Alresheedi *et al. Hum. Cent. Comput. Inf. Sci.* (2019) 9:15
https://doi.org/10.1186/s13673-019-0174-9

Human-centric Computing and Information Sciences

RESEARCH Open Access

Improved multiobjective salp swarm optimization for virtual machine placement in cloud computing

Shayem Saleh Alresheedi[1], Songfeng Lu[1,2*], Mohamed Abd Elaziz[1,3] and Ahmed A. Ewees[4,5]

*Correspondence: lusongfeng@hust.edu.cn
[1] School of Computer Science & Technology, Huazhong University of Science and Technology, Wuhan 430074, China
Full list of author information is available at the end of the article

Abstract

In data center companies, cloud computing can host multiple types of heterogeneous virtual machines (VMs) and provide many features, including flexibility, security, support, and even better maintenance than traditional centers. However, some issues need to be considered, such as the optimization of energy usage, utilization of resources, reduction of time consumption, and optimization of virtual machine placement. Therefore, this paper proposes an alternative multiobjective optimization (MOP) approach that combines the salp swarm and sine-cosine algorithms (MOSSASCA) to determine a suitable solution for virtual machine placement (VMP). The objectives of the proposed MOSSASCA are to maximize mean time before a host shutdown (MTBHS), to reduce power consumption, and to minimize service level agreement violations (SLAVs). The proposed method improves the salp swarm and the sine-cosine algorithms using an MOP technique. The SCA works by using a local search approach to improve the performance of traditional SSA by avoiding trapping in a local optimal solution and by increasing convergence speed. To evaluate the quality of MOSSASCA, we perform a series of experiments using different numbers of VMs and physical machines. The results of MOSSASCA are compared with well-known methods, including the nondominated sorting genetic algorithm (NSGA-II), multiobjective particle swarm optimization (MOPSO), a multiobjective evolutionary algorithm with decomposition (MOEAD), and a multiobjective sine-cosine algorithm (MOSCA). The results reveal that MOSSASCA outperforms the compared methods in terms of solving MOP problems and achieving the three objectives. Compared with the other methods, MOSSASCA exhibits a better ability to reduce power consumption and SLAVs while increasing MTBHS. The main differences in terms of power consumption between the MOSCA, MOPSO, MOEAD, and NSGA-II and the MOSSASCA are 0.53, 1.31, 1.36, and 1.44, respectively. Additionally, the MOSSASCA has higher MTBHS value than MOSCA, MOPSO, MOEAD, and NSGA-II by 362.49, 274.70, 585.73 and 672.94, respectively, and the proposed method has lower SLAV values than MOPSO, MOEAD, and NSGA-II by 0.41, 0.28, and 1.27, respectively.

Keywords: Cloud computing, Virtual machine placement, Multiobjective optimization, Salp swarm algorithm, Sine-Cosine algorithm

Introduction

Currently, the evolution and spread of technology double the quantity of data several times every minute; therefore, traditional data centers, which store these data locally, do not work as efficiently as before and have become unsuitable for many companies. Thus, traditional data centers should be improved to overcome several issues, including high maintenance costs, wasted floor space, high energy consumption, security, and high human resource costs. In addition, quality of service (QoS) requires companies to introduce high-quality services for their clients, including sending tasks to resources, task scheduling, and output satisfying results in an appropriate amount of time [1]. These problems can be approached by cloud computing servers.

Cloud computing can help the owners of datacenters overcome several issues through its flexibility, security, high scalability, QoS, and improved maintenance and support [1]. Cloud computing has the ability to host multiple types of heterogeneous servers; these servers host thousands of virtual machines (VMs); and each machine is provided with suitable resources (e.g., CPU, RAM, and storage) to be able to run all processes. Some cloud computing servers hold only one VM, while most servers host multiple VMs. This structure may cause overloading problems in some servers, which can reduce the performance of the servers and consume a large amount of energy [2].

VMs have also different workload types and resource specifications; thus, this may result in an imbalance in resource usage and high energy consumption [1]. Therefore, many issues need to be considered, such as the utilization of all resources, reduction of time consumption, and optimization of energy usage. The consumption of energy leads to increases in operating costs and it is related to resource utilization in datacenters. In this context, the energy cost of Amazon's datacenters consumed more than 40% of its total operating costs [3]. In addition, the increase in energy consumption causes an increase carbon dioxide emissions and ecological problems [3]. Therefore, if energy consumption is reduced, the operating costs of datacenters will be reduced. An effective technique to achieve this goal is to apply and optimize the VM placement (VMP) process [4].

VMP relates to the allocation of VMs to a relevant physical machine (PM). This process is considered to be an important problem in managing datacenter resources. It should be done in real-time without large time consumption. If this process consumes more time, it will reduce its main benefits of being applied in a real environment. This research topic has several optimization criteria that can be effective in achieving highly positive economic and ecological results. For this reason, resource allocation in cloud computing platforms has attracted increasing attention and has been widely studied by many researchers [5–8].

The advantages of meta-heuristic algorithms in solving many machine learning optimization issues has led several studies to successfully employ them in different fields, such as image processing, feature selection, classification, and prediction [9–12]. In the same way, several meta-heuristic methods have been introduced, such as thermal exchange optimization [13], that depend on Newton's law of cooling. The law assumes there is a relationship between the rate of a body's heat loss and the temperature difference between the body and its surroundings. This algorithm is used in some applications such as solving global optimization [13] and structural damage identification [14]. Selfish

herd optimizer (SHO) [15] is another meta-heuristic that emulates selfish herd behavior. The SHO is used to solve global optimization problems [15], and there is another version of SHO that depends on the opposite-based learning method and that applied this algorithm to an optimization function [16]. The tree-seed algorithm (TSA) has been applied to address continuous optimization problems [17]. The TSA simulates the relationship between trees and their seeds. This algorithm has several applications: large-scale binary optimization [18], optimal power flow problem [19], and similarity and logic gate binary optimization [20].

Therefore, we introduce an alternative method called MOSSASCA to find a suitable solution for the VMP problem. MOSSASCA is a multiobjective salp swarm algorithm (SSA) using sine-cosine algorithms (SCAs) as local operators.

SSA is a recently developed optimization method that simulates the natural behavior of salp, barrel-shaped plankton (family Salpidae) that are mostly water by weight. In addition, they move in the same way as jellyfish. SSA was proposed to solve different kinds of optimization issues [21]. SCA is an evolutionary method proposed to provide various solutions to different problems. It uses the mathematical model of sine and cosine functions and switches between them to obtain the optimal solution [22]. These methods were used in many previous studies and have shown good results [23–25].

The objectives of the proposed MOSSASCA are to maximize the mean time before a host shutdown, to reduce the power consumption and to minimize service level agreement violations (SLAVs). The proposed method starts by generating an integer population that represents solutions for the VMP and evaluates the quality of each solution by computing the objective functions. Then, the nondominated (ND) solutions are determined and saved in the archive. The next step uses the leader selection method to choose the best solution from the archive that is applied to update other solutions. Thereafter, the solutions in the current population are updated using the modified version of SSA, based on SCA. In MOSSASCA, the operators of SCA are used as local operators to improve the solutions of SSA and preventing them from becoming stuck at a local optimal point. All solutions updated by MOSSASCA are added to the archive. ND solutions are found and then are updated based on the size of the archive. The previous steps are executed until reaching stop conditions.

The rest of this study is organized as follows. "Related work" section lists the related works and problem formulation. "Background" secion describes the background of the methods used in this study. "Proposed MOSSASCA for VMP method" section explains the proposed method. The experiments and results are explained in "Results and discussion" section. The conclusion and future work are illustrated.

Related work

The massive numbers of data centers consume a substantial amount of power resources; this situation negatively affects the stability and extension of these centers. Therefore, this issue should be considered when designing cloud computing platforms [3, 26, 27]. In this context, various studies have been performed to overcome the problems of VMP such as power consumption, network traffic minimization, resource utilization, and performance maximization [28–32].

The techniques that can be applied to solve the problem of VMP resource allocation can be defined as multiobjective (MO) or single-objective. The process of finding the solution for more than one objective function simultaneously with different criteria is considered a multiobjective optimization problem (MOP).

Different types of meta-heuristic (MH) techniques are applied to solve the problem of single-objective VMPs. For example, Gabay et al. [33] applied first-fit decreasing, best-fit decreasing, and worst-fit techniques to locate destination PM for placing VMs. The authors reported that their proposed method is flexible, fast, and appropriate to be applied to real-life problems.

The authors of [34, 35] introduced simulated annealing (SA) algorithm-based VM consolidation algorithm.

However, single-objective methods are not suitable in the case of balancing between different objectives; therefore, multiobjective (MOP) methods are used to find solutions that balance between the objectives.

