Multi-objective dynamic virtual machine consolidation in the cloud using ant colony system

Adnan Ashraf and Ivan Porres
Faculty of Natural Sciences and Technology, Åbo Akademi University, Finland

ABSTRACT
In this paper, we present a novel multi-objective ant colony system algorithm for virtual machine (VM) consolidation in cloud data centres. The proposed algorithm builds VM migration plans, which are then used to minimise over-provisioning of physical machines (PMs) by consolidating VMs on under-utilised PMs. It optimises two objectives that are ordered by their importance. The first and foremost objective in the proposed algorithm is to maximise the number of released PMs. Moreover, since VM migration is a resource-intensive operation, it also tries to minimise the number of VM migrations. The proposed algorithm is empirically evaluated in a series of experiments. The experimental results show that the proposed algorithm provides an efficient solution for VM consolidation in cloud data centres. Moreover, it outperforms two existing ant colony optimization-based VM consolidation algorithms in terms of number of released PMs and number of VM migrations.

MOACS advances the state of the art on ACO-based VM consolidation by implementing a multi-objective, multi-colony ACS algorithm. It extends our previous single-objective, single-colony ACO algorithm for VM consolidation and similar works by other researchers that implement single-objective, single-colony ACO algorithms. The proposed multi-objective, multi-colony approach eliminates the need for an aggregate objective function (AOF) and allows to combine the optimisation objectives in an appropriate manner.

ARTICLE HISTORY
Received 19 October 2016
Accepted 1 January 2017

KEYWORDS
Virtual machines; consolidation; metaheuristic; ant colony system; cloud computing

1. Introduction
Cloud data centres comprise thousands of physical machines (PMs) and networking devices. These resources and their cooling infrastructure incur huge energy footprints. High energy consumption not only translates into a high operating cost, but also leads to huge carbon emissions [1,2]. Therefore,
energy footprint of data centres is a major concern for cloud providers. The high energy consumption of data centres can partly be attributed to the large-scale installations of computing and cooling infrastructures, but more importantly it is due to the inefficient use of the computing resources [3].

Hardware virtualisation technologies allow to share a PM among multiple, performance-isolated virtual machines (VMs) to improve resource utilisation. Further improvement in resource utilisation and reduction in energy consumption can be achieved by consolidating VMs on PMs and switching idle PMs off or to a low-power mode. VM consolidation has emerged as one of the most effective and promising techniques to reduce energy footprint of cloud data centres [3,4]. A VM consolidation approach uses live VM migration to consolidate VMs on a reduced set of PMs. Thereby, allowing some of the under-utilised PMs to be turned-off or switched to a low-power mode to conserve energy.

There is currently an increasing amount of interest on developing and evaluating efficient VM consolidation approaches for cloud data centres. Over the past few years, researchers have used a multitude of ways to develop novel VM consolidation approaches [3,5–15]. Some of these approaches have also been reported in recent literature reviews [16,17].

The main output of a VM consolidation algorithm is a VM migration plan, which is implemented by first migrating VMs from one PM to another and then shutting down or allocating new work to the idle PMs. The quality of a VM consolidation algorithm can be evaluated according to different criteria, including the number of released PMs (to be maximised), the number of VM migrations from one PM to another (to be minimised), and the algorithm execution time (to be minimised). Moreover, since a migration plan with a higher number of released PMs is always preferred to a migration plan with a lower number of VM migrations, maximising the number of released PMs takes precedence over minimising the number of VM migrations.

The VM consolidation problem is an NP-hard combinatorial optimisation problem [18]. Therefore, it requires advanced strategies in order to be viable in practice. One way to address this problem is to use the exact optimisation techniques, such as mixed-integer linear programming, which find optimal solutions with exponential runtime complexity. However, such techniques are mostly impractical for realistic, large-sized problem instances. Moreover, the currently available commercial, exact optimisation tools, such as IBM ILOG CPLEX, do not provide support for multi-objective optimization problems. A widely used alternative approach to solve difficult combinatorial optimisation problems involves the use of metaheuristics [19]. Metaheuristics are high-level procedures that efficiently explore the search-space of available solutions with the aim to find near-optimal solutions with a polynomial time complexity [20].

Some of the recent works [4,18,21,22] on VM consolidation use a highly adaptive metaheuristic called ant colony optimization (ACO) [23]. The existing ACO-based VM consolidation approaches [4,18,21,22] tend to use single-objective, single-colony algorithms with an aggregate objective function (AOF) that tries to combine multiple objectives. The benefit of the AOF approach is that it reduces complexity and may improve the runtime of the algorithm by limiting the search to a subspace of the feasible solutions. However, the main drawback is that a correct combination of the objectives requires certain weights to be assigned to each objective, which often requires an in-depth knowledge of the problem domain [17]. Therefore, the assignment of the weights is essentially subjective [24]. Moreover, an AOF may not combine the optimisation objectives in an appropriate manner. For instance, the AOFs in the existing ACO-based VM consolidation approaches [4,18,21,22] do not allow to order the objectives by their importance.

In this paper, we present a novel multi-objective ACO-based VM consolidation algorithm for cloud data centres that, according to our evaluation, outperforms the existing ACO-based VM consolidation algorithms [4,18,21,22]. It uses ant colony system (ACS) [25], which is currently one of the best performing ACO algorithms. The proposed multi-objective ACS algorithm for VM consolidation is called MOACS and it optimises two objectives. The first and foremost objective in MOACS is to maximise the number of released PMs. Moreover, since VM migration is a resource-intensive operation, MOACS also tries to minimise the number of VM migrations. We adapt and use a multi-objective, multi-colony ACO algorithm by Gambardella et al. [26], which orders the objectives by their importance. The proposed
algorithm is not dependent on a particular deployment architecture or system topology. Therefore, it can be implemented in centralised as well as decentralised deployment architectures and system topologies, as the one proposed in [21].

1.1. Contributions

The main task of the proposed MOACS algorithm is to find a VM migration plan that maximises the number of released PMs while minimising the number of VM migrations. In the rest of this section, we summarise the main contributions of this paper.

1.1.1. Improved multi-objective, multi-colony optimization

MOACS advances the state of the art on ACO-based VM consolidation by implementing a multi-objective, multi-colony ACS algorithm. It extends our previous single-objective, single-colony ACO algorithm for VM consolidation [4,18] and similar works by other researchers [21,22] that implement single-objective, single-colony ACO algorithms. The proposed multi-objective, multi-colony approach eliminates the need for an AOF and allows to combine the optimisation objectives in an appropriate manner.

1.1.2. Improved reduction of search space

Since VM consolidation is an NP-hard problem, it requires fast and scalable algorithms. In order to improve the runtime performance of the proposed algorithm, we present three simple constraints in Section 3. These constraints determine which PMs and VM migrations can be excluded from the consolidation process without compromising on the quality of the solutions. We refine two constraints from our previous work [4,18] and complement them with a new constraint concerning neighbourhoods of PMs.

