SN Computer Science (2024) 5:466
https://doi.org/10.1007/s42979-024-02832-2

ORIGINAL RESEARCH

Efficient Virtual Machine Placement Strategy Based on Enhanced Genetic Approach

Varun Barthwal[1] · M. M. S. Rauthan[1] · Rohan Varma[2] · Sachin Gaur[3]

Received: 10 September 2023 / Accepted: 26 March 2024

Abstract
The background of the study is rooted in the critical importance of efficient virtual machine (VM) placement in cloud computing environments. VM placement efficiency is critical in cloud computing, especially when utilizing an improved genetic technique. This paper incorporates genetic meta-heuristic to integrate VMs into the minimal number of physical machines (PMs). In order to describe the fitness function in the proposed algorithm (GaMat and GaLin), we have also incorporated predicted usage of PMs CPU usages. The aim of the article is to demonstrate the effectiveness of genetic meta-heuristic in improving VMs placement efficiency in cloud computing environments. The study focuses on minimizing energy consumption (EC) VM migration and violations of the Service Level Agreement (SLA) by integrating VMs into the minimal number of PMs using the proposed algorithms. The algorithms' performance is evaluated by comparing them with the best-fit power-aware decreasing (Pa) VM placement strategy, based on metrics like EC, VM migration, and SLA violations. Tests were conducted in CloudSim through detailed simulations using actual workload data. The average values of the performance metrics for 10 days of the workload are collected for the proposed VM placement approach. The proposed work reduces EC by 25%, VM migration more than 50% and SLA by 58% when compared to the power aware best fit decreasing. The results of the simulations are interpreted and analysed, which shows the effectiveness of the proposed algorithms in contrast to the best-fit Pa strategy. In conclusion, the study demonstrates the effectiveness of genetic meta-heuristic based proposed algorithms (GaMat and GaLin) in optimizing VM placement in cloud computing environments. By integrating VMs into the minimal number of PMs while considering predicted resource usage, GaMat and GaLin significantly reduce EC, VM migration, and SLA violations compared to the best-fit power-aware decreasing (Pa) strategy.

Keywords VM placement · Energy consumption · Genetic algorithm · VM consolidation · Service level agreement

Introduction

Due to the convergence of IoT and cloud computing to create smart applications [1], the proliferation of consumer needs for computing services has contributed to the growth of cloud technology on a large scale. It is also contributing to massive growth in the infrastructure of storage or cloud datacenters. Cloud datacenters primarily consume energy as user requests are computed and produce unnecessary heat that affects the efficiency of relevant equipment. Therefore, to preserve the temperature in the cloud datacenter, cooling is necessary. Thus, computing components and cooling devices are the primary consumers of energy consumption in the datacenter. Research has found that cooling infrastructures account for 40 percent of the cloud datacenter's total energy requirement [2]. In addition, the datacenter releases traces of carbon, also one more reason for environmental issues.

This article is part of the topical collection "Advanced Computing: Innovations and Applications" guest edited by Sanjay Madria, Parteek Bhatia, Priyanka Sharma and Deepak Garg.

✉ Varun Barthwal
varuncsed1@gmail.com

M. M. S. Rauthan
mms_rauthan@rediffmail.com

Rohan Varma
rohan_12varma@yahoo.com

Sachin Gaur
ersgaur1234@gmail.com

1 Hemvati Nandan Bahuguna Garhwal University, Garhwal, Uttarakhand, India

2 Graphic Era University, Dehradun, India

3 BT Kumaon Institute of Technology, Almora, Dwarahat, India

Published online: 20 April 2024

Therefore, for the development of an eco-friendly cloud data center, a green computing environment must be encouraged. In a study it was found that global EC in 2015 was more than 600 TWh [3]. A study shows that the datacenter EC will gain a rise from 200 TWh in 2016 to 2967 TWh in 2030 [4]. The Fiona et al. study estimates the EC of hyper scale datacenters will nearly quadruple between 2015 and 2023 [5]. The work done [6] was mainly focused energy consumption issue where, an integrated model for IoT-fog-cloud was developed to use available resources in a efficient manner and gain proper load balancing. Apart from EC, security is also a key concern in cloud datacenter, a high level virtual machine (HLVM) was developed to provide a secure trading system environment [7]. Virtualization is the key feature of cloud computing, in this proposed work, virtual machines are mapped to physical machines, where CPU and memory have been virtualized in a single PM to process applications in multiple VMs in isolation. Virtualization of network may be useful to manage EC in cloud environment. Ajagbe et al. [8] developed a model for network virtualization, where virtual switches were created using emulator.

