutions in the current iteration or the external set of non-dominated solutions, this solution is added to the external set and the quantity of pheromone in all movements which constructed it will be increased. Then all solutions dominated by the added one are eliminated from the external set. In Eq. (10), NA is the number of ants and NI_s represents the number of iterations that solution S has resided in the external set. λ is an adaptive coefficient, used to control how a solution in the external set contributes to pheromone information over time. This global updating rule tries to increasing the learning of ants.

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

5. Computational results

In this section, we use some simulation experiments to evaluate the proposed algorithm with respect to performance and scalability. The performance of the proposed ant algorithm is compared to that of a multi-objective grouping genetic algorithm (MGGA) proposed in [16], a single-objective ACO (SACO) algorithm proposed in [24] and a single-objective FFD algorithm proposed in [40]. The programs for the proposed algorithm, MGGA algorithm and FFD heuristic were coded in the Java language and ran on an Intel Pentium® Dual-Core processor with 2.50 GHz CPU and 3 GB RAM. The settings for various parameters in VMPACS have a direct effect on the algorithm performance. Appropriate parameter values were determined on the basis of preliminary experiments. The final parameter settings were determined to be $NA = 10$, $M = 100$, $\alpha = 0.45$, $\rho_l = \rho_g = 0.35$, and $q_0 = 0.8$. In the case of the MGGA algorithm the population size is 12. The initial population was generated randomly. The crossover rate is 0.7 and the mutation rate is 0.05. The maximum number of generations for each search process is 10.

With the above configuration, we randomly generated problem instances. The instances were a demand set of CPU and memory utilizations for 200 VMs. The number of servers was set to the number of VMs in order to support the worst VM placement scenario, in which only one VM is assigned per server. For simplicity, we simulated homogeneous server environments but the proposed approach can be used for the case of heterogeneous servers. After the VM placement algorithm was finished, if there were several non-dominated solutions, a solution belonging to the set of non-dominated solutions was randomly chosen.

Every test was repeated with 20 runs for each instance and the average results over 20 independent runs are reported. We introduced the linear correlations of CPU and memory utilizations into the instances and used the method proposed in [40] to generate random sequences of CPU and memory utilizations in the experiments that had several correlations. The detailed algorithm is as follows:

1. For $i = 1$ to n do
2. $\quad R_{pi} = \text{rand}(2\overline{R_p})$;
3. $\quad R_{mi} = \text{rand}(\overline{R_m})$;
4. $\quad r = \text{rand}(1.0)$;
5. $\quad$ If $(r < P \wedge R_{pi} \geqslant \overline{R_p}) \vee (r \geqslant P \wedge R_{pi} < \overline{R_p})$ then
6. $\quad\quad R_{mi} = R_{mi} + \overline{R_m}$;
7. $\quad$ End if
8. End For

where rand(a) is a function that returns a random, uniformly distributed number of double type in the range $[0, a)$; $\overline{R_p}$ represents the reference CPU utilization, and $\overline{R_m}$ represents the reference memory utilization, and the probability P is a reference value. We can control the correlations of CPU and memory utilizations to some extent by varying probability P.

Two kinds of the reference values and five probabilities were used in the experiments. We set both $\overline{R_p}$ and $\overline{R_m}$ to 25% and then 45%. The distributions of CPU and memory utilizations are in the range $[0, 50\%)$ when $\overline{R_p} = \overline{R_m} = 25\%$, and $[0, 90\%)$ when $\overline{R_p} = \overline{R_m} = 45\%$. For $\overline{R_p} = \overline{R_m} = 25\%$, we set P to 0.00, 0.25, 0.50, 0.75, and 1.0, and then the average correlation coefficients became -0.754, -0.348, -0.072, 0.371, and 0.755 for each set of instances. These correlation coefficients correspond to strong-negative, weak-negative, no, weak-positive, and strong-positive correlations. We similarly set P for $\overline{R_p} = \overline{R_m} = 45\%$. The correlation coefficients were then -0.755, -0.374, -0.052, 0.398, and 0.751. We set the thresholds of both utilizations to $T_{pi} = T_{mi} = 90\%$ throughout the experiments.