MOP methods have been applied to different applications, such as restructuring traffic networks [36], data mining methods for knowledge discovery [37], multiple sequence alignment [38], wind turbine blade geometry design [39] and others [40, 41].

Several MOP methods have been proposed, including in [42], a multiobjective method based on the grasshopper optimization algorithm (GOA). This algorithm simulates the navigation behavior of grasshoppers in nature. The authors of [43] proposed an MO version of the ant lion optimizer (ALO) and applied it to solve engineering design problems. In [44], the multiobjective flower pollination algorithm is proposed, which simulates the behavior of plant flowers' proliferation role. For finding solutions for constrained MOP problems, a modified water cycle algorithm has been proposed [45].

With all of this attention on MOP methods, minimal efforts have been exerted toward the optimization of multiobjective VMP problems by MH and hybrid-heuristic algorithms. For example, the authors of [2] applied biogeography-based optimization (BBO) as an optimization method in order to determine a solution for the VMP problem considering server loads, inter-VMs, power consumption, resource wastage, and storage network traffic. In addition, the authors of [3] introduced a hybrid genetic algorithm (GA) to solve the VMP problem in a communication network and PMs in a data center. Gao et al. [46] proposed an MO ant colony optimization (ACO) algorithm to deal with VMP issues; it was applied to determine efficient ND solutions that minimize power consumption and resource usage. Other studies proposed intelligent models to decrease the power consumption in a cloud environment such as [47–51].

However, these models suffer from slow convergence and become stuck in local optima. In addition, the no free lunch (NFL) theorem [52] assumes that the optimization method does not have the ability to solve all problems with the same quality. Therefore, this consideration motivated us to propose an alternative multiobjective VMP method that can overcome the limitations of other methods.

The proposed MOP VNP method depends on improving the traditional SSAs using the operators of the SCA. The SSA has demonstrated its performance in several fields; for example, the SSA is used to determine the suitable parameters of the solar cell [53, 54] and parameters of PEM fuel cells [23]. In addition, SSA is applied to improve classification through selecting relevant features [24, 25, 55–57]. In [58], the authors presented a method

based on SSA for practical considerations in radial distribution systems that determine optimal conductor and hosting capacity. For the fractional order proportional-integral-derivative (FOPID) controller problem, the SSA is used [59]. In [60], the authors presented an alternative job shop scheduling problem, whereas [61] presented training neural networks using SCA for improving pattern classification.

In general, the contributions of this study can be formulated as follows:

- Proposing an alternative optimization approach that uses the multiobjective optimization concept to enhance the properties of SSA based on SCA as a local search.
- Applying the MOSSASCA method to solve the problem of VMP and allocating resources in cloud computing platforms.

Background

Problem formulation of VMP

Physical cloud computing structures consist of many PMs. They contain the physical layers, or infrastructure, of cloud computing and they contain many components such as storage, RAM, CPUs, GPUs, and network connectors [8]. Therefore, cloud computing would not work without PMs. Each PM contains multiple VMs, where each VM can run an independent operating system (OS) and applications. VMs require many resources include CPU time, memory space, and storage. Therefore, a VM obtains these resources from its PM. Figure 1 illustrates the VMP problem.

Figure 1 contains four servers that run four VMs. Four virtualized applications are hosted on the system. The following points describe the process of VMP.

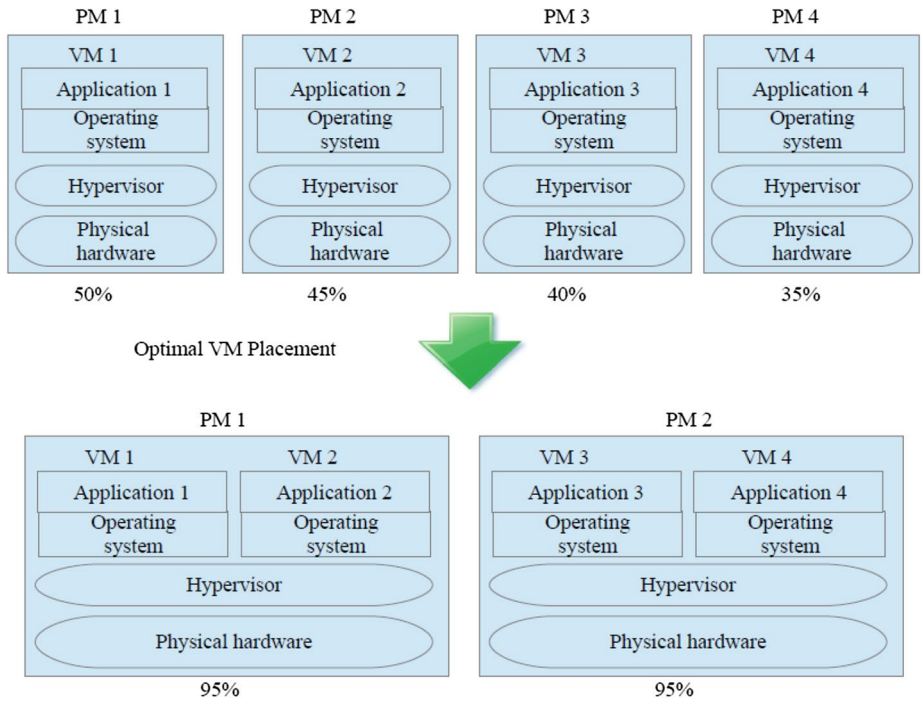

Fig. 1 VMP problem

1. For each PM, calculate the requirement of the application's resource through the statistics of the usage of the server's resource in a period of time.
2. Select a proper PM with the comparable type of CPU, compatible software, storage, and similar network properties.
3. Assign VM_1 to the PM_1 in step 2, and assign VM_2 to the PM_1 if it has sufficient resources.
4. Repeat this mechanism until each VM is assigned to a suitable PM and, if required, add a new PM to build the structure of a server farm.

The following subsections describe the PMs, VMs, constraints, and objective function.

Physical machine

The PMs consist of more than one machine, and all these machines have physical layers, as shown in the following equation:

$$PM = [PM_1, PM_2, PM_3, \ldots, PM_{N_P}], \tag{1}$$

where N_P describes the total number of PMs. Each PM has the following:

$$PM_i = (PM_i^{CPU}, PM_i^{ram}, PM_i^{hdd}, PM_i^{pmax}), \tag{2}$$

where $PM_i, (i = 1, 2, \ldots, N_P)$ defines the number of the current PMs. PM_i^{pmax} is the maximum power consumption of the ith PM in units of $[W]$. The PM_i^{CPU}, PM_i^{ram} and PM_i^{hdd} are the processing resources (in [ECU]), the memory resources (in [GB]), and the storage resources (in [GB]) of the PM_i.

Virtual machine

The VMs consist of one or more VMs, as shown in the following equation:

$$VM = [VM_1, VM_2, VM_3, \ldots, VM_{N_V}], \tag{3}$$

where N_V describes the total number of VMs. Each VM has the following resources:

$$VM_j = (VM_j^{CPU}, VM_j^{ram}, VM_j^{hdd}), \quad j = 1, 2, \ldots, N_V, \tag{4}$$

where j is the current VM's number. VM_j^{CPU}, VM_j^{ram}, and VM_j^{hdd} represent the processing, memory, and the storage requirements of the jth VM, respectively.

Objective functions

In this section, the three objective functions are defined below:

– Energy consumption: The servers consume a large volume of power when they are in an idle state.

$$f^{Energy} = \sum_{i=1}^{N_V}((PM_i^{pmax} - PM_i^{pmin}) \times U_{CPU_i}(g) + PM_i^{pmin}) \times Y_i(g), \tag{5}$$

where f^{Energy} represents the total power consumption of the PMs, while $PM_i^{pmin} = PM_i^{pmax} \times 0.6$ [62, 63] defines the minimum power consumption of PM_i. $U_{CPU_i}(g)$ represents the utilization ratio of resource utilized by PM_i at instant t, while $Y_i(g) \in [0, 1]$ is equal to 1 if the PM_i is turned on; otherwise, $Y_i(g) = 0$.

– SLAV: The infrastructure of cloud computing and hosts try to meet QoS requirements, which are modeled in the SLAV form to maximum response time or minimum throughput. SLAV can be caused by a host, as defined in the following equations.

$$f^{SLAV} = SLAO \times SLAM, \tag{6}$$

where $SLAO$ represents the average ratio of the period in the case the host experiences CPU utilization of 100% and is defined as follows:

$$SLAO = \frac{1}{N_P} \sum_{i=1}^{N_P} \frac{T_{si}}{T_{ai}}, \tag{7}$$

where T_{ai} is the active time of i-th host, and T_{si} represents the total time when the i-th host experiences 100% SLAV utilization. Moreover, the $SLAM$ represents the degradation in the performance due to the migration of VMs and is defined as follows:

$$SLAM = \frac{1}{N_V} \sum_{j=1}^{N_V} \frac{C_{dj}}{C_{rj}}, \tag{8}$$

where C_{rj} indicates the total CPU utilization needed by the VM_j, and C_{dj} represents the degradation in the performance that results from VM migration.