Therefore, the minimization of EC in the cloud datacenter is a significant activity that may effectively reduce the expense of cloud infrastructure. It was observed that PM's CPU usage has a linear relationship with its EC [9], so CPU use presence is considered to test the EC in the cloud datacenter in most of the studies. Cooling of the datacenter to manage the thermal atmosphere of the datacenter also consumes energy greatly. It is therefore important to minimize the heat's after-effects when the user requests are computed. Computing and cooling must also be accomplished with minimal EC and maximum service delivery. The latest researches have been undertaken on resource utilization and has been analyzed using criteria pertaining to energy management and the quality of service [10]. The proposed work aims at improving the efficiency of the operation, the allocation of services, and a substantial decrease in the EC standard.

The main task for reducing EC is the dynamic consolidation of VMs (DCVM) in a cloud datacenter. Because of its similarities to the bin packing problem, heuristic, metaheuristic and soft-computing methods have been used previously to solve the VM placement problem. The placement of VMs was carried out in our study using the Genetic Algorithm (GA), which is a meta-heuristic suggested by Holland [11, 12] as shown in Table 1. It is predominantly used to create high-quality solutions using nature-inspired operators such as selection, crossover, and mutation to optimize search issues. The fitness function has been defined using predicted utilization of PMs. In many research fitness function is calculated using various factors e.g. minimization of EC, SLA, rank based etc. In several of the experiments, DCVM was conducted in the cloud datacenter to increase service efficiency, resource usage, and EC minimization. It requires the identification of overloaded and under-used PMs, prediction of resource usage, collection of VMs, and location of migrating VMs to the destination PM [13]. In this method, to minimize overloaded and underutilized PMs, the computational loads on PMs are spread among the minimum active PMs. The overloading and underuse of PMs decrease the system's efficiency. Therefore, VM migration is required to transfer VMs from one PM to another without interrupting the system's performance. However, migration incurs significant degradation in the efficiency of the system [14].

In this article to consolidate VMs dynamically, two VM placement approaches (GaMat and GaLin) are implemented, where GaLin is inspired by the work done in previous research. The identification of overloaded PMs and the estimation of usage (Pu) were performed using the Local Regression (Lr) approach in this work. In addition, sufficient VM selection is made to eliminate overloading in PMs using the Minimum Migration Time (MMT) approach [15]. Lr-Mmt prepares a VM list from the overloaded PMs, this VM list is one of the input for the solution coding process in the proposed work and predicted utilization has been used to define the fitness function. Finally, the positioning of VMs is done by proposed methods. The findings showed proposed

Table 1 Genetic algorithm's parameters and their role

S. no.	Genetic algorithm's parameter	Role in genetic algorithm
1	Population	The population is a collection of potential solutions. Although a bigger population offers a more diverse search space, it also needs more computing power
2	Fitness function	The algorithm is directed toward better solutions through this fitness function. Higher values of fitness of individuals are more likely to be selected for reproduction
3	Selection	Individual from among present solutions will be chosen to reproduce and give rise to the future generation is determined through selection process
4	Crossover	In a process of crossover, genetic information from two parents solution is combined to produce one or more solution
5	Mutation	An individual's genetic information undergoes minor, random changes due to mutation. As a result, diversity is increased in solutions

methods efficacy and outperformed most current heuristic methods. Therefore, meta-heuristic techniques may play a significant role in the autonomous management of virtual machines (VMs) in a cloud datacenter, effectively balancing energy consumption (EC) and ensuring the delivery of optimal service quality.

The subsequent portion of this document is organized into distinct sections. Section "Literature Survey" covers the relevant literature, investigating the utilization of the genetic algorithm for VM placement on PMs. Section Problem Formulation presents the proposed algorithm along with its detailed description. The simulation environment and findings that shows the efficacy of the Genetic Algorithm-based VM placement with the heuristic method are discussed in Sects. The Proposed Approach for VM Placement and Simulation Environment, and finally, the conclusion along with future work are given.