1.1.3. Improved experimental results

We have implemented the proposed MOACS algorithm in Java and have compared it with two existing ACO-based VM consolidation algorithms. The first one is the single-objective, single-colony max–min ant system VM consolidation algorithm by Feller et al. [21], that we name Feller-ACO for evaluation. We selected the Feller-ACO algorithm for comparison due to its excellent overall performance in many aspects as shown by its authors [21]. The second one is our previously published single-objective, single-colony ACS VM consolidation algorithm [4], that we refer to as ACS for evaluation. The ACS algorithm was selected as baseline for our work since it outperformed many other existing algorithms at the time of publication, as shown in [4]. Our results show that the proposed MOACS algorithm outperforms Feller-ACO in all attributes: number of released PMs, number of VM migrations, packing efficiency and algorithm execution time. Similarly, it outperforms ACS [4] in all attributes except in execution time.

We proceed as follows. Section 2 provides background and discusses important related works. The proposed MOACS algorithm is described in detail in Section 3 while its experimental evaluation is presented in Section 4. Finally, we present our conclusions in Section 5.

2. Background and related work

ACO metaheuristic is inspired from the foraging behaviour of real ant colonies [23]. While transporting food from the food source to their nest, ants deposit and follow trails of a chemical substance on their paths called pheromone. It allows them to indirectly communicate with one another to find better paths between their nest and the food source. Empirical results from previous research on ACO has shown that the simple pheromone trail following behaviour of ants can give rise to the emergence of the shortest paths.
Each ant finds a complete path or solution, but high quality solutions emerge from the indirect communication and global cooperation among multiple concurrent ants [4]. Ants must also avoid stagnation, which is a premature convergence to a suboptimal solution. It is achieved using pheromone evaporation and stochastic state transitions. There are a number of ACO algorithms, such as ant system (AS), ACS, and max–min ant system (MMAS) [23]. ACS [25] is currently one of the best performing ant algorithms. Therefore, in this paper, we apply ACS to the VM consolidation problem.

The existing ACO-based resource allocation, VM placement and VM consolidation approaches include [4,18,21,22,27–30]. Yin and Wang [28] applied ACO to the nonlinear resource allocation problem, which seeks to find an optimal allocation of a limited amount of resources to a number of tasks. Chaharsooghi and Kermani [29] proposed a modified version of ACO for the multi-objective resource allocation problem. Feller et al. [21] applied the single-objective, single-colony MMAS algorithm to the VM consolidation problem in the context of cloud computing. Ferdaus et al. [22] integrated ACS with a vector algebra-based server resource utilisation capturing technique [31]. In our previous work [27], we applied the original single-objective, single-colony ACS algorithm [25] to the web application consolidation problem. Similarly, in our previous work [4,18], we used the original ACS algorithm [25] for energy-aware VM consolidation in cloud data centres.

Gao et al. [30] used a multi-objective ACS algorithm with two equally important objectives: minimise energy consumption and minimise resource wastage. In their approach, both energy consumption and resource wastage are derived from the number of PMs used for the placement of VMs. Their approach only provides an initial placement of VMs on PMs. It does not migrate VMs from one PM to another. Therefore, it cannot be used to consolidate VMs or to minimise the number of VM migrations. The output of Gao et al.’s algorithm is a Pareto set of solutions, from which a solution is randomly selected. A drawback of this approach is that the objectives cannot be ordered by their importance. Moreover, the randomly selected solution may not be the most desired solution. To the best of our knowledge, none of the existing ACO-based VM consolidation approaches use a multi-objective ACS algorithm that orders the objectives by their importance.

The main difference between the existing ACO-based VM consolidation approaches [4,18,21,22] and our proposed MOACS algorithm is that the existing approaches tend to use a single-objective, single-colony ACO algorithm with an AOF that tries to combine multiple objectives, whereas MOACS uses a multi-objective ACS algorithm with two independent ant colonies. The first colony maximises the number of released PMs, while the second colony minimises the number of VM migrations. We adapt and use a multi-objective, multi-colony ACS algorithm by Gambardella et al. [26], which was originally proposed for the vehicle routing problem with time windows.

### 3. Multi-objective ACS algorithm for VM consolidation

In this section, we present our proposed multi-objective ACS-based VM consolidation algorithm (MOACS). As its first objective, MOACS maximises the number of released PMs $|P_R|$. Moreover, its second objective is to minimise the number of VM migrations $nM$. Since a global best migration plan $\Psi^+$ with a higher number of released PMs $|P_R|$ is always preferred to a $\Psi^+$ with a lower $|P_R|$ even if the number of VM migrations $nM$ is higher in the former $\Psi^+$, maximising $|P_R|$ takes precedence over minimising $nM$. For the sake of clarity, important concepts and notations used in the following sections are tabulated in Table 1.

Figure 1 illustrates MOACS architecture. MOACS coordinates the operations of two ACS-based ant colonies, which simultaneously optimise their respective objectives. The first objective concerning the number of released PMs $|P_R|$ is optimised by the first colony called $\text{ACS}_{|P_R|}$. Similarly, the second colony called $\text{ACS}_{nM}$ optimises the second objective concerning the number of VM migrations $nM$. Both colonies work independently and use independent pheromone and heuristic matrices. However, they collaborate on the global best migration plan $\Psi^+$, which is maintained by MOACS.

In the VM consolidation problem, each PM $p \in P$ hosts multiple VMs $v \in V$. Each PM that hosts at least one VM is a potential source PM. Both the source PM and the VM are characterised by their resource
Table 1. Summary of concepts and their notations.