– Mean time before a host shutdown (MTBHS): This time is measured in seconds, and the average is calculated as follows.

$$f_j^{MTBHS} = \frac{1}{N_P} \sum_{i=1}^{N_P} h_i, \tag{9}$$

where h_i represents the host shutdown time.

Constraints:

$$\sum_{i=1}^{N_P} B_{ji}(g) \leq 1 \; \forall j \in \{1, \ldots, N_V\}, \tag{10}$$

$$\sum_{i=1}^{N_P} VM_{cpu_j} \times B_{ji}(g) \leq PM_{CPU_i}, \tag{11}$$

$$\sum_{i=1}^{N_P} VM_{ram_j} \times B_{ji}(g) \leq PM_{ram_i}, \tag{12}$$

$$\sum_{i=1}^{N_P} VM_{hdd_j} \times PM_{ji}(g) \leq PM_{hdd_i}, \tag{13}$$

where $B_{ji} = 1$ when the VM_j allocates on PM_i; otherwise $B_{ji} = 0$. In constraint (10), VM_i should be executed on a single PM_j and when its $SLAV_i$ is lowest priority (in this work, s = 2). Here, it should not allocated to any PM. In constraints (11)–(13), the current PM (PM_j) must have sufficient resources to be able to work properly and serve all its VMs at instant *t*.

Multiobjective optimization

The MOP problems are optimization problems that need solutions with more than one objective function [64]. However, this kind of problem is considered to be an NP hard problem because it contains a set of conflicting objectives. In general, the MOP is given as [64]:

$$\min f(x) = [f_1(x), \ldots, f_M(x)], \quad x = [x_1, \ldots x_n] \in \Omega, \tag{14}$$

$$subject\ to : h_j(x) = 0, \quad j = 1, 2, \ldots, N_h, \tag{15}$$

$$G_l(x) \geq 0, \quad (l = 1, 2, \ldots, N_G), \tag{16}$$

where $f_m(x)$ and Ω are the m-th of the objective function and of the search domain, respectively. The L_j and U_j are lower and upper boundaries of i-th solution, respectively. The inequality and the equality constraints are represented by $G_j(x)$ and $h_j(x)$, respectively.

To find the optimal solution that balances all conflicting objective functions, we use the concept in MOPs known as Pareto optimal dominance. Based on this concept, we consider that the solution x has better results than another solution y if (and only if) it has objective function values better than the other objective values of (y). This can be formulated as

$$\forall j : f_j(x) \leq f_j(y)\ and\ \exists i : f_i(x) < f_i(y). \tag{17}$$

Therefore, the solution x that satisfied Eq. (17) is said to dominate y.

From the previous definition of the dominated solution, there exists a set of solutions called Pareto optimal (PF; also called nondominated solutions) where no solution can dominate them. The set of all PF solutions is represented as follows:

$$Pareto\ set = \{x | x \in \Omega\ and\ x\ is\ Pareto\ optimal\}. \tag{18}$$

In addition, the values of the solutions that belong to the Pareto set in the objective domain is called Pareto front and represented as follows:

$$Pareto\ front = \{f(x) | x \in Pareto\ set\ and\ f(x) \in R^M\}. \tag{19}$$

Salp swarm algorithm

Salp swarm algorithm (SSA) is a new meta-heuristic method, introduced by [21]. SSA emulates the behavior of salps of the family Salpidae. They move like jellyfish, and a high percentage of their weight is water [65]. The working mechanism of SSA starts by producing a random set of N_S solutions (X) with dimension D. The X is split into two sets based on the location of salps in the population. A salp in front of the food

chain will be in the leader group; the rest will be in the followers' group. The leader group x^1 states the solution to the problem. It is updated using the following Eq. [21]:

$$x_j^1(g+1) = \begin{cases} x_j^b(g) + \alpha_1((ub_j - lb_j) \times \alpha_2 + lb_j) & \alpha_3 \leq 0, \\ x_j^b(g) - \alpha_1((ub_j - lb_j) \times \alpha_2 + lb_j) & \alpha_3 > 0, \end{cases} \tag{20}$$

where $x_j^1(g)$ and $x_j^b(g)$ define the leader's position and the food source in the j-th dimension at iteration g, respectively. ub_j and lb_j define the upper and lower bounds of dimension j, respectively. α_2 and α_3 represent random values in [0, 1] that maintain the search space [21]. To give balance to the exploitation and exploration phases, the α_1 is updated at each iteration g depending on the maximum number of iterations g_{max} as follows [21]:

$$\alpha_1 = 2e^{-\left(\frac{4g}{gmax}\right)^2}, \tag{21}$$

Then, the position of the followers' solution x^i is updated using Eq. (22) [21]:

$$x_j^i(g+1) = \frac{1}{2}(x_j^i(g) + x_j^{i-1}(g)), \quad i = 2, \ldots, N_S \tag{22}$$

Algorithm 1 shows all steps of the SSA [21].

Algorithm 1 steps of SSA

```
Produce a random set of N_S solutions X.
while (g < g_max) do
    For each solution compute its objective functions.
    Define the leader (x^b).
    Calculate r_1 with Eq. (21)
    for each solution x_i, i = 1, ..., N_S do
        if (i == 1) then
            Use Eq. (20) to update the leader's position or
        else
            Use Eq. (22) to update the followers' position.
        end if
    end for
end while
Output the leader position (x^b).
```

Sine-cosine algorithm

The sine-cosine algorithm (SCA), presented by [22], is based on the mathematical model of the sine and cosine functions to improve its population and obtain the solution for the problem. It switches between these models to achieve the best result.

The working mechanism of SCA begins by randomly producing an initial set of N_S solutions X with D dimensions. Afterward, for each solution, it calculates the fitness values. The solution with the best fitness value (f_b) is considered as the best solution (x_b). The x_b and the parameters $\beta_i, i = 1, 2, 3, 4$ are used to update the solutions as in Eq. (25). SCA repeats these steps until the stop condition is met, shown in Algorithm 2.

SCA updates its population using the sine function as follows [22]:

$$x_i(g+1) = x_i(g) + \beta_1 \times sin(\beta_2) \times |\beta_3 x_b(g) - x_i(g)| \tag{23}$$

and updates its population using the cosine function as follows [22]:

$$x_i(g+1) = x_i(g) + \beta_1 \times cos(\beta_2) \times |\beta_3 x_b(g) - x_i(g)| \tag{24}$$

The SCA uses a parameter β_4 to switch between these equations as shown in the following [22]:

$$x_i(g+1) = \begin{cases} x_i(g) + \beta_1 \times sin(\beta_2) \times |\beta_3 x_b(g) - x_i(g)| & if\ \beta_4 < 0.5 \\ x_i(g) + \beta_1 \times cos(\beta_2) \times |\beta_3 x_b(g) - x_i(g)| & if\ \beta_4 \geq 0.5 \end{cases} \tag{25}$$

where $|.|$, $x_b(g)$, and $x_i(g)$ define the absolute value, the target and the current solutions at iteration g, respectively. The $\beta_i, i = 1, 2, 3, 4$ represent random variables of [0, 1] [22]. β_2 defines the path of the next solution's movement either toward or away from x_b. β_3 represents a random number that adds weight to x_b to check whether it stochastically preserves (when $\beta_3 > 1$) or ($\beta_3 < 1$) the influence of desalination upon finding the domain. β_1 is applied to switch between exploration and exploitation in order to define the optimal part for the next solution. This part may be higher than the upper bound or lower than the lower bound, Thus, it is updated as follows [22]:

$$\beta_1 = \sigma - g\frac{\sigma}{g_{max}}, \tag{26}$$

where g, σ, and g_{max} define the current iteration, a constant, and the iterations number, respectively.

Algorithm 2 The steps of SCA

1: Produce a set of solutions X.
2: **while** $(g < g_{max})$ **do**
3: Compute the fitness value f for each solution.
4: Define x_b that has the best fitness value f_b.
5: Update $\beta_1, \beta_2, \beta_3$, and β_4 values.
6: Use Eq. (25) to update X.
7: **end while**
8: Output x_b.