Literature Survey

In this segment, we have discussed the work done for resource management using a genetic algorithm-based approach only. Many methods were developed for the VM placement using genetic algorithm based meta-heuristics; some of them are discussed in the current article:

Hu et al. proposed a VM load balancing scheduling approach using a genetic algorithm that manages the load and minimizes the expense of migration. To construct the mapping solution, they used a tree structure. There was a tree representing each chromosome, where the root manages the nodes. For PMs, the second-level nodes were used, and third-level nodes were used to present the VMs. Compared to conventional scheduling algorithms, their approach effectively increased the load balancing and relocation cost [16]. A hybrid genetic algorithm (HGA) proposed by Tang et al. for the VM placement issue to handle the EC in cloud computing. It involves the infeasible solution repair and local enhancement to increase the exploitation potential and convergence of conventional GA [17]. A novel strategy for VM allocation using the family gene approach (FGA) to minimize EC and VM migration was suggested by Joseph et al. Compared with the standard genetic algorithm, FGA performed better in terms of computing time [18]. Algorithms have been developed by Li Deng et al. to boost VM placement stability patterns with lower migration overhead and decreased energy usage. Centered on the group encoding scheme, where multiple VMs live on a single PM, they coded the solution. Their simulation results demonstrated a major increase in VM redistribution stability [19]. In terms of IT facilities and datacenter ventilation, Arianyan et al. discussed energy efficiency. In order to measure the fitness value, they used the least rise in power after placement, the most available PM resources, and the least number of hosted VMs in PM. A scattered crossover was used in their work to generate new descendants. To conduct a mutation on the solution, the Gaussian distribution was used. To test the algorithm in CloudSim, detailed simulations have been carried out, and the results indicate that there has been a notable change in terms of EC, SLAV, and VM migration [20]. A memetic algorithm to address the multi-objective issues (EC, QoS, network traffic etc.) of VM placement was suggested by Fabio et al. They proposed a structure that formulates multi objectives VM placement issue. With separate problem cases, the authors checked their work, and experiment results showed that the proposed model can solve the multi-objective issues with significant PMs and VMs [21]. Oshin et al. proposed a decision-making VM placement method based on genetic meta-heuristics. They analysed it with three predefined methods of VM positioning, and the results revealed that EC and SLA violations were successfully overcome by the genetic algorithm [22]. Mosa et al. tackled the issue of VM positioning by understanding the CPU and memory criteria for VMs. To reduce underutilized/over-used PMs and boost the SLA, they developed a genetic algorithm-based system. Tournament selection was made for each generation, and then uniform crossover was applied to generate new offsprings [23]. Amin et al. proposed a cost and energy-effective VM placement solution to address the minimization of cost and energy usage of datacenter. VM placement was formulated and solved by using genetic meta-heuristic as a mixed-integer nonlinear programming problem [24]. A micro-genetic algorithm for achieving a trade-off in the cloud datacenter between EC and SLA was suggested by Maryam et al. In order to address the issue of dynamic consolidation of VMs, they used k-means and micro genetic algorithms (KMGA). The studies showed that KMGA achieved a smooth equilibrium between EC and SLA, minimizing the migration of VMs [25]. A multi-objective approach to forecasting resource usage and optimizing datacenter energy use was introduced by Tseng et al. For the optimization of resource utilization, the algorithm considered computing and memory resources usage of VMs and PMs. Based on resource usage historical statistics, it forecasts the necessary resources in the next time slots. Using the outcomes expected using GA, VM placement was achieved. Their proposed work increased the total CPU and memory consumption and substantially decreased EC [26]. Kaaouache et al. presented an energy-efficient placement of VMs using the hybrid genetic algorithm to reduce EC in PMs and communication networks. The collection of chromosomes for the crossover from the original population was performed using the roulette wheel process. Using the single-point crossover, new offspring with higher fitness values were produced. Two PMs from

Table 2 Comparison of existing GA based VM placement strategies

Paper	Population coding	Selection	Crossover	Mutation	Fitness function
Hu et al. [16]	Spanning Tree	Probability of individual	Rotating selection	Self-adaptive probability	Penalty Function
Tang et al. [17]	Array	Roulette wheel selection	Biased Uniform	Inverting a gene randomly	Depends upon the EC of a solution
Joseph et al. [18]	Family genetic algorithm	Fitness based	Random	Self-adjusting probability	Resource utilization
Deng et al. [19]	Group encoding	Random selection	Multi-point	Random	Resource utilization
Arianyan et al. [20]	Array	Random selection	Scattered function	Gaussian distribution	PM power and resource utilization
Fabio et al. [21]	Matrix	Binary Tournament	Single point	Self-adaptive probability	Rank-based
Oshin et al. [22]	Tree structure	Random selection	Random	Random	PM power and resource utilization
Mosa et al. [23]	Array	Tournament selection	Uniform	Random	EC and cost
Amin et al. [24]	Array	Roulette wheel selection	Single point	Random	EC and cost
Maryam et al. [25]	Array	Tournament selection	Two-point	Random	EC based
Tseng et al. [26]	Array	Random selection	Two-point	Random	Placement result and resource status
Kaaouache et al. [27]	Array	Roulette wheel selection	Single point	Random	Depends upon EC of a solution
Abohamama et al. [28]	Permutation	Roulette wheel selection	Ordered	Random	EC minimization