| Symbol | Description |
|--------|-------------|
| $M$    | set of migration plans |
| $P$    | set of PMs |
| $P_R$  | set of PMs that are released when a migration plan $\Psi$ is enforced |
| $T$    | set of tuples |
| $T_k$  | set of tuples not yet traversed by ant $k$ |
| $V$    | set of VMs |
| $V_p$  | set of VMs running on a PM $p$ |
| $C_{p_{de}}$ | total capacity vector of the destination PM $p_{de}$ |
| $N$    | a neighbourhood of PMs |
| $p_{so}$ | source PM in a tuple |
| $q$    | a uniformly distributed random variable |
| $S$    | a random variable selected according to (7) |
| $S_{ck}$ | thus far best score of ant $k$ |
| $U_{p_{de}}$ | used capacity vector of the destination PM $p_{de}$ |
| $U_{p_{so}}$ | used capacity vector of the source PM $p_{so}$ |
| $U_v$  | used capacity vector of the VM $v$ |
| $\eta$ | heuristic value |
| $\tau$ | amount of pheromone |
| $\tau_0$ | initial pheromone level |
| $\Psi$ | a migration plan |
| $\Psi^+$ | the global best migration plan |
| $\Psi_{nM}$ | thus far best migration plan from ACS$_{nM}$ |
| $\Psi_{PR}$ | thus far best migration plan from ACS$_{|PR|}$ |
| $\Psi_p$ | ant-specific migration plan of ant $k$ |
| $\Delta_{nM}$ | ant-specific temporary migration plan of ant $k$ |
| $\Delta_{PR}$ | additional pheromone amount given to the tuples in $\Psi_{nM}^+$ |
| $\Delta_{PR}$ | additional pheromone amount given to the tuples in $\Psi_{PR}^+$ |
| $\alpha$ | pheromone decay parameter in the global updating rule |
| $\beta$ | parameter to determine the relative importance of $\eta$ |
| $\rho$ | pheromone decay parameter in the local updating rule |
| $\rho_0$ | parameter to determine relative importance of exploitation |
| $nA$  | number of ants that concurrently build their migration plans |
| $nI$  | number of ant generations |
| $nM$  | number of VM migrations |
| $f(\Psi)$ | objective function in (3) concerning number of released PMs |
| $g_i(\Psi)$ | objective function in (13) concerning number of VM migrations |

Figure 1. MOACS architecture.
utilisations, such as CPU utilisation and memory utilisation. MOACS uses a notion of neighbourhoods of PMs. Neighbourhoods are mutually exclusive subsets of $P$. A neighbourhood is an abstract entity that represents a set of closely located PMs in a cloud data centre. For example, the PMs in a data centre rack may constitute a neighbourhood of PMs. A VM can be migrated to any other PM located in any neighbourhood $N$. Therefore, every other PM within as well as outside the neighbourhood of the source PM is a potential destination PM, which is also characterised by its resource utilisations. Thus, MOACS makes a set of tuples $T$, where each tuple $t \in T$ consists of three elements: source PM $p_{so}$, VM $v$, and destination PM $p_{de}$

$$t := (p_{so}, v, p_{de})$$

(1)

The computation time of the proposed VM consolidation algorithm is primarily based on the number of tuples $|T|$. Therefore, in order to reduce the computation time, MOACS applies three constraints, which result in a reduced set of tuples by removing some least important and unwanted tuples. The first constraint ensures that only under-utilised PMs are used as the source PMs. Similarly, the second constraint allows only under-utilised PMs to be considered as the destination PMs. In other words, migrations from and to well-utilised PMs are excluded. The rationale is that a well-utilised PM should not become part of the consolidation process because migrating to a well-utilised PM may result in its overloading. Similarly, migrating from a well-utilised PM is less likely to result in the termination of the source PM and thus it would not reduce the total number of required PMs. The third constraint further restricts the size of the set of tuples $|T|$ by preventing inter-neighbourhood migrations. Therefore, as a general rule, a VM can only be migrated to another PM within the neighbourhood of its source PM. The only exception to this rule is when a neighbourhood has only one PM in it. In this case, the VMs from the lone PM can be migrated to any other PM in any neighbourhood. By applying these three simple constraints in a series of preliminary experiments, we observed that the computation time of the algorithm was significantly reduced without compromising on the quality of the solutions.

The space complexity of the proposed algorithm is $O(|T|)$, where $|T|$ is the number of tuples. Moreover, the worst-case space complexity corresponds to a VM consolidation scenario that does not involve any well-utilised PMs. In such a scenario, each PM is considered as a source as well as a destination PM. The maximum number of tuples in the worst-case is computed as

$$\text{maximum } |T| := |P| \cdot |V| \cdot (|N| - 1)$$

(2)

where $|P|$ is the number of PMs, $|V|$ is the number of VMs, and $|N|$ is the neighbourhood size. Since real VM consolidation scenarios usually involve one or more well-utilised PMs and the proposed algorithm excludes migrations from and to well-utilised PMs, the actual number of tuples $|T|$ in a real scenario is often smaller than that of the worst-case scenario.

The pseudocode of the proposed MOACS algorithm is presented in Algorithm 1. Initially, the global best migration plan $\Psi^+$ is not known. Therefore, $\Psi^+$ is empty (line 2). The main loop in line 1 iterates until a stopping criterion is met. For instance, when all remaining PMs are well-utilised or when no further improvements are achieved in a given number of consecutive iterations [26]. In each iteration of the main loop, the two ACS-based colonies $\text{ACS}_{PR}$ and $\text{ACS}_{nM}$ try to find the global best migration plan $\Psi^+$ according to their respective objectives. $\text{ACS}_{PR}$ tries to find a migration plan with a higher number of released PMs $|P_R|$ (line 3). Similarly, $\text{ACS}_{nM}$ tries to find a migration plan with fewer VM migrations (line 4). The global best migration plan $\Psi^+$ is updated every time an improved migration plan is found. If $\text{ACS}_{PR}$ finds a migration plan with a higher number of released PMs (line 5), $\Psi^+$ is updated according to the thus far best migration plan from $\text{ACS}_{PR}$ denoted as $\Psi^+_{PR}$ (line 6). Likewise, when $\text{ACS}_{nM}$ finds a migration plan with fewer VM migrations, but with at least as many released PMs $P_R$ as in $\Psi^+$ (lines 8–9), $\Psi^+$ is updated according to the thus far best migration plan from $\text{ACS}_{nM}$ denoted as $\Psi^+_{nM}$ (line 10). Finally, at the end of each iteration of the main loop, VMs are consolidated according to the global best migration plan $\Psi^+$ and the released PMs are terminated (line 13).
3.1. ACS-based colony to maximise the number of released PMs

The ACS\(|PR|\) colony optimises the first objective concerning the number of released PMs \(|P_R|\). Therefore, the objective function for the ACS\(|PR|\) algorithm is

\[
\text{maximize } f(\Psi) := |P_R| \tag{3}
\]

where \(\Psi\) is the migration plan and \(P_R\) is the set of PMs that will be released when \(\Psi\) is enforced. Since the primary objective of VM consolidation is to minimise the number of active PMs, the objective function is defined in terms of number of released PMs \(|P_R|\). Moreover, when a migration plan is enforced, we apply a constraint which reduces the number of VM migrations \(nM\) by restricting migrations to only those PMs that are not included in the set of released PMs \(P_R\), that is

\[
\forall p_{de} \in P \mid p_{de} \notin P_R \tag{4}
\]

In our approach, a PM can only be considered as released when all VMs migrate from it. Therefore, the set of released PMs \(P_R\) is defined as

\[
P_R := \{\forall p \in P | V_p = \emptyset\} \tag{5}
\]

where \(V_p\) is the set of VMs running on a PM \(p\). Thus, a PM can only be included in the set of released PMs \(P_R\) when it no longer hosts any VMs.