Proposed MOSSASCA for VMP method

The section discusses the main steps of the multiobjective VM placement method that combines the improvement of the behavior of the SSA using the SCA. Therefore, the proposed method is called MOSSASCA. The goal of the proposed method is to search for the optimal set of approximate Pareto front (*PF*), which represents the optimal solutions for minimizing SLAV, reducing power consumption and maximizing time before a host shutdown [as shown in Eqs. (5)–(9)].

In general, the proposed MOSSASCA begins by receiving the parameters (i.e., the number of VMs (N_V), the number of hosts (N_P), the number of solutions (N_S), and the max number of iterations. Afterwards, the next steps are to generate a random population that contains a set of solutions, each of which represents the index of VM. Subsequently, the performance of each solution is measured by computing objective functions, and the best solution is selected using concepts of dominance. Therefore, each solution is changed and updated by the SSASCA, where SCA is applied to improve the SSA by working as a local search technique. The nondominated (ND) solutions in the archive are updated by comparing them with the ND solutions from the updated population. This sequence is executed until reaching a maximum number of iterations.

The previously mentioned steps of the proposed MOSSASCA can be classified into three phases: (1) initialization, (2) updating the population using SSASCA, and (3) updating the archive of ND solutions. These phases are discussed in detail in the following subsections.

Initialization

In this phase, the MOSSASCA starts by initializing a set of N_S solutions X where the value of each solution X_i, $(i = 1, 2, \ldots, N_S)$ form the index of the VM and the dimension equal to the total number of VMs (N_V). The solution is generated using the following equation:

$$X_{ij} = floor(L_P + rand \times (N_P - L_P)), \quad j = 1, \ldots, N_V \tag{27}$$

where N_P represents the number of PMs, L_P is the minimum number of PMs (set to one), and *floor* is a function used to convert the real number to an integer value. To clarify this definition, we consider the $N_P = 5$, and $N_V = 8$, then $X_i = [2\ 5\ 3\ 3\ 3\ 1\ 2\ 4]$. This representation means that the first and seventh VMs are allocated to the second PM and the second VM is allocated to the fifth PM. The third to fifth VMs are allocated to the third PM, and the sixth and eighth VMs are allocated at the first and fourth PMs, respectively.

The next step is that the objective function is calculated for each solution using the following equation:

$$Min\ \ F = [f^{Energy}; f^{SLAV}; f^{Meantime}]^T, \tag{28}$$

where f^{SLAV},f^{Energy}, and f^{MTBHS} are defined in Eqs. (5)–(9). Thereafter, the ND solutions are allocated and are stored in archive *AR*.

Algorithm 3 Initialization Stage

```
Put t = 0, archive AR = []
for i = 1 to N do
    Generate X_ij using Equation (27).
    Compute the objective function for x_i using Equation (28).
end for
Update the ND solutions in AR.
```

Update the population using SSASCA

In this phase, the solutions X are updated by using the hybrid SSA and SCA. The aim of using the SCA algorithm is to give the SSA the capability of skipping the local point. The hybrid SSASCA algorithm starts by determining the best solution. However, this process is different from the single-objective optimization, where each solution has only one function. That in MOP has more than two objective values. Therefore, the leader selection mechanism (LSM) is applied to the nondominated solutions in archive *AR* (see [66] for more details) to select the best solution (X^b). The LSM uses the roulette-wheel method to choose X^b based on the probability ($Prob_{Sel}$), which is computed as follows:

$$Prob_{Sel} = C \times Nseg_i, \tag{29}$$

where $Nseg_i$ represents the number Pareto optimal solutions of the crowded segment i, while $C > 1$ is a constant number.

Therefore, to switch between SSA and SCA, we compute the probability (*Pr*) of the power fitness function as follows:

$$Pr_i = \frac{f_1}{\sum_{i=1}^{N_S} f_1}. \tag{30}$$

If the current solution has $Pr_i \geq \eta$ ($\eta \in [0, 1]$), then the SSA is used to update X_i; otherwise, the SCA is used to update X_i. The next step is to convert the updated solution into an integer solution using the *floor* function (i.e., $X_i = floor(X_i)$); then, the objective functions are calculated for each solution.

Algorithm 4 Update stage

```
Determine x_b using the leader selection method (LSM) using Equation (29).
for i = 1 to N do
    Calculate the Pr_i value of X_i.
    if (Pr_i ≤ η) then
        Use Equation (25) to update X_i,
    else
        if i ≤ N/2 then
            Use Equation (20) to update X_i (leader),
        else
            Use Equation (22) to update X_i (follower).
        end if
    end if
end for
```

Update the archive

This stage begins by combining the current population with the archive *AR*, followed by the determination of ND solutions. This process is critical to improve convergence to the true *PF* and preserve the population's diversity by using the density estimation information [67–70].

In general, the archive *AR* is updated by comparing its ND solutions with each solution in the population (i.e., $X_i \in X$) using the Pareto dominance concepts. Here, the solution X_i will swap any solution X_A in *AR* when X_i dominates X_A, and if there is a set of solutions in *AR* is dominated by X_i, then this set will be removed from *AR*, and X_i will be added to *AR*. However, if X_i must be added to *AR* and the size of *AR* is full, then the crowded segment method is used. The segmentation of the objective space is rearranged by using the grid mechanism, and the most crowded segment is determined to delete the solution from *AR*. Thereafter, the X_i will be combined with the least crowded segment to improve the *PF*'s diversity. In addition, solution X_i is not appended to *AR* when any solution dominates it (i.e., X_i) [71]. The final step in this phase is to enhance the population *X* by selecting the best N_S solutions from *AR*.

Algorithm 5 Stage of Updating Archive *AR*

```
In: Updated X,
Add X to the ND solution in AR.
Determine the ND solutions of AR.
Check the size of AR and choose only the best N_S ND solutions.
Return: Archive AR.
```

Full description of MOSSASCA method

The remaining parts of the proposed MOSSASCA method for VMP are presented in Algorithm 6. Here, the current generation is represented by g and the maximum number of iterations is given by g_{max}. The proposed MOSSASCA starts by determining the input to the initialization phase (as described in Algorithm 3) and receiving the output (i.e., population X and archive AR) from it. The next step is to update X based on the hybrid SSASCA algorithm (as described in 4). Then, Algorithm 5 is used to find the NDS and update AR. The steps of two algorithms (i.e., Algorithm 4, Algorithm 5) are repeated until the terminal conditions are satisfied.

Algorithm 6 Multiobjective salp swarm algorithm based on sine-cosine algorithm (MOSSASCA).

1: Input: number of VMs and PMs, population size N_S, and maximum number of generations g_{max}
2: Output: nondominated solutions saved in archive AR.
3: Use Algorithm 3 to generate the population X.
4: **repeat**
5: Use Algorithm 4 to update the solutions.
6: Use Algorithm 5 to update the nondominated solutions saved in archive AR and update the population X.
7: $g = g + 1$
8: **until** $g <= g_{max}$.

Complexity of MOSSASCA method

The complexity of the MOSSASCA method depends on several items, for example, the dimension of the tested problem, the number of solutions N_S, the number of objectives M, the maximum number of iterations t_{max}. Therefore, the complexity of MOSSASCA is given by

$$\begin{aligned} O(MOSSASCA) = O(t_{max}(N_V \times (N_S \times N_{prob1} + N_S \times N_{prob2}) \\ + cof \times N_S + M \times N_S^2)), \end{aligned} \tag{31}$$

where N_{prob1} and N_{prob1} represents the number of solutions updated using the SSA and SCA respectively. *cof* is the cost of objective function.

Results and discussion

A set of experiments are performed to assess the performance of the MOSSASCA as a multiobjective VMP method to find the optimal solution to the VM allocation problem. In addition, the results of MOSSASCA are compared with those of other approaches, such as NSGAII [72], MOEA-D [73], MOPSO [74], and MOSCA [75].

Environment description

In this section, the description of the environment is explained, where the CloudSim toolkit simulation framework is used to simulate the cloud computing environments [76]. The experimental setup used in this study simulates the real configurations of cloud computing infrastructure, which includes 160 IBM Server x3250 and Intel Xeon 3480 CPU, 160 IBM Server x3250 and Intel Xeon 3470 CPU model, 160 HP ProLiant ML110

Table 1 The Parameters of VM simulation

Parameter	Value
VM types	2, 4
VM RAM	870, 1740 MB
VM bandwidth	100 to 200 Mbit/s
VM MIPS	2500, 100
Number of VMs	50, 100, 150, 200
VM PES	1,1

Table 2 Properties of the workload

Data	Standard deviation (SD) (%)	Mean (%)	The number of VMs
03/03/2011	17.09	12.31	1052
06/03/2011	16.83	11.44	898
09/03/2011	15.57	10.70	1061
22/03/2011	12.78	9.26	1516
25/03/2011	14.14	10.56	1078
03/04/2011	16.55	12.39	1463
09/04/2011	15.09	11.12	1358
11/04/2011	15.07	11.56	1233
12/04/2011	15.15	11.54	1054
20/04/2011	15.21	10.43	1033

G3, 160 HP ProLiant ML110 G4, and 160 HP ProLiant ML110 G5. Table 1 shows a set of parameters used in the experiments for VM simulations.