5.1. Comparison of VMPACS with MGGA

To evaluate the effectiveness of the proposal VM placement algorithm, its performance is compared to that of the MGGA algorithm, which is used as the benchmark because it is an effective and efficient method used by [16] to solve multi-objective VM placement problems. We computed two measures, overall non-dominated vector generation (ONVG) [41] and Spacing (SP) [42] for each algorithm. ONVG and SP can be calculated as

$$ONVG = |Y_{known}|_c$$

$$SP = \sqrt{\frac{1}{|Y_{known}|_c - 1} \sum_{i=1}^{|Y_{known}|_c} (\bar{d} - d_i)^2}$$

where Y_{known} denotes the calculated Pareto front, $|\,|_c$ denotes cardinality, $d_i = \min_j(\sum_{k=1}^{m} \| f_m^i - f_m^j \|)$, $i, j = 1, \ldots, |Y_{known}|_c$, f is the objection function, m is the number of objectives and $\bar{d}$ is the mean of all d_i. The higher the value of the ONVG, the better for understanding Pareto front details. A good solution set should have a value close to 0 for the SP metric. Table 1 shows ONVG and SP under VMPACS and MGGA. The column "Corr." indicates the correlation coefficients for the CPU

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

Table 1
ONVG and SP performance comparison of VMPACS and MGGA.

Reference value	Corr.	Algorithm	ONVG	SP
$\overline{R_p} = \overline{R_m} = 25\%$	−0.754	MGGA	15.13	0.62
		VMPACS	20.34	0.23
	−0.348	MGGA	16.58	0.49
		VMPACS	22.64	0.17
	−0.072	MGGA	11.50	0.40
		VMPACS	23.13	0.14
	0.371	MGGA	14.27	0.29
		VMPACS	18.49	0.12
	0.755	MGGA	14.89	0.17
		VMPACS	24.31	0.08
$\overline{R_p} = \overline{R_m} = 45\%$	−0.755	MGGA	16.38	0.20
		VMPACS	20.57	0.11
	−0.374	MGGA	19.46	0.23
		VMPACS	27.80	0.12
	−0.052	MGGA	14.54	0.29
		VMPACS	25.16	0.14
	0.398	MGGA	12.04	0.16
		VMPACS	23.76	0.07
	0.751	MGGA	10.42	0.11
		VMPACS	24.36	0.05