Since there is no notion of path in the VM consolidation problem, ants deposit pheromone on the tuples defined in (1). Each of the \(nA\) ants uses a stochastic state transition rule to choose the next tuple to traverse. The state transition rule in ACS\(|PR|\) is called pseudo-random-proportional-rule [25]. According to this rule, an ant \(k\) chooses a tuple \(s\) to traverse next by applying

\[
s := \begin{cases} 
\arg \max_{u \in T_k} ([\tau_u] \cdot [\eta_u]^{\beta}), & \text{if } q \leq q_0 \\
S, & \text{otherwise}
\end{cases} \tag{6}
\]

where \(\tau\) denotes the amount of pheromone and \(\eta\) represents the heuristic value associated with a particular tuple. \(\beta\) is a parameter to determine the relative importance of the heuristic value with respect to the pheromone value. The expression \(\arg \max\) returns the tuple for which \([\tau] \cdot [\eta]^{\beta}\) attains its maximum value. \(T_k \subset T\) is the set of tuples that remain to be traversed by ant \(k\). \(q \in [0, 1]\) is a uniformly distributed random variable and \(q_0 \in [0, 1]\) is a parameter. \(S\) is a random variable selected.

**Algorithm 1 Multi-objective ACS algorithm for VM consolidation**

1: while until a stopping criterion is met do
2: \(\Psi^+ := \emptyset\)
3: \(\Psi^+_{PR} := \text{ACS}_{|PR|}\)
4: \(\Psi^+_{nM} := \text{ACS}_{nM}\)
5: if \(\Psi^+ = \emptyset \lor f(\Psi^+_{PR}) > f(\Psi^+_{nM})\) then
6: \(\Psi^+ := \Psi^+_{PR}\)
7: end if
8: if \(f(\Psi^+_{nM}) \geq f(\Psi^+_{PR})\) then
9: if \(g(\Psi^+_{nM}) > g(\Psi^+_{PR})\) then
10: \(\Psi^+ := \Psi^+_{nM}\)
11: end if
12: end if
13: consolidate VMs according to \(\Psi^+\) and terminate released PMs
14: end while
according to the probability distribution given in (7), where the probability \( \text{prob}_s \) of an ant \( k \) to choose tuple \( s \) to traverse next is defined as

\[
\text{prob}_s := \begin{cases} \frac{[\tau_s][\eta_s]^\beta}{\sum_{u \in T_k} [\tau_u][\eta_u]^\beta}, & \text{if } s \in T_k \\ 0, & \text{otherwise} \end{cases}
\]  

The heuristic value \( \eta_s \) of a tuple \( s \) is defined as

\[
\eta_s := \begin{cases} \frac{U_{pde} + U_v}{C_{pde}}, & \text{if } U_{pde} + U_v \leq C_{pde} \\ 0, & \text{otherwise} \end{cases}
\]  

where \( C_{pde} \) is the total capacity vector of the destination PM \( p_{de} \), \( U_{pde} \) is the used capacity vector of \( p_{de} \), and likewise \( U_v \) is the used capacity vector of the VM \( v \) in tuple \( s \). The heuristic value \( \eta \) is based on the ratio of \( (U_{pde} + U_v) \) to \( C_{pde} \). Therefore, destination PMs with the minimum unused capacity receive the highest amount of heuristic value. Thus, the heuristic value favours VM migrations that would result in a reduced under-utilisation of PMs. Moreover, the constraint \( U_{pde} + U_v \leq C_{pde} \) prevents VM migrations that would result in the overloading of the destination PM \( p_{de} \). In the proposed algorithm, we assumed two resource dimensions, which represent CPU utilisation and memory utilisation. However, if necessary, it is possible to add more dimensions in the total and used capacity vectors.

The stochastic state transition rule in (6) and (7) prefers tuples with a higher pheromone concentration and which result in a higher number of released PMs. The first case in (6) where \( q \leq q_0 \) is called exploitation [25]. It chooses the best tuple that attains the maximum value of \( [\tau_s][\eta_s]^\beta \). The second case, called biased exploration, selects a tuple according to (7). The exploitation helps the ants to quickly converge to a high quality solution, while at the same time, the biased exploration helps them to avoid stagnation by allowing a wider exploration of the search space. In addition to the stochastic state transition rule, ACS\( |PR| \) also uses a global and a local pheromone trail evaporation rule. The global pheromone trail evaporation rule is applied towards the end of an iteration after all ants complete their migration plans. It is defined as

\[
\tau_s := (1 - \alpha) \cdot \tau_s + \alpha \cdot \Delta^{PR}_{\tau_s}
\]  

where \( \alpha \in (0, 1] \) is the pheromone decay parameter and \( \Delta^{PR}_{\tau_s} \) is the additional pheromone amount that is given only to those tuples that belong to the thus far best migration plan from ACS\( |PR| \) denoted as \( \Psi^+_{PR} \) in order to reward them. It is defined as

\[
\Delta^{PR}_{\tau_s} := \begin{cases} |P_R|, & \text{if } s \in \Psi^+_{PR} \\ 0, & \text{otherwise} \end{cases}
\]  

The local pheromone trail update rule is applied on a tuple when an ant traverses the tuple while making its migration plan. It is defined as

\[
\tau_s := (1 - \rho) \cdot \tau_s + \rho \cdot \tau_0
\]  

where \( \rho \in (0, 1] \) is similar to \( \alpha \) and \( \tau_0 \) is the initial pheromone level, which is computed as the multiplicative inverse of the product of the number of PMs \( |P| \) and the approximate optimal \( |\Psi| \)

\[
\tau_0 := (|P| \cdot |P|)^{-1}
\]  

Here, any very rough approximation of the optimal \( |\Psi| \) suffices [25]. The pseudo-random-proportional-rule in ACS\( |PR| \) and the global pheromone trail update rule are intended to make the search more
Algorithm 2 ACS-based colony to maximise the number of released PMs (ACS_{PR})