In addition, the workload data are extracted from the project of CoMon data [77, 78]. Statistical values of the utilization (shown in %) of data (daily) are given in Table 2.

The performance of the MOSSASCA is compared with other algorithms, such as NSGAII, MOEAD, MOPSO, and MOSCA. To evaluate the performance of these methods, we use different numbers of VMs (25, 75, and 100) and hosts (50, 150, and 200), where 50 VMs are assigned to 25 hosts, 150 VMs are assigned to 75 hosts, and 200 VMs are assigned to 100 hosts.

All algorithms are implemented using jMetal java framework, which is installed on Windows 10 (64-bit).

Performance measures

The proposed method is evaluated on the basis of MOP performance measurements, including hypervolume (HV), spread (SP), generational distance (GD), epsilon (EPS), and inverted generational distance (IGD) [79].

1. HV: It is applied to evaluate the nearness and variety of *PF* by computing the area size dominated by its solutions. If the HV of $X > B$, then X Pareto optimal converges to be greater than B.

$$HV = volume\left(\bigcup_{i=1}^{|PF|} a_i\right) \tag{32}$$

2. SP: It is used to evaluate the extent of spread achieved between the solutions and compute the nonuniformity in the distribution of solutions, and it is defined as

$$Spread = \frac{\sum_{i=1}^{M} d_i^e + \sum_{i=1}^{|Q|} |d_i - \bar{d}|}{\sum_{i=1}^{M} d_i^E + |Q|\bar{d}}, \tag{33}$$

where d_i represents the Euclidean distances between the set of solutions that have the mean value $\bar{d}$, and where d_i^e represents the distance between the extreme solutions and PF^*. M and $|Q|$ are the total number of objective functions and the number of solutions in PF^*, respectively. The smaller *Spread* value indicates that the algorithm is better than others.

3. GD: GD defines the average distance between the true Pareto front and the estimated front. Equation (34) defines this measure:

$$GD(fp, fp^*) = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}. \tag{34}$$

In Eq. (34), $n = |fp|$ defines the number of elements of the ND solutions, and d_i defines the Euclidean between the closest front's solution of Pareto optimal in the objective space and reference solution (*RP*).

4. IGD: It is applied to compute the nearness between the reference optimal solution (*RP*) and the archived Pareto solution. Equation (35) defines this measure:

$$IGD = \frac{\sum_{r \in RP} d}{|RP|}, \quad d = ||a - r||_2, \forall a \in PF, r \forall RP. \tag{35}$$

The smaller IGD value indicates better performance.

5. EPS: It is used to find the minimum value where the approximation *PF* is mapped to the objective space that dominates the true *PF*.

$$EPS = \max_{r \in PF_t} \min_{a \in A} \max_{i=1,2,\ldots,M} (a_i - r_i), \tag{36}$$

where r and a are the solutions in the true *PF* and the approximate *PF*, respectively. The algorithm with the smallest *EPS* is considered to be the better one.

Experimental results analysis

Comparison with other algorithms

The comparison results between the MOSSASCA and the other algorithms in order to determine the optimal solution for the VMP in cloud computing are given in Table 3 and Figs. 2, 3, 4. The results show that the proposed MOSSASCA method performs better than the other methods. Specifically, the proposed method has lower minimum and maximum power consumption in all tested problems. The standard deviation of the

proposed method is smaller than all other methods except the MOEAD in the VM200-PM100 problem.

In addition, the minimum value of SLAV is achieved using the proposed method and by using the MOSCA, MOPSOA, and MOEAD algorithms in problems VM50-PM25 and VM150-PM75, but the proposed method achieves the best value in the VM200-PM100 problem. The proposed method has better maximum SLAV value than the other methods. However, the NSGA-II algorithm has the best standard deviation in problem

Table 3 The comparison of results among the algorithms in each tested VMP problem

	MOSSASCA	MOSCA	MOPSO	MOEAD	NSGAII
Power consumption					
VM50-PM25					
Min	4.22	4.24	4.77	4.77	4.41
Max	6.25	8.25	6.78	6.78	7.68
Std	0.55	0.97	0.62	0.66	0.73
VM150-PM75					
Min	11.06	11.98	14.25	13.27	12.20
Max	17.73	14.25	23.85	24.58	19.91
STD	1.75	0.66	1.96	2.93	2.46
VM200-PM100					
Min	13.03	14.24	14.78	15.77	13.37
Max	18.92	19.77	22.04	19.77	19.49
STD	1.73	1.99	2.26	1.35	1.82
SLAV					
VM50-PM25					
Min	10.00	10.00	10.00	10.00	10.30
Max	14.94	16.10	18.50	17.00	14.90
STD	1.63	1.75	2.58	2.24	1.50
VM150-PM75					
Min	9.67	9.67	9.67	9.67	10.21
Max	13.29	14.16	14.07	14.07	13.62
STD	1.60	0.97	1.76	1.98	1.25
VM200-PM100					
Min	9.60	10.00	10.00	10.00	10.00
Max	12.13	13.33	14.00	14.00	15.00
STD	1.15	1.00	1.72	1.45	1.96
MTBHS					
VM50-PM25					
Min	8845.46	7397.86	8538.01	8547.97	7942.10
Max	10860.11	10773.76	8739.08	8724.04	8566.20
STD	868.70	856.34	45.33	64.88	192.77
VM150-PM75					
Min	7630.12	7613.31	8513.32	7570.60	7524.73
Max	9229.50	9165.59	8723.81	8441.30	8529.89
STD	305.74	603.68	95.95	314.67	337.52
VM200-PM100					
Min	8236.32	7397.86	8111.96	7614.16	7516.34
Max	8995.71	8773.76	8731.38	8715.56	8756.34
STD	205.42	472.11	244.68	275.83	401.91

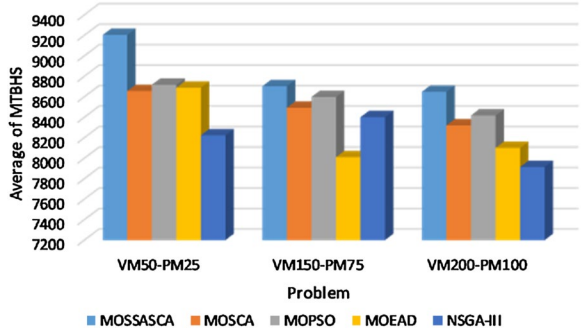

Fig. 2 Average MTBHS for each method in each VMP problem

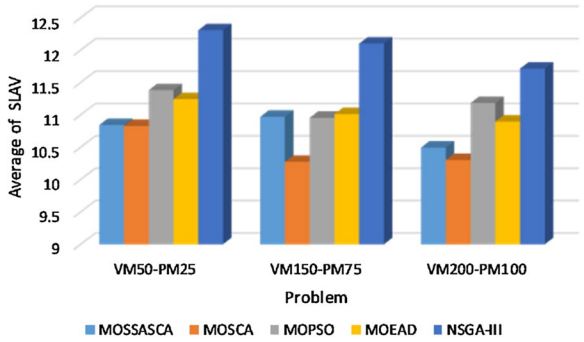

Fig. 3 Average SLAV for each method in each VMP problem

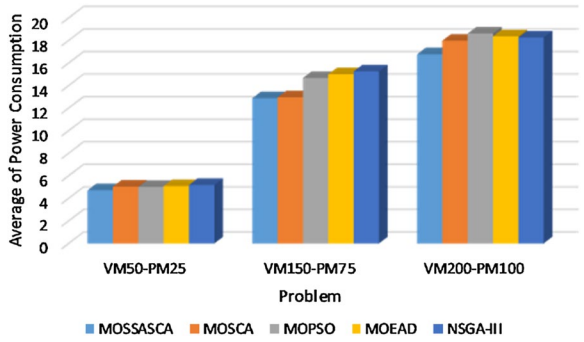

Fig. 4 Average power consumption for each method in each VMP problem

VM50-PM25, and the MOSCA method has better value in problems VM150-PM75 and VM200-PM100.

The best maximum and minimum MTBHS values are achieved by the MOSSASCA in all the tested problems, but the best standard deviation is achieved by MOPSO in problems VM50-PM25 and VM150-PM75. Furthermore, the proposed method has the smaller standard deviation value in the VM200-PM100 problem.