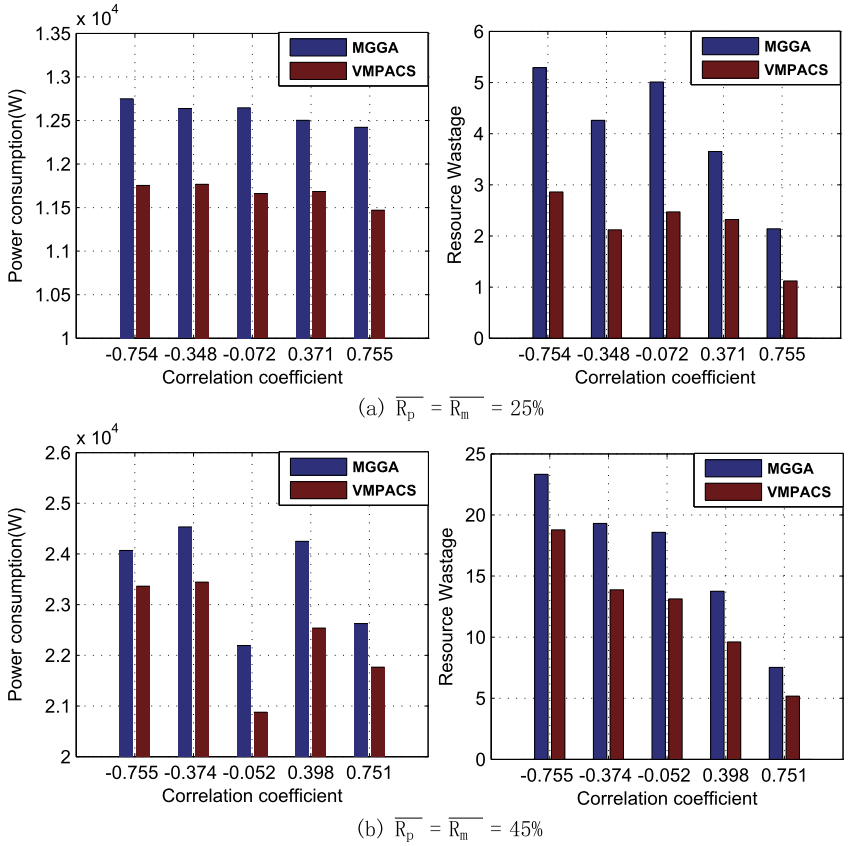

Fig. 3. Power consumption and resource wastage of VMPACS and MGGA in the case of $\overline{R_p} = \overline{R_m} = 25\%$ (a) and $\overline{R_p} = \overline{R_m} = 45\%$ (b). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and memory utilizations. Fig. 3 compares the total resource wastage and power consumption for each of the algorithms under consideration. From the results, we can clearly see that the VMPACS algorithm outperforms MGGA. The reason is that VM placement under VMPACS combines the partial solution information under construction and the feed information of

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

Table 2
Power consumption and resource wastage of VMPACS and two single-objective algorithms.

Reference value	Corr.	Algorithm	Power (W)	Wastage
$\overline{R_p} = \overline{R_m} = 25\%$	−0.754	FFD	14848.58	30.54
		SACO	13268.31	11.46
		VMPACS	11754.38	2.86
	−0.348	FFD	13937.74	21.57
		SACO	12906.80	10.52
		VMPACS	11766.94	2.12
	−0.072	FFD	13346.34	16.79
		SACO	12538.19	8.92
		VMPACS	11661.55	2.47
	0.371	FFD	12901.98	11.31
		SACO	12315.27	6.42
		VMPACS	11684.99	2.32
	0.755	FFD	12021.44	5.22
		SACO	11829.85	3.61
		VMPACS	11470.24	1.12
$\overline{R_p} = \overline{R_m} = 45\%$	−0.755	FFD	24968.68	29.56
		SACO	24549.19	25.68
		VMPACS	23364.88	18.78
	−0.374	FFD	25533.82	28.25
		SACO	24395.42	15.48
		VMPACS	23444.02	13.88
	−0.052	FFD	25092.92	25.57
		SACO	22571.14	20.85
		VMPACS	20876.32	13.13
	0.398	FFD	23850.40	19.93
		SACO	23048.95	15.48
		VMPACS	22538.20	9.61
	0.751	FFD	22625.49	11.13
		SACO	22158.37	9.52
		VMPACS	21766.89	5.18

the reserved time of a non-dominated solution in the external set and simultaneously incorporates continuous updating of pheromone, therefore it can find more appropriate VM placement and achieve better performance.

5.2. Comparison of VMPACS with two single-objective approaches

In this set of experiments, we compared the proposed approach with two single-objective algorithms, an FFD algorithm and a SACO approach. SACO is a modified MMAS algorithm for VM placement. FFD considers VMs in a decreasing order of utilization of a certain resource and places each VM into the first host that has enough resource remaining. Table 2 compares the total resource wastage and power consumption for each of the algorithms under consideration. From the table, we can see that: (1) FFD yields the highest power consumption and resource wastage because it tends to use a larger number of servers compared with other algorithms. (2) VMPACS produces the lowest power consumption and resource wastage because it is able to search the solution space more efficiently and globally so that it can find the solutions with a smaller number of used servers and high resource utilization compared with FFD and SACO. (3) The power consumption and resource wastage of SACO are between those of the other two. The reason is that SACO can find solutions with a smaller number of used servers compared with FFD and a larger number of used servers compared with VMPACS.