1: $\Psi^+_\text{PR} := \emptyset, M := \emptyset$
2: $\forall t \in T | T := t_0$
3: for $i \in [1, nI]$ do
4:   for $k \in [1, nA]$ do
5:     $\psi^m_k := \emptyset, \Psi_k := \emptyset, \text{Scr}_k := 0$
6:     while $|\psi^m_k| < |T|$ do
7:       compute $\text{probs}_t \forall s \in T$ using (7)
8:       choose a tuple $t \in T$ to traverse using (6)
9:       $\psi^m_k := \psi^m_k \cup \{t\}$
10:      apply local update rule in (11) on $t$
11:     if the migration in $t$ does not overload destination PM $p_{de}$ then
12:       update used capacity vectors $U_{p_sio}$ and $U_{p_{de}}$ in $t$
13:       if $f(\psi^m_k) > \text{Scr}_k$ then
14:         $\text{Scr}_k := f(\psi^m_k)$
15:       $\Psi_k := \Psi_k \cup \{t\}$
16:     end if
17:   end if
18: end while
19: $M := M \cup \{\Psi_k\}$
20: end for
21: $\Psi^+_{\text{PR}} := \arg \max_{\Psi_k \in M} f(\Psi_k)$
22: apply global update rule in (9) on all $s \in T$
23: end for
24: return $\Psi^+_{\text{PR}}$

directed. The pseudo-random-proportional-rule prefers tuples with a higher pheromone level and a higher heuristic value. Therefore, the ants try to search other high quality solutions in the close proximity of the thus far global best solution. On the other hand, the local pheromone trail update rule complements exploration of other high-quality solutions that may exist far from the thus far global best solution. This is because whenever an ant traverses a tuple and applies the local更新 rule, the tuple looses some of its pheromone and thus becomes less attractive for other ants. Therefore, it helps in avoiding stagnation where all ants end up finding the same solution or where they prematurely converge to a suboptimal solution.

The pseudocode of the ACS_{PR} algorithm is given as Algorithm 2. The algorithm makes a set of tuples $T$ using (1) and sets the pheromone value of each tuple to the initial pheromone level $t_0$ using (12) (line 2). Then, it iterates over $nI$ iterations (line 3), where each iteration $i \in nI$ creates a new generation of $nA$ ants that concurrently build their migration plans (lines 4–20). Each ant $k \in nA$ iterates over $|T|$ tuples (lines 6–18). It computes the probability of choosing the next tuple to traverse using (7) (line 7). Afterwards, based on the computed probabilities and the stochastic state transition rule in (6) and (7), each ant chooses a tuple $t$ to traverse (line 8) and adds $t$ to its temporary migration plan $\psi^m_k$ (line 9). The local pheromone trail update rule in (11) and (12) is applied on $t$ (line 10). If the migration in $t$ does not overload the destination PM $p_{de}$, the used capacity vectors at the source PM $U_{p_sio}$ and the destination PM $U_{p_{de}}$ in $t$ are updated to reflect the impact of the migration (line 12). Then, the objective function in (3) is applied on $\psi^m_k$. If it yields a score higher than the ant's thus far best score $\text{Scr}_k$ (line 13), $t$ is added to the ant-specific migration plan $\Psi_k$ (line 15). Afterwards, when all ants complete their migration plans, all ant-specific migration plans are added to the set of migration plans $M$ (line 19), each migration plan $\Psi_k \in M$ is evaluated by applying the objective function in (3), the thus far global best VM migration plan $\Psi^+$ is selected (line 21), and the global pheromone trail update rule in (9) and (10) is applied on all tuples (line 22). Finally, when all iterations $i \in nI$ complete, ACS_{PR} returns the thus far best migration plan from ACS_{PR} denoted as $\Psi^+_{\text{PR}}$ (line 24).

The time complexity of the ACS_{PR} algorithm is $O(nI \cdot |T|^2)$, where $nI$ is the number of ant generations and $|T|$ is the number of tuples. It can be derived from the pseudocode in Algorithm 2. The main loop
in line 3 iterates over $nI$. The second loop in line 4 does not add to the time complexity because the ants concurrently build their migration plans. The while loop in line 6 iterates over $|T|$. Finally, the probability calculation in line 7 requires an iteration over $|T|$.

### 3.2. ACS-based colony to minimise the number of VM migrations

The ACS$_{nM}$ algorithm tries to find a migration plan with fewer VM migrations, but with at least as many released PMs $P_R$ as in $\Psi^+$. Thus, the objective function for ACS$_{nM}$ is

$$\maximize \ g(\Psi) := (nM)^{-1}$$

where $\Psi$ is the migration plan and $nM$ is the number of VM migrations. Since VM migration is a resource-intensive operation, the objective function for ACS$_{nM}$ is defined as the multiplicative inverse of the number of VM migrations $nM$.

The ants in the ACS$_{nM}$ colony use the same pseudo-random-proportional-rule as in (6) and (7) to choose the next tuple to traverse. Moreover, as a general rule, the heuristic value $\eta_s$ in (8) favours tuples with a greater VM used capacity vector $U_v$. Therefore, the VM migrations that are more likely to result in a reduced number of VM migrations receive a higher amount of heuristic value $\eta_s$. Thus, the heuristic value $\eta_s$ in (8) supports the objective function of ACS$_{nM}$ in (13).

The ACS$_{nM}$ colony also uses the same local pheromone trail update rule as in (11). However, the global pheromone trail evaporation rule in ACS$_{nM}$ is defined as

$$\tau_s := (1 - \alpha) \cdot \tau_s + \alpha \cdot \Delta^n_{\tau_s}$$

where $\Delta^n_{\tau_s}$ is the additional pheromone amount that is given only to those tuples that belong to the thus far best migration plan from ACS$_{nM}$ denoted as $\Psi^+_n$ in order to reward them. It is defined as

$$\Delta^n_{\tau_s} := \begin{cases} 
(nM)^{-1}, & \text{if } s \in \Psi^+_n \\
0, & \text{otherwise} 
\end{cases}$$

The pseudocode of the ACS$_{nM}$ algorithm is given as Algorithm 3. Most of the steps in ACS$_{nM}$ are similar to those in the ACS$_{|PR|}$ colony in Algorithm 2 (lines 1–12). Similarly, in line 13, the algorithm uses the objective function concerning the number of released PMs $|P_R|$ defined in (3) instead of the objective function concerning the number of VM migrations $nM$ defined in (13) because at this step it is important to find a migration plan with a higher number of released PMs $|P_R|$. However, when selecting the thus far best migration plan from ACS$_{nM}$ denoted as $\Psi^+_n$ (line 21), all ant-specific migration plans $\Psi_k \in M$ are evaluated first by applying the objective function concerning the number of released PMs $|P_R|$ defined in (3) and then by the objective function concerning the number of VM migrations $nM$ defined in (13). Therefore, the migration plan with the highest number of released PMs $|P_R|$ and a lower number of VM migrations $nM$ is selected as the best migration plan $\Psi^+_n$. Afterwards, the algorithm applies the global pheromone trail update rule of the ACS$_{nM}$ colony defined in (14) and (15) on all tuples (line 22). Finally, it returns $\Psi^+_n$ (line 24).

The time complexity of the ACS$_{nM}$ algorithm is similar to that of the ACS$_{|PR|}$ algorithm. Since the ACS$_{|PR|}$ and ACS$_{nM}$ colonies work concurrently and independently, the overall time complexity of the proposed MOACS algorithm for finding the global best migration plan $\Psi^+$ is $O(nl \cdot |T|^2)$.