Figures 2, 3, and 4 show the average time, SLAV, and energy, respectively. The figures show that the proposed method provides lower average power consumption and

SLAV values than the other methods in each tested problem. The methods have similar performance in VM50-PM25, but MOPOS and NSGA-II have the worst value in VM200-PM100 and VM150-PM75, respectively. Power consumption is increased as the numbers of PMs and VMs are increased. The proposed method and MOSCA nearly have the same SLAV value in VM50-PM25, but the proposed method produces better results than all other methods in the other two problems. Among the methods, NSGA-II has the worst results in three problems, whereas MOSCA, MOPSO, and MOEAD have nearly the same average value of SLAV in the VM150-PM75 problem. However, MOSCA has better results than MOPSO and MOEAD in the VM200-PM100 problem. Moreover, the worst MTBHS value is achieved by NSGA-II in problems VM50-PM25 and VM200-PM100, and MOEAD in VM150-VM75. The MOSSASCA shows best value among all tested algorithms in each tested problem. These results suggest that MOSSASCA provides maximum utilization of each resource.

Performance evaluation based on MOP indicators

To investigate the quality of PFs obtained by each method we evaluate a set of MOP indicators given in Table 4 for each tested problem. From this table, it can be noted that the MOSSASCA performs better than the other methods in approximating PFs. For example, in terms of EP, HV, and IGD, the MOSSASCA ranks first of in all tested problems. However, in terms of spread, MOSCA ranks first in testing problems in the VM50-PM25 and VM150-PM75, followed by the proposed MOSSASCA method, which has a smaller GD value in the third problem (i.e., VM200-PM100). MOSSASCA has a smaller GD value only in problem VM50-PM25, but MOPSO has a smaller GD value in two other problems. In addition, we can conclude from the table that MOEAD and NSGA-II generally have the worst performance in the tested problems.

From all the previous results, it can be concluded that the MOSSASCA outperforms the other methods based on the MOP indicators.

Influence the parameters of proposed method

In this experiment, we investigate the influence of different parameters on the performance of the MOSSASCA. Table 5 illustrates the results of the proposed MOSSASCA based on different values of $a = [1,\ 4]$, N_S=[50, 200], and g_{max}=[50, 200]. The proposed method at $a = 1.5$ is better than that at $a = 0.5$, indicating that the performance of the MOSSASCA increases with increasing a. In addition, population size affects the performance of MOSSASCA. Specifically, the MOSSASCA performs better at pop=200 than at pop=50 and pop=100 as in Table 4. Moreover, similarly, the performance of MOSSASCA is affected by the number of iterations, and the results showed that the MOSSASCA achieves high performance at iter=200.

Influences of VMs and PMs on the proposed method

The effect of variants number of VMs and PMs on the performance of the MOSSASCA method is tested in this section. Table 6 illustrates the results of different VMs and PMs, in which we note that the performance of the MOSSASCA in terms of EPS has better value in problem VM300-PM600; however, this is not largely different than the

Table 4 Comparison results based on MOP indicators

	MOSSASCA	MOSCA	NSGAII	MOEAD	MOPSO
VM50-PM25					
EPS	1.54E−01	2.77E−01	7.14E−01	6.12E−1	6.12E−01
	2.2E−02	2.5E−02	2.3E−04	0.00	0.0E+00
Spread	1.36E+00	4.10E−01	1.08E+00	1.30E+00	1.05E+00
	2.6E−02	1.3E−01	2.3E−04	6.0E−02	2.9E−03
GD	4.29E−03	4.73E−02	1.58E−01	3.54E−02	3.48E−02
	1.4E−03	2.2E−02	7.1E−05	1.1E−04	1.1E−06
HV	4.91E−01	4.16E−01	2.31E−01	3.36E−01	3.36E−01
	1.5E−02	1.9E−02	1.7E−04	1.3E−04	2.6E−08
IGD	1.08E−03	2.77E−03	1.42E−02	1.33E−02	1.35E−02
	7.8E−05	4.6E−04	1.6E−06	1.6E−05	2.1E−05
VM150-PM75					
EPS	1.16E−01	1.95E−01	4.43E−01	3.40E−01	2.40E−01
	8.3E−02	5.1E−02	9.2E−05	2.2E−04	0.0E+00
Spread	1.50E+00	7.79E−01	1.01E+00	1.27E+00	9.08E−01
	9.1E−02	8.2E−02	2.0E−02	5.7E−02	5.2E−02
GD	1.93E−03	1.31E−02	1.22E−02	7.15E−03	9.99E−05
	3.1E−04	7.3E−03	1.9E−04	3.8E−04	1.4E−04
HV	6.79E−01	6.15E−01	4.53E−01	5.40E−01	6.33E−01
	2.8E−03	4.8E−03	8.1E−05	9.9E−03	1.1E−07
I GD	1.22E−03	2.46E−03	8.73E−03	7.44E−03	7.79E−03
	8.7E−04	7.2E−05	3.9E−05	1.6E−04	1.0E−04
VM200-PM100					
EPS	1.27E−01	4.58E−01	3.97E−01	3.46E−01	2.91E−01
	3.8E−03	5.8E−04	1.5E−04	9.4E−03	6.8E−02
Spread	7.35E−01	1.17E+00	1.07E+00	1.16E+00	1.14E+00
	2.6E−02	1.2E−04	2.0E−02	9.9E−02	5.0E−03
GD	3.90E−02	4.61E−02	6.65E−03	3.50E−03	1.08E−04
	2.8E−03	1.30E−03	3.6E−04	1.6E−03	9.8E−05
HV	6.10E−01	6.68E−01	4.76E−01	5.34E−01	7.00E−01
	1.1E−03	8.1E−03	1.4E−04	1.7E−02	5.1E−02
IGD	5.55E−03	7.1E−03	1.18E−02	1.10E−02	1.10E−02
	1.2E−04	2.1E−03	4.1E−05	1.90E−04	1.2E−04

result with VM200-PM500. In VM400-PM1000, the EPS of the MOSSASCA is the largest among the three problems. Based on the spread measure, the quality of the MOSSASCA in the three problems is nearly the same; however, the best value is achieved in VM200-PM500. The same observation is found for the values of GD, where the value of the MOSSASCA at VM200-PM500 is better than its value in the other problems. In addition, the HV value of the MOSSASCA at VM400-PM100 is better than in other problems. In addition, it can be observed that the best IGD value is achieved when MOSSASCA is used to solve the VM300-PM600. From these experimental series, it can be observed that the performance of MOSSASCA, in terms of MOP indicators, is not largely notable when solving the VM problem with VMs of 200–400 and PMs of 500–1000.

Table 5 Results of MOP indicators under different parameter values in the proposed method

	$a = 1.5$		$a = 0.5$		
EPS	1.40E−01	1.6E−01	EPS	1.63E−01	1.2E−01
Spread	8.79E−01	2.5E−03	Spread	9.51E−01	1.5E−01
GD	4.79E−03	7.9E−03	GD	5.18E−03	5.9E−03
HV	6.62E−01	3.0E−03	HV	6.16E−01	5.3E−03
IGD	8.38E−03	4.7E−03	IGD	2.04E−02	2.3E−02
	$N_S = 200$		$N_S = 50$		
EPS	3.47E−02	2.9E−02	EPS	1.71E−01	7.1E−02
Spread	7.89E−01	3.0E−02	Spread	9.37E−01	6.5E−02
GD	1.32E−03	1.6E−03	GD	6.08E−03	8.6E−03
HV	6.83E−01	2.3E−03	HV	6.04E−01	6.5E−02
IGD	1.12E−03	4.2E−03	IGD	9.37E−03	2.9E−03
	g	$max = 50$		g	$max = 200$
EPS	9.47E−01	5.7E−02	EPS	8.36E−02	5.6E−02
Spread	9.14E−01	8.3E−02	Spread	8.44E−01	1.7E−02
GD	1.54E−02	5.3E−03	GD	1.18E−03	4.7E−03
HV	6.37E−01	7.3E−02	HV	7.45E−01	5.0E−03
IGD	9.66E−03	5.7E−03	IGD	1.16E−03	4.5E−03

Table 6 Results of changing the number of VMs and PMs

	Power consumption	SLVA	MTBHS	EPS	Spread	GD	HV	IGD
VM500-PM200	26.03	10.48	8371.35	7.43E−02	7.20E−01	3.01E−03	4.92E−01	2.61E−03
VM600-PM300	36.71	10.81	7318.01	4.56E−02	9.17E−01	4.85E−03	6.91E−01	1.39E−03
VM1000-PM400	59.24	11.44	6038.51	1.44E−01	7.23E−01	9.79E−03	2.17E−01	2.26E−03

From all the previous experimental results, it can be noted that the performance of MOSSASCA is better than the other methods in terms of the three objectives functions and the performance measurements. This performance results from the high skill of the SSA in exploration of the search domain and in using the operators of the SCA algorithm to improve the exploitation ability of the SSA through working as a local search method. Additionally, this leads to finding the nondominated solution that best balances among the three objectives of solving the VMP problem in cloud computing. Moreover, it can be noted that the performance of the other multiobjective VMP methods was less than that of the proposed method in this study and that the quality of the algorithms was not fixed when changing the problems. This outcome is because each algorithm has a different ability regarding either the exploration or exploitation of the search domain. However, there are some limitations for MOSSASCA, such as the fact that its computational time will require additional improvement to make it more suitable for real-time VMP problems.