5.3. Scalability of VMPACS

In this subsection, we provide experimental results about whether the proposed algorithm is scalable to larger data centers and more VM requests. In the experiment, we fix $P = 0.5$ and change the number of VM requests from 100 to 2000, and set both $\overline{R_p}$ and $\overline{R_m}$ to 25% and then 45%. Fig. 4 shows the result of the experiment conducted for instances with $\overline{R_p} = \overline{R_m} = 25\%$ and $\overline{R_p} = \overline{R_m} = 45\%$. We can see from the graph that in the case of $\overline{R_p} = \overline{R_m} = 25\%$ it takes 31 seconds to calculate a new placement of 1000 VMs but the running time increases to 114 seconds when we increase the number of VMs to 2000. However, in the case of $\overline{R_p} = \overline{R_m} = 45\%$, we can see that it increases faster than for $\overline{R_p} = \overline{R_m} = 25\%$. It takes 36 and 133 seconds to calculate a new placement for 1000 and 2000 VMs with $\overline{R_p} = \overline{R_m} = 45\%$. The reason for two different results with the same number of VMs is that the number of servers used to contain VMs differs in each case: on average 90/25 VMs per server in the case of $\overline{R_p} = \overline{R_m} = 25\%$ and two VMs in the case of $\overline{R_p} = \overline{R_m} = 45\%$. This experiment

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

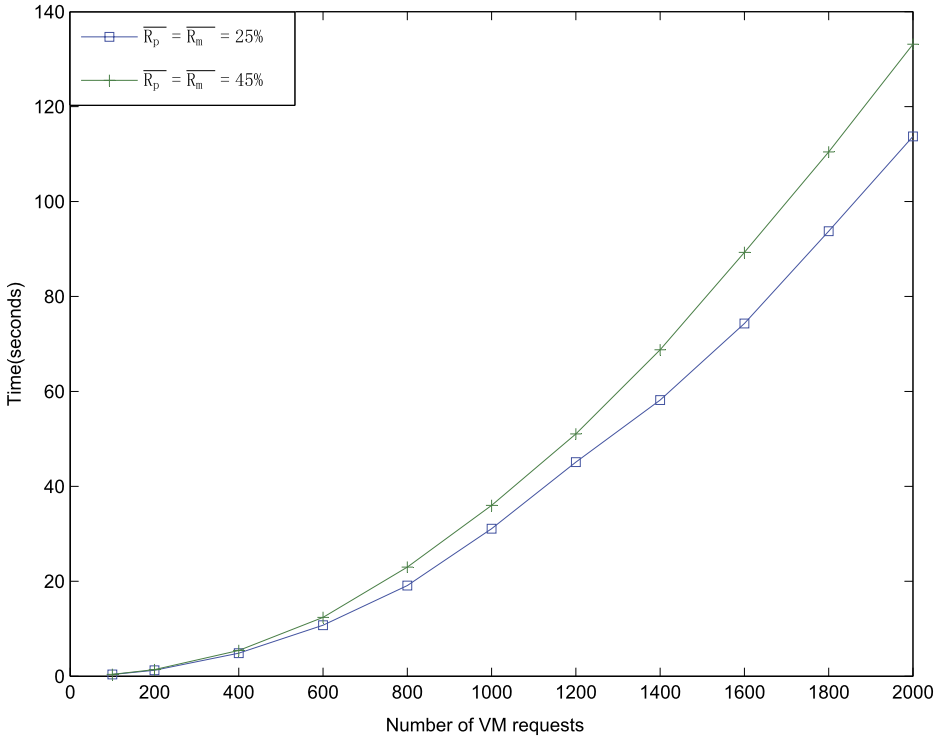

Fig. 4. VMPACS algorithm running time relative to number of VM requests.

shows that our algorithm takes less than 3 minutes to solve a difficult placement problem with up to 2000 VMs. Therefore, we can say that our algorithm is suitable for large data centers.

6. Conclusion

With the increasing prevalence of large scale cloud computing environments, how to efficiently place VMs into available computing servers has become an essential research problem. In this paper, we propose a multi-objective ant colony system algorithm for the virtual machine placement problem. The goal is to efficiently obtain a set of non-dominated solutions that simultaneously minimizes total resource wastage and power consumption. The proposed algorithm is tested with some instances from the literature. Its solution performance is compared to that of an existing multi-objective grouping genetic algorithm. The results demonstrate that our algorithm is competitive. We also compare our algorithm with two single-objective approaches. The comparison shows that VMPACS is superior to those algorithms. Finally the scalability of the proposed algorithm is verified by means of several experiments.