### 4. Evaluation

In this section, we describe the experimental evaluation of the proposed MOACS algorithm and its comparison with the single-objective, single-colony MMAS VM consolidation algorithm (Feller-ACO) by Feller et al. [21] and our previously published single-objective, single-colony ACS VM consolidation algorithm (ACS) [4].
Algorithm 3 ACS-based colony to minimise the number of VM migrations (ACS\textsubscript{nM})

1: \( \Psi^+_{nM} := \emptyset, M := \emptyset \)
2: \( \forall t \in T, t_0 := \tau_0 \)
3: \( \text{for } i \in [1, n] \) do
4: \( \text{for } k \in [1, n] \) do
5: \( \psi^m_k := \emptyset, \psi^m_k := \emptyset, \text{Sc}_{rk} := 0 \)
6: \( \text{while } |\psi^m_k| < |T| \) do
7: \( \text{compute } \text{prob}_s \ \forall s \in T \text{ using (7)} \)
8: \( \text{choose a tuple } t \in T \text{ to traverse using (6)} \)
9: \( \psi^m_k := \psi^m_k \cup \{t\} \)
10: \( \text{apply local update rule in (11) on } t \)
11: \( \text{if the migration in } t \text{ does not overload destination PM } \rho_{de} \text{ then} \)
12: \( \text{update used capacity vectors } U_{psio} \text{ and } U_{pde} \text{ in } t \)
13: \( \text{if } f(\psi^m_k) > \text{Sc}_{rk} \text{ then} \)
14: \( \text{Sc}_{rk} := f(\psi^m_k) \)
15: \( \psi^m_k := \psi^m_k \cup \{t\} \)
16: \( \text{end if} \)
17: \( \text{end if} \)
18: \( \text{end while} \)
19: \( M := M \cup \{\psi^m_k\} \)
20: \( \text{end for} \)
21: \( \psi^+_{nM} := \arg \max \psi^m_k \in M \{f(\psi_k)\} \land \arg \max \psi^m_k \in M \{g(\psi_k)\} \)
22: \( \text{apply global update rule in (14) on all } s \in T \)
23: \( \text{end for} \)
24: \( \text{return } \psi^+_{nM} \)

We have implemented our proposed MOACS algorithm as a Java program called the MOACS Solver. It is available online under an open-source license.\(^2\) We have also developed Java solvers for the Feller-ACO [21] and ACS [4] algorithms.

4.1. Experimental design

The objective of the experiment was to compare the performance of the three implemented solvers: ACS, MOACS, and Feller-ACO. The input of these algorithms is a VM consolidation problem that can be characterised by the following parameters: number of PMs, number of VMs to consolidate, CPU utilisation of each VM, memory requirements of each VM, and the current location of each VM. We used a factorial experiment design [32], in which the three solvers were tested in four different scenarios: (1) low CPU and small memory requirements with respect to the capacity of the PMs, (2) high CPU and large memory requirements, (3) high CPU and small memory requirements, and (4) low CPU and large memory requirements. The experimental used randomly generated workloads, homogeneous VMs, and homogeneous PMs. The experimental parameters are summarised in Tables 2 and 3. For Scenario 1, the number of VMs to consolidate was 1000 and the number of PMs was 100 (ratio 10:1), while in the other three scenarios there were 1000 VMs and 200 PMs (ratio 5:1). The neighbourhood size was set to 5 and the neighbours were chosen randomly. The ACO parameters used in the ACS, MOACS, and Feller-ACO solvers are tabulated in Table 3. These parameter values were obtained in a series of preliminary experiments. The dependent variables of the experiment were:

- Number of released PMs after consolidation, to be maximised.
- Packing efficiency, defined as the ratio between the number of released PMs and the total number of PMs, to be maximised.
- Number of VM migrations during consolidation, to be minimised.
- Solver execution time, to be minimised.
Table 2. Experiment design.

| Algorithm                  | Low                  | High                  |
|----------------------------|----------------------|-----------------------|
| ACS, MOACS, Feller-ACO     | Scenario 1           | Scenario 3            |
| Memory                     | $|V| = 1000$          | $|V| = 1000$          |
|                            | $|P| = 100$           | $|P| = 200$           |
|                            | $|N| = 5$             | $|N| = 5$             |
|                            | Number of runs = 10  | Number of runs = 10  |
| Small                      |                      |                       |
|                            | $|V| = 1000$          | $|V| = 1000$          |
|                            | $|P| = 200$           | $|P| = 200$           |
|                            | $|N| = 5$             | $|N| = 5$             |
|                            | Number of runs = 10  | Number of runs = 10  |
| Large                      |                      |                       |

Table 3. ACO parameters.

| $\alpha$ | $\beta$ | $\rho$ | $q_0$ | $n_A$ | $n_I$ |
|-----------|---------|--------|-------|-------|-------|
| 0.1       | 2.0     | 0.1    | 0.9   | 10    | 2     |

4.2. Execution

The three solvers under evaluation used approximated algorithms. Therefore, we ran each solver 10 times for each scenario, every time with a different random seed. Consequently, the experiment comprised a total of 40 test runs for each solver. The experiments were run on an Intel Core i7-4790 processor with 16 gigabytes of memory.

4.3. Results

4.3.1. Number of released PMs and packing efficiency

Figure 2 presents the number of released PMs by the ACS, MOACS, and Feller-ACO solvers for the different scenarios as box plots. Moreover, Table 4 provides a summary of the results in the numerical form. The table also provides the packing efficiency achieved by each solver. This variable is derived easily from the number of released PMs.

The results show that the MOACS solver was able to release 25–37% more PMs than the Feller-ACO solver, depending on the scenario. For example, in Scenario 1, MOACS released 15 PMs (median of 10 test runs) while Feller-ACO released only 11 PMs (median of 10 test runs). Since the packing efficiency is derived from the number of released PMs, it follows a similar trend. The difference in the number of released PMs between the MOACS and Feller-ACO solvers is statistically significant (Wilcoxon Signed-Rank Test, $p$-value = 0.005).

4.3.2. Number of VM migrations

The third dependent variable of interest in our experiment was the number of VM migrations, which should be minimised. Figure 3 presents the results for this variable for the three solvers in the graphical form while Table 5 provides a summary of the results in the numerical form.