Conclusion

This paper proposes an alternative MOP method for finding the optimal solution to the VM consolidation problem. The main objectives of the proposed MOSSASCA are to maximize MTBHS, to reduce power consumption, and to minimize SLAV. MOSSASCA is proposed to find solutions that can minimize conflict between the three objectives. In MOSSASCA, the SCA is applied as a local search approach to enhance the performance of traditional SSAs by preventing them from getting stuck in a local optimal solution and increasing convergence speed. To assess the performance of the proposed method, we perform a set of experiments using different numbers of VMs and physical machines. The results of MOSSASCA are compared with that of well-known MOP methods, including NSGA-II, MOPSO, and MOSCA. The experimental results show that MOSSASCA is better than others on the basis of MOP indicators and in achieving the three objectives. Here, the proposed method achieves the best results in three MOP indicators, namely, EPS, HV, and IGD. The MOSCA and MOPSO methods are better according to the value of spread and GD, respectively. Compared with the other methods, MOSSASCA exhibits a better ability to reduce power consumption and SLAV while increasing MTBHS.

Given the superiority of the proposed MOSSASCA method, it can be extended to solve more than the three objectives in VM placement in cloud computing. It can also be applied to different fields, for example, feature selection, image segmentation and job scheduling.

Authors' contributions

All authors contributed equally to this study. SSA collected and prepared the data; MAE and AAE implemented the proposed method; and SL and the other three authors wrote the main text and discussed the results. All authors read and approved the final manuscript.

Author details

[1] School of Computer Science & Technology, Huazhong University of Science and Technology, Wuhan 430074, China. [2] Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen 518063, China. [3] Department of Mathematics, Faculty of Science, Zagazig University, Zagazig, Egypt. [4] University of Bisha, Bisha, Kingdom of Saudi Arabia. [5] Department of Computer, Damietta University, Damietta, Egypt.

Acknowledgements

This work is supported by the Science and Technology Program of Shenzhen of China under Grant Nos. JCYJ20180306124612893, JCYJ20170818160208570 and JCYJ20170307160458368.

Competing interests

The authors declare that they have no competing interests.

Data availability statement

The data used to support the findings of this study are available from the corresponding author upon request.

Funding

Not applicable.

Publisher's Note

Received: 20 October 2018 Accepted: 25 March 2019
Published online: 09 April 2019

References

1. Xu M, Tian W, Buyya R (2017) A survey on load balancing algorithms for virtual machines placement in cloud computing. Concurrency Comput Pract Exp 29(12):e4123
2. Zheng Q, Li Jia, Dong Bo, Li Rui, Shah Nazaraf, Tian Feng (2016) Multi-objective optimization algorithm based on bbo for virtual machine consolidation problem. In: IEEE international conference on parallel and distributed systems. Piscataway, IEEE, pp 414–421

3. Tang M, Pan S (2015) A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers. Neural Process Lett 41(2):211–221
4. Fu X, Zhao Q, Wang J, Zhang L, Qiao L (2018) Energy-aware vm initial placement strategy based on bpso in cloud computing. Sci Program 2018:9471356
5. Bajo J, De la PF, Corchado JM, Rodríguez S (2016) A low-level resource allocation in an agent-based cloud computing platform. Appl Softw Comput 48:716–728
6. Cao Z, Lin J, Wan C, Song Y, Zhang Y, Wang X (2017) Optimal cloud computing resource allocation for demand side management in smart grid. IEEE Trans Smart Grid 8(4):1943–1955
7. Lin Y-K, Chong CS (2017) Fast ga-based project scheduling for computing resources allocation in a cloud manufacturing system. J Intell Manuf 28(5):1189–1201
8. LopezPires F, Baran B (2015) Virtual machine placement literature review. Comput Sci
9. El AM Abd, Ewees AA, Hassanien AE (2016) Hybrid swarms optimization based image segmentation. In: Hybrid soft computing for image segmentation, Springer, Berlin, pp 1–21
10. Ewees AA, El Aziz MA, Hassanien AE (2017) Chaotic multi-verse optimizer-based feature selection. Neur Comput Appl. https://doi.org/10.1007/s00521-017-3131-4
11. Ibrahim RA, Oliva D, Ewees AA, Lu S (2017) Feature selection based on improved runner-root algorithm using chaotic singer map and opposition-based learning. In: International conference on neural information processing, Springer, Berlin, pp 156–166
12. Oliva D, Ewees AA, El Aziz MA, Hassanien AE, Peréz-Cisneros M (2017) A chaotic improved artificial bee colony for parameter estimation of photovoltaic cells. Energies 10(7):865
13. Kaveh A, Dadras A (2017) A novel meta-heuristic optimization algorithm: thermal exchange optimization. Adv Eng Softw 110:69–84
14. Kaveh A, Dadras A (2018) Structural damage identification using an enhanced thermal exchange optimization algorithm. Eng Optimization 50(3):430–451
15. Fausto F, Cuevas E, Valdivia A, González A (2017) A global optimization algorithm inspired in the behavior of selfish herds. Biosystems 160:39–55
16. Jiang S, Zhou Y, Wang D, Zhang S (2018) Elite opposition-based selfish herd optimizer. In: International conference on intelligent information processing, Springer, Berlin, pp 89–98
17. Kiran MS (2015) Tsa: TreE—seed algorithm for continuous optimization. Expert Syst Appl 42(19):6686–6698
18. Cinar AC, Iscan H, Kiran MS (2018) Tree-seed algorithm for large-scale binary optimization. KnE Soc Sci 3(1):48–64
19. El-Fergany AA, Hasanien HM (2018) Tree-seed algorithm for solving optimal power flow problem in large-scale power systems incorporating validations and comparisons. Appl Softw Comput 64:307–316
20. Cinar AC, Kiran MS (2018) Similarity and logic gate-based tree-seed algorithms for binary optimization. Comput Ind Eng 115:631–646
21. Mirjalili S, Gandomi AH, Mirjalili SZ, Saremi S, Faris H, Mirjalili SM (2017) Salpswarm algorithm: a bio-inspired optimizer for engineering designproblems. Adv Eng Softw 114:163–191
22. Mirjalili S (2016) Sca: a sine cosine algorithm for solving optimization problems. Knowl Based Syst 96:120–133
23. El-Fergany AA (2018) Extracting optimal parameters of pem fuel cells using salp swarm optimizer. Renew Energy 119:641–648
24. Ibrahim RA, Ewees AA, Oliva D, Elaziz MA, Lu S (2018) Improved salp swarm algorithm based on particle swarm optimization for feature selection. J Ambient Intell Hum Comput. https://doi.org/10.1007/s12652-018-1031-9
25. Elaziz Mohamed EA, Ewees AA, Oliva D, Duan P, Xiong S (2017) A hybrid method of sine cosine algorithm and differential evolution for feature selection. In: International conference on neural information processing, Springer, Berlin, pp 145–155
26. Teng F, Lei Y, Li T, Deng D, Magouls F (2017) Energy efficiency of vm consolidation in iaas clouds. J Supercomput 73(2):782–809
27. Zheng Q, Li R, Li X, Shah N, Zhang J, Tian F, Chao K-M, Li J (2016) Virtual machine consolidated placement based on multi-objective biogeography-based optimization. Future Gener Comput Syst 54:95–122
28. Hosseinimotlagh S, Khunjush F, Samadzadeh R (2015) Seats: smart energy-aware task scheduling in real-time cloud computing. J Supercomput 71(1):45–66
29. Mann Z (2016) Multicore-aware virtual machine placement in cloud data centers. IEEE Trans Comput 65(11):3357–3369
30. Lawey AQ, El-Gorashi TEH, Elmirghani JMH (2014) Distributed energy efficient clouds over core networks. J Lightwave Technol 32(7):1261–1281
31. Dong J, Jin X, Wang H, Li Y, Zhang P, Cheng S (2013) Energy-saving virtual machine placement in cloud data centers. In IEEE/ACM international symposium on cluster, cloud and grid computing, IEEE, Piscataway, pp 618–624
32. Grant W, Tang M, Tian YC, Li W (2012) Energy-efficient virtual machine placement in data centers by genetic algorithm. Springer, Berlin Heidelberg
33. Gabay M, Zaourar S (2016) Vector bin packing with heterogeneous bins: application to the machine reassignment problem. Ann Oper Res 242(1):161–194
34. Marotta A, Avallone S (2015) A simulated annealing based approach for power efficient virtual machines consolidation. In: 2015 IEEE 8th International conference on cloud computing (CLOUD), IEEE, Piscataway, pp 445–452
35. Farahnakian F, Ashraf A, Pahikkala T, Liljeberg P, Plosila J, Porres I, Tenhunen H (2015) Using ant colony system to consolidate vms for green cloud computing. IEEE Trans Serv Comput 8(2):187–198
36. Baquela EG, Olivera AC (2018) A multi-objective optimization via simulation framework for restructuring traffic networks subject to increases in population. In: Recent developments in metaheuristics, Springer, Berlin, pp 199–218
37. Bandaru S, Ng AHC, Deb K (2017) Data mining methods for knowledge discovery in multi-objective optimization: part a-survey. Expert Syst Appl 70:139–159
38. Zambrano-VC, Nebro AJ, García-Nieto J, Aldana-Montes JF (2017) A multi-objective optimization framework for multiple sequence alignment with metaheuristics. In: International conference on bioinformatics and biomedical engineering, Springer, Berlin, pp 245–256