Acknowledgments

This work is supported by the Program for PCSIRT and NCET of MOE, National Natural Science Foundation of China (Grant No. 61073151), the 863 Program (Grant Nos. 2011AA01A202, 2012AA010905), 973 Program (Grant No. 2012CB723401), the International Cooperation Program of China (Grant No. 2011DFA10850), International Cooperation Program of Shanghai (No. 11530700500), the Ministry of Education and Intel joint research foundation (Grant No. MOE-INTEL-11-05), the “Dawn” Program of Shanghai Education Commission (Grant No. 09DJW602002).

References

[1] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges, J. Internet Services Appl. 1 (1) (2010) 7–18.
[2] M. Randles, D. Lamb, E. Odat, A. Taleb-Bendiab, Distributed redundancy and robustness in complex systems, J. Comput. System Sci. 77 (2) (2011) 293–304.
[3] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, G. Jiang, Power and performance management of virtualized computing environments via lookahead control, Cluster Computing 12 (1) (2009) 1–15.
[4] M. Cardosa, M. Korupolu, A. Singh, Shares and utilities based power consolidation in virtualized server environments, in: Proceedings of IFIP/IEEE Integrated Network Management (IM'09), 2009, pp. 327–334.
[5] L. Grit, D. Irwin, A. Yumerefendi, J. Chase, Virtual machine hosting for networked clusters: Building the foundations for autonomic orchestration, in: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing, 2006, p. 7.