Again, we can observe that MOACS outperforms Feller-ACO for this objective. The results show that the MOACS solver was required to perform only 82–83% of the number of migrations required by the Feller-ACO solver to achieve an even better packing efficiency. For example, in Scenario 1, MOACS required 189 migrations (median of 10 test runs) while Feller-ACO required 226 migrations (median of 10 test runs). The difference in the number of migrations per solver is statistically significant (Wilcoxon Signed-Rank Test, $p$-value = 0.006).
4.3.3. Execution time and scalability

The last comparison attribute is the execution time required by each solver to find a near-optimal, global best migration plan. Ideally, the solvers should use as less time as possible. Figure 4 presents the execution time for the ACS, MOACS, and Feller-ACO solvers when solving problems based on Scenario 2 with the number of PMs varying from 50 to 500 in increments of 50 and the number of VMs varying from 250 to 2500 in increments of 250.

We can observe in the figure that MOACS performed better than Feller-ACO. Moreover, ACS performed better than MOACS but, to be fair, it used a single-objective algorithm while MOACS explored the search-space for two different objectives.

For reference, we also report the execution times to solve Scenario 1 with $|P| = 100$ and $|V| = 1000$ in Table 6. For this scenario, ACS required a bit more than one minute. In contrast, MOACS required...
almost 2 min while Feller-ACO required almost 6 min. We report the median value for 10 test runs. However, we have observed that the standard deviation for the time variable was rather small for all solvers. The difference in the execution time between the MOACS and Feller-ACO solver is statistically significant (Wilcoxon Signed-Rank Test, $p$-value = 0.002).

4.4. Analysis

We can observe that the proposed MOACS algorithm and its corresponding solver outperformed Feller-ACO in all the measured variables: number of released PMs and packing efficiency (change 125%), number of VM migrations (change 82%), and execution time (speedup 2.97×). The solvers were exercised in four different scenarios involving different VM requirements. Each scenario was
### Table 4. Number of released PMs.

| Scenario   | Median released | SD released | Packing | Efficiency | Change | p-value |
|------------|-----------------|-------------|---------|------------|--------|---------|
| Scenario 1 | ACS 9           | MOACS 15    | Feller-ACO 11 | 136%       | 0.005  |
|            | 0.63            | 0.52        | 0.82    |            |        |         |
|           | 9%              | 15%         | 11%     |            |        |         |
| Scenario 2 | ACS 7           | MOACS 11    | Feller-ACO 8 | 137%       | 0.005  |
|            | 0.67            | 0.67        | 0.99    |            |        |         |
|           | 3.5%            | 5.5%        | 4%      |            |        |         |
| Scenario 3 | ACS 7           | MOACS 10    | Feller-ACO 8 | 125%       | 0.004  |
|            | 0.67            | 0.52        | 0.82    |            |        |         |
|           | 3.5%            | 5%          | 4%      |            |        |         |
| Scenario 4 | ACS 7           | MOACS 11    | Feller-ACO 8 | 137%       | 0.005  |
|            | 0.88            | 0.88        | 0.74    |            |        |         |
|           | 3.5%            | 5.5%        | 4%      |            |        |         |

### Table 5. Number of VM migrations.

| Scenario   | Median migrations | SD migrations | Packing | Efficiency | Change | p-value |
|------------|-------------------|---------------|---------|------------|--------|---------|
| Scenario 1 | ACS 201.5         | MOACS 189     | Feller-ACO 226 | 83%       | 0.002  |
|            | 3.13              | 4.77          | 5.23    |            |        |         |
| Scenario 2 | ACS 176           | MOACS 154.5   | Feller-ACO 186 | 83%       | 0.006  |
|            | 3.37              | 2.31          | 4.52    |            |        |         |
| Scenario 3 | ACS 173           | MOACS 154.5   | Feller-ACO 187.5 | 82%       | 0.005  |
|            | 3.34              | 2.58          | 2.35    |            |        |         |
| Scenario 4 | ACS 171           | MOACS 154     | Feller-ACO 186 | 83%       | 0.006  |
|            | 3.14              | 2.07          | 0.74    |            |        |         |

### Figure 4. Scalability of the ACS, MOACS, and Feller-ACO solvers.
Table 6. Execution time (in minutes) for the three solvers for Scenario 1.

| | ACS | MOACS | Feller-ACO | Speedup | p-value |
|---|---|---|---|---|---|
| $|P| \cdot |V| = 100000$ | Median time | 1.03 | 1.99 | 5.92 | 2.97x | 0.002 |
| | Sd time | 0.03 | 0.08 | 0.17 | |

evaluated in 10 independent test runs. The differences were statistically significant for all variables (Wilcoxon Signed-Rank Test, p-values less than or equal to 0.006).

The differences in the performance can be explained by the design of each algorithm. Feller-ACO uses a single-objective, single-colony MMAS algorithm with an AOF that combines two different objectives concerning the number of released PMs and the number of VM migrations, whereas MOACS uses a multi-objective algorithm with two independent ant colonies for optimising the two objectives. The AOF approach in Feller-ACO uses several parameters to determine the relative importance of the two objectives in the overall optimisation process. We consider that this approach has two drawbacks: (1) it is difficult to find appropriate values for the different parameters in an AOF and (2) an AOF may not combine the different objectives in an appropriate manner. For instance, as described in Section 3, maximising the number of released PMs takes precedence in MOACS over minimising the number of VM migrations. However, the AOF in Feller-ACO does not support precedence of one objective over another. Finally, MOACS uses additional constraints over its search-space, which significantly reduces the algorithm execution time without compromising on the quality of the solutions. The experimental evaluation clearly showed that these design decisions have an actual impact on the performance of the solvers.

It was also interesting to compare MOACS to ACS. MOACS clearly released more PMs and required less VM migrations than ACS. However, it was slower than ACS. When comparing ACS and Feller-ACO, we observed that ACS was faster and required less VM migrations than Feller-ACO, although it did not achieve the same packing efficiency. Still, ACS was the fastest of the three solvers and it can be a good alternative to consider when execution time is critical.

5. Conclusion

We presented a novel multi-objective ACS algorithm for VM consolidation in cloud data centres. The proposed algorithm builds VM migration plans, which are then used to reduce the number of required PMs by migrating and consolidating VMs on under-utilised PMs. It optimises two objectives that are ordered by their importance. The first and foremost objective in the proposed algorithm is to maximise the number of released PMs. Moreover, since VM migration is a resource-intensive operation, it also tries to minimise the number of VM migrations.

The proposed algorithm was evaluated in a series of experiments. The experimental evaluation compared the proposed algorithm with two previously published ACO based VM consolidation algorithms, which were chosen for comparison due to their excellent performance with respect to different attributes. We considered four different scenarios of interest to test the three algorithms under different VM configurations. The experimental results showed that the proposed algorithm provided an efficient solution for VM consolidation in cloud data centres. Moreover, it outperformed the two existing ACO based VM consolidation algorithms in terms of number of released PMs, packing efficiency, and number of VM migrations.