39. Neto JXV, Junior EJG, Moreno SR, Ayala HV H, Mariani VC, dos Santos CL (2018) Wind turbine blade geometry design based on multi-objective optimization using metaheuristics. Energy 162:645–658
40. El Aziz MA, Ewees AA, Hassanien AE (2018) Multi-objective whale optimization algorithm for content-based image retrieval. Multimedia Tools Appl 77:26135–26172
41. El Aziz MA, Ewees A\A, Hassanien AE, Mudhsh M, Xiong S (2018) Multi-objective whale optimization algorithm for multilevel thresholding segmentation. In: Advances in soft computing and machine learning in image processing, Springer, Berlin, pp 23–39
42. Mirjalili SZ, Mirjalili S, Saremi S, Faris H, Aljarah I (2018) Grasshopper optimization algorithm for multi-objective optimization problems. Appl Intell 48(4):805–820
43. Mirjalili S, Jangir P, Saremi S (2017) Multi-objective ant lion optimizer: a multi-objective optimization algorithm for solving engineering problems. Appl Intell 46(1):79–95
44. Yang X-S, Karamanoglu M, He X (2014) Flower pollination algorithm: a novel approach for multiobjective optimization. Eng Optimization 46(9):1222–1237
45. Sadollah A, Sadollah A, Eskandar H, Kim JH (2015) Water cycle algorithm for solving constrained multi-objective optimization problems. Appl Softw Comput 27:279–298
46. Gao Y, Guan H, Qi Z, Hou Y, Liu L (2013) A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J Comput Syst Sci 79(8):1230–1242
47. Sathish K, Reddy RM (2017) Workflow scheduling in grid computing environment using a hybrid gaaco approach. J Inst Eng 98(1):1–8
48. Wang X, Wang Y, Cui Y (2014) A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing. Future Gener Comput Syst 36(7):91–101
49. Zhang F, Cao J, Li K, Khan SU, Hwang K (2014) Multi-objective scheduling of many tasks in cloud platforms. Future Gener Comput Syst 37:309–320
50. Shieh WY, Pong CC (2013) Energy and transition-aware runtime task scheduling for multicore processors. J Parallel Distributed Comput 73(9):1225–1238
51. Ramezani F, Jie L, Hussain F (2013) Task scheduling optimization in coud computing applying multi-objective particle swarm optimization. Springer, Berlin Heidelberg
52. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
53. Ekinci S, Hekimoglu B (2018) Parameter optimization of power system stabilizer via salp swarm algorithm. In: 2018 5th International conference on electrical and electronic engineering (ICEEE), IEEE, Piscataway, pp 143–147
54. Abbassi R, Abbassi R, Abbassi A, Heidari AA, Mirjalili S (2019) An efficient salp swarm-inspired algorithm for parameters identification of photovoltaic cell models. Energy Convers Manag 179:362–372
55. Sayed GI, Khoriba G, Haggag MH (2018) A novel chaotic salp swarm algorithm for global optimization and feature selection. Appl Intell 48:3462–3481
56. Faris H, Faris H, Mafarja MM, Heidari AA, Aljarah I, AlaM A-Z, Mirjalili S, Fujita H (2018) An efficient binary salp swarm algorithm with crossover scheme for feature selection problems. Knowl Based Syst 154:43–67
57. Aljarah I, Mafarja M, Heidari AA, Faris H, Zhang Y, Mirjalili S (2018) Asynchronous accelerating multi-leader salp chains for feature selection. Appl Softw Comput 71:964–979
58. Ismael SM, Aleem SHE, Abdelaziz AY, Zobaa AF (2018) Practical considerations for optimal conductor reinforcement and hosting capacity enhancement in radial distribution systems. IEEE Access
59. Baygi SMH, KA (2018) A hybrid optimal pid-lqr control of structural system: A case study of salp swarm optimization. In: 2018 3rd conference on swarm intelligence and evolutionary computation (CSIEC), IEEE, Piscataway, pp 1–6
60. Sun Z-X, HR, QB, LB, CG-L (2018) Salp swarm algorithm based on blocks on critical path for reentrant job shop scheduling problems. In: International conference on intelligent computing, Springer, Berlin, pp 638–648
61. Sahlol AT, Ewees AA, Hemdan AM, Hassanien AE (2016) Training feedforward neural networks using sine-cosine algorithm to improve the prediction of liver enzymes on fish farmed on nano-selenite. In: Computer engineering conference (ICENCO), 2016 12th international, IEEE, Piscataway, pp 35–40
62. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst 28(5):755–768
63. López-Pires F, Barán B (2017) Many-objective virtual machine placement. J Grid Comput 15(2):161–176
64. Zhang X, Tian Y, Cheng R, Jin Y (2015) An efficient approach to nondominated sorting for evolutionary multiobjective optimization. IEEE Trans Evol Comput 19(2):201–213
65. Henschke N, Everett JD, Richardson AJ, Suthers IM (2016) Rethinking the role of salps in the ocean. Trends Ecol Evol 31(9):720–733
66. Mirjalili SM, dos Coelho LS, Mirjalili S, Saremi S (2106) Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization. Expert Syst Appl 47:106–119
67. Deb K, Agrawal S, Pratap A, Meyarivan T (2000) A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-II. In: International conference on parallel problem solving from nature, Springer, Berlin, pp 849–858
68. Deb K, Beyer H-G (2001) Self-adaptive genetic algorithms with simulated binary crossover. Evol Comput 9(2):197–221
69. Zhang Q, Zhou A, Zhao S, Suganthan PN, Liu W, Tiwari S (2008) Multiobjective optimization test instances for the cec 2009 special session and competition. University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, Technical Report, 264
70. Bosman PAN, Thierens D (2003) The balance between proximity and diversity in multiobjective evolutionary algorithms. IEEE Trans Evol Comput 7(2):174–188
71. Agarwal S, Deb K, Pratap A, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput 6(2):182–197
72. Deb K, Pratap A, Agarwal S, Meyarivan TAMT (2002) A fast and elitist multiobjective genetic algorithm: Nsga-II. IEEE Trans Evol Comput 6(2):182–197

73. Zhang Q, Li H (2007) Moea/d: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731
74. Elsedimy EI, Rashad MZ, Darwish MG (2017) Multi-objective optimization approach for virtual machine placement based on particle swarm optimization in cloud data centers. J Comput Theor Nanosci 14(10):5145–5150
75. Tawhid MA, Savsani V (2017) Multi-objective sine-cosine algorithm (mo-sca) for multi-objective engineering design problems. Neur Comput Appl. https://doi.org/10.1007/s00521-017-3049-x
76. Buyya R, Ranjan R, Calheiros RN (2009) Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In: International conference on high Performance computing and simulation, IEEE, Piscataway, pp 1–11
77. Beloglazov A, Buyya R (2012) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency Comput Pract Exp 24(13):1397–1420
78. Pai VS, Park KS (2006) Comon: a mostly-scalable monitoring system for planetlab. ACM SIGOPS Operating Syst Rev 40(1):65–74
79. Ravber M, Mernik M, Črepinšek M (2017) The impact of quality indicators on the rating of multi-objective evolutionary algorithms. Appl Softw Comput 55:265–275