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004

[6] W. Vogels, Beyond server consolidation, ACM Queue 6 (1) (2008) 20–26.
[7] K. Li, H. Shen, Proxy placement problem for coordinated en-route transcoding proxy caching, Comput. Systems Sci. Engrg. 19 (6) (2004) 327–335.
[8] K. Li, H. Shen, Optimal proxy placement for coordinated en-route transcoding proxy caching, IEICE Trans. Inform. Syst. 87 (12) (2004) 2689–2696.
[9] K. Li, H. Shen, Optimal placement of web proxies for tree networks, in: Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service, 2004, pp. 479–486.
[10] K. Li, H. Shen, F. Chin, S. Zheng, Optimal methods for coordinated enroute web caching for tree networks, ACM Trans. Internet Technol. (TOIT) 5 (3) (2005) 480–507.
[11] K. Li, H. Shen, F. Chin, W. Zhang, Multimedia object placement for transparent data replication, IEEE Trans. Parallel Distrib. Syst. 18 (2) (2007) 212–224.
[12] S. Chaisiri, B. Lee, D. Niyato, Optimal virtual machine placement across multiple cloud providers, in: Proceedings of the IEEE Asia-Pacific Services Computing Conference, 2009, pp. 103–110.
[13] M. Bichler, T. Setzer, B. Speitkamp, Capacity planning for virtualized servers, in: Workshop on Information Technologies and Systems (WITS), 2006.
[14] B. Speitkamp, M. Bichler, A mathematical programming approach for server consolidation problems in virtualized data centers, IEEE Trans. Services Comput. (2010) 266–278.
[15] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, L. Yuan, Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers, in: Proceedings of the IEEE International Conference on Services Computing, 2010, pp. 514–521.
[16] J. Xu, J. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in: Proceedings of the IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, 2010, pp. 179–188.
[17] H. Van, F. Tran, J. Menaud, Performance and power management for cloud infrastructures, in: Proceedings of the IEEE 3rd International Conference on Cloud Computing, 2010, pp. 329–336.
[18] F. Hermenier, X. Lorca, J. Menaud, G. Muller, J. Lawall, Entropy: a consolidation manager for clusters, in: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2009, pp. 41–50.
[19] J. Békési, G. Galambos, H. Kellerer, A 5/4 linear time bin packing algorithm, J. Comput. System Sci. 60 (1) (2000) 145–160.
[20] N. Bobroff, A. Kochut, K. Beaty, Dynamic placement of virtual machines for managing sla violations, in: Proceedings of the 10th IEEE Symposium on Integrated Management (IM), 2007, pp. 119–128.
[21] A. Verma, P. Ahuja, A. Neogi, pMapper: power and migration cost aware application placement in virtualized systems, in: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, 2008, pp. 243–264.
[22] S. Srikantaiah, A. Kansal, F. Zhao, Energy aware consolidation for cloud computing, in: Proceedings of HotPower'08 Workshop on Power Aware Computing and Systems, 2008.
[23] B. Li, J. Li, J. Huai, T. Wo, Q. Li, L. Zhong, Enacloud: an energy-saving application live placement approach for cloud computing environments, in: Proceedings of the IEEE International Conference on Cloud Computing, 2009, pp. 17–24.
[24] E. Feller, L. Rilling, C. Morin, Energy-aware ant colony based workload placement in clouds, in: Proceedings of the IEEE/ACM International Conference on Grid Computing (GRID), 2011, pp. 26–33.
[25] G. Khanna, K. Beaty, G. Kar, A. Kochut, Application performance management in virtualized server environments, in: Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS), 2006, pp. 373–381.
[26] C. Lin, G. Wu, F. Xia, M. Li, L. Yao, Z. Pei, Energy efficient ant colony algorithms for data aggregation in wireless sensor networks, J. Comput. System Sci. 78 (6) (2012) 1686–1702.
[27] M. Dorigo, L. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, IEEE Trans. Evol. Comput. 1 (1) (1997) 53–66.
[28] S. Shyu, B. Lin, P. Yin, Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time, Comput. Indust. Engrg. 47 (2–3) (2004) 181–193.
[29] V. Maniezzo, A. Colorni, The ant system applied to the quadratic assignment problem, IEEE Trans. Knowl. Data Engrg. 11 (5) (1999) 769–778.
[30] K. Socha, C. Blum, An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training, Neural Comput. Appl. 16 (3) (2007) 235–247.
[31] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybernet. Part B: Cybernetics 26 (1) (1996) 29–41.
[32] T. Stutzle, H. Hoos, Max–min ant system and local search for the traveling salesman problem, in: Proceedings of the IEEE International Conference on Evolutionary Computation, 1997, pp. 309–314.
[33] C. Garcia-Martinez, O. Cordón, F. Herrera, A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp, European J. Oper. Res. 180 (1) (2007) 116–148.
[34] M. Ikeda, L. Barolli, A. Koyama, A. Durresi, G. De Marco, J. Iwashige, Performance evaluation of an intelligent cac and routing framework for multimedia applications in broadband networks, J. Comput. System Sci. 72 (7) (2006) 1183–1200.
[35] J. Branke, K. Deb, K. Miettinen, Multiobjective Optimization: Interactive and Evolutionary Approaches, Springer-Verlag, New York, 2008.
[36] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, Wiley, 2001.
[37] K. Deb, Multi-objective genetic algorithms: Problem difficulties and construction of test problems, Evol. Comput. 7 (3) (1999) 205–230.
[38] X. Fan, W. Weber, L. Barroso, Power provisioning for a warehouse-sized computer, in: Proceedings of the 34th Annual International Symposium on Computer Architecture, 2007, pp. 13–23.
[39] V. Maniezzo, Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem, INFORMS J. Comput. 11 (4) (1999) 358–369.
[40] Y. Ajiro, A. Tanaka, Improving packing algorithms for server consolidation, in: Proceedings of the International Conference for the Computer Measurement Group (CMG), Computer Measurement Group, 2007.
[41] D. Van Veldhuizen, Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations, PhD thesis, Grad. School of Eng. of the Air Force Institute of Technology, Air University, 1999.
[42] J. Schott, Fault tolerant design using single and multicriteria genetic algorithm optimization, Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 1995.

Please cite this article in press as: Y. Gao et al., A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. System Sci. (2013), http://dx.doi.org/10.1016/j.jcss.2013.02.004