Notes

1. www.ibm.com/software/commerce/optimization/cplex-optimizer.
2. https://github.com/SELAB-AA/moacs-wac.
Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the Need for Speed (N4S) Research Program (http://www.n4s.fi) of the DIGILE, the Finnish Strategic Centre for Science, Technology and Innovation in the Field of ICT [grant number 2328/31/2013].

Notes on contributors

Adnan Ashraf received MSc and MS degrees in Computer Science from Mohammad Ali Jinnah University, Islamabad, Pakistan in 2003 and 2006, respectively, and doctoral degree in Software Engineering from Åbo Akademi University, Turku, Finland in 2014. He currently works as a postdoctoral researcher at Åbo Akademi University. His research interests include cloud computing and search-based software engineering.

Ivan Porres is a professor in Software Engineering and head of the Computer Engineering education at Åbo Akademi University. He is the leader of the Software Engineering Laboratory at the Turku Centre for Computer Science (TUCS) and principal investigator at Åbo Akademi for the Cloud Software Finland (2009–2013) and N4S (2014–2017) projects at the DIGILE, the Finnish Strategic Centre for Science, Technology and Innovation in the Field of ICT. He has received the Ten-Year Most Influential Paper Award at the ACM/IEEE Conference on Model Driven Engineering Languages and Systems in two occasions and has participated in many review appointments and the organisation of research events.

ORCID

Adnan Ashraf http://orcid.org/0000-0001-8015-2335

References

[1] Kaur T, Chana I. Energy efficiency techniques in cloud computing: a survey and taxonomy. ACM Comput Surv. 2015;48:22:1–22:46.
[2] Mastelic T, Oleksiak A, Claussen H, et al. Cloud computing: survey on energy efficiency. ACM Comput Surv. 2014;47:33:1–33:36.
[3] Beloglazov A, Buyya R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency Comput: Pract Experience. 2012;24:1397–1420.
[4] Farahnakian F, Ashraf A, Pahikkala T, et al. Using ant colony system to consolidate VMs for green cloud computing. IEEE Trans Serv Comput. 2015;8:187–198.
[5] Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst. 2012;28:755–768.
[6] Corradi A, Fanelli M, Foschini L. VM consolidation: a real case based on OpenStack cloud. Future Gener Comput Syst. 2014;32:118–127.
[7] Ferreto TC, Netto MA, Calheiros RN, et al. Server consolidation with migration control for virtualized data centers. Future Gener Comput Syst. 2011;27:1027–1034.
[8] He S, Guo L, Ghanem MM, et al. Improving resource utilisation in the cloud environment using multivariate probabilistic models. In: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD); 2012; Honolulu, HI, USA. p. 574–581.
[9] Hwang I, Pedram M. Hierarchical virtual machine consolidation in a cloud computing system. In: 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD); 2013. p. 196–203.
[10] Liao X, Jin H, Liu H. Towards a green cluster through dynamic remapping of virtual machines. Future Gener Comput Syst. 2012;28:469–477.
[11] Murtazaev A, Oh S. Sercon: Server consolidation algorithm using live migration of virtual machines for green computing. IETE Tech Rev. 2011;28:212–231.
[12] Marzolla M, Babaoglu O, Panzieri F. Server consolidation in clouds through gossiping. 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM); 2011; Lucca, Italy.
[13] Wang M, Meng X, Zhang L. Consolidating virtual machines with dynamic bandwidth demand in data centers. In: Proceedings of IEEE INFOCOM; 2011 Apr; Shanghai, China. p. 71–75.
[14] Wood T, Shenoy P, Venkataramani A, et al. Sandpiper: black-box and gray-box resource management for virtual machines. Comput Networks. 2009;53:2923–2938.

[15] Vogels W. Beyond server consolidation. ACM Queue. 2008;6:20–26.

[16] Ahmad RW, Gani A, Hamid SHA, et al. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. J Network Comput Appl. 2015;52:11–25.

[17] Pires FL, Baràn B. A virtual machine placement taxonomy. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid); 2015 May; Shenzhen, China. p. 159–168

[18] Farahnakian F, Ashraf A, Liljeberg P, et al. Energy-aware dynamic VM consolidation in cloud data centers using ant colony system. 7th IEEE International Conference on Cloud Computing (CLOUD); 2014; Anchorage, AK, USA.

[19] Blum C, Puchinger J, Raidl GR, et al. Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput. 2011;11:4135–4151.

[20] Harman M, Lakhota K, Singer J, et al. Cloud engineering is search based software engineering too. J Syst Softw. 2013;86:2225–2241.

[21] Feller E, Morin C, Esnault A. A case for fully decentralized dynamic VM consolidation in clouds. In: IEEE International Conference on Cloud Computing Technology and Science; 2012; Taipei, Taiwan. p. 26–33.

[22] Ferdaus M, Murshed M, Calheiros R, et al. Virtual machine consolidation in cloud data centers using ACO metaheuristic. In: Silva F, Dutra I, Santos Costa V, editors. Euro-Par 2014 parallel processing. Springer International Publishing; 2014; Porto, Portugal. p. 306–317. (Lecture Notes in Computer Science; vol. 8632).

[23] Dorigo M, Di Caro G, Gambardella LM. Ant algorithms for discrete optimization. Artif Life. 1999;5:137–172.

[24] Hu XB, Wang M, Paolo ED. Calculating complete and exact pareto front for multiobjective optimization: a new deterministic approach for discrete problems. IEEE Trans Cybern. 2013;43:1088–1101.

[25] Dorigo M, Gambardella L. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput. 1997;1:53–66.

[26] Gambardella LM, Taillard É, Agazzi G. MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. In: Corde D, Dorigo M, Glover F, editors. New ideas in optimization. McGraw-Hill: London; 1999. p. 63–76.

[27] Ashraf A, Porres I. Using ant colony system to consolidate multiple web applications in a cloud environment. In: 22nd Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP); 2014; Torino, Italy. p. 482–489.

[28] Yin PY, Wang JY. Ant colony optimization for the nonlinear resource allocation problem. Appl Math Comput. 2006;174:1438–1453.

[29] Chaharsooghi SK, Kermani AHM. An effective ant colony optimization algorithm (ACO) for multi-objective resource allocation problem (MORAP). Appl Math Comput. 2008;200:167–177.

[30] Gao Y, Guan H, Qi Z, et al. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J Comput Syst Sci. 2013;79:1230–1242.

[31] Mishra M, Sahoo A. On theory of VM placement: anomalies in existing methodologies and their mitigation using a novel vector based approach. In: 2011 IEEE International Conference on Cloud Computing (CLOUD); 2011 Jul. p. 275–282; Washington, DC, USA.

[32] Wohlin C, Runeson P, Höst M, et al. Experimentation in software engineering. 1st ed. Springer-Verlag Berlin Heidelberg; 2012.