a chromosome were randomly chosen, so, in the mutation phase, VMs from the selected PMs were switched. Experimental studies have shown that it outperforms the other strategies significantly [27]. A permutation-based genetic algorithm has been developed by Abohamama et al. that improves the rate of EC and minimizes the active numbers of PMs. They accomplished the load balancing of active PMs' processing, memory, and network resources. They developed a model by integrating the enhanced permutation-based GA (IGA-POP) and the resource-aware best fit strategy for energy-efficient VM placement [28]. The non-dominated sorting genetic algorithm (NSGA-III) was used by Parvizi et al. to solve the problem of VM placement for multi-objective optimization. Their main goal is to mitigate the waste of resources, EC, and active PMs. In the context of a nonlinear convex optimization strategy, they formulated VM positioning and used NSGA-III meta-heuristics to reduce the complexity of time. In a study proposed model was tested in simulator, and the findings show that NSGA-III increased the efficiency of the method relative to the other algorithms of MO-VMP [29].

Similar to the work done in as mentioned in Table 2, our work mainly focused on implementation of genetic based algorithm for VM placement. However, in comparison to the study described above we have used predicted utilization for the calculation of fitness value. In relation to the methods listed above, our major contributions are as follows in this article:

1. Using a genetic algorithm, a method has been proposed to address the problem of VM placement.
2. In this problem, minimizing energy consumption (EC), maximizing SLA, VM migrations reduction, and avoiding PM overload are established as goals in the VM placement process.
3. An algorithm, GaMat, in which chromosome architecture is based on matrix form, representing the mapping between VM and PM, has been suggested and implemented.
4. An algorithm, GaLin, was introduced in which chromosomes are designed in a linear fashion (linear representation of chromosome is used in most of the research, as shown in Table 2).
5. A fitness function that measures the fitness of the solutions using the predicted use of the PM's CPU utilization is proposed.
6. In CloudSim, the performance of GaMat and GaLin is measured using PlanetLab workload and SPEC benchmarks. Additionally, with a power-aware best fit decreasing (Pa) heuristic-based VM placement process, we conducted a comparative study of these algorithms.

Problem Formulation

The VM placement approach involves mapping VMs to PMs, where the quest space encompasses a cumulative total of *mn* possible directions for mapping *m* VMs to *n* PMs. The problem of VM positioning is somewhat similar to the problem of bin packaging, in which products with varying costs are stored in bins of different sizes. VMs are represented by objects in this problem, and PMs are represented by bins. The problem is expressed as follows:

1. The datacenter consists of 'n' number of PMs $\{p1, p2, p3, \ldots, pn\}$ to hosts the 'm' number of VMs $\{v1, v2, v3, \ldots, vm\}$.
2. An objective function is used to position all VMs in such a way that the amount of PMs consumed is reduced with the fulfilment of the energy usage and quality of services constraints described below.

The objective is to accommodate maximum VMs in minimum PMs to satisfy the constraints mentioned below:

Constraint 1 In order to reach minimal EC in the datacenter, the overall number of PMs should be reduced to the minimum for the VMs placement.

$$\int(\mathrm{s}) = \mathrm{Minimize}|\mathrm{Active\ PMs}| \tag{1}$$

Constraint 2 To meet the full SLA of the datacenter, the VM requests should not exceed the PM capacity.

$$\mathrm{Cap}(\mathrm{PM_j}) > \mathrm{Avail}(\mathrm{PM_j}) + \sum_{i=0}^{k} \mathrm{Req}(\mathrm{VM_i}) \forall \mathrm{VM_i} \in \mathrm{Vm_{mig}} \quad \text{and} \quad \forall \mathrm{PM_j} \in \mathrm{Pm_{cand}} \tag{2}$$

where $Cap(PM_j)$ is the cumulative capacity of the services of the PM, $Avail(PM_j)$ is the PM capacity available, and $Req(VM_i)$ is the VM resource request. Vm_{mig} is the list of migrated VMs from overloaded PMs, and Pm_{cand} includes PMs not in the overloaded PMs and having non-zero predicted usage. Best fit heuristic is an approach for solving the bin packing problem; Beloglazov et al. [15] have used this approach to solve the VM placement problem. They have extended this approach in terms of energy consumption to perform energy aware PM-VM mapping. In order to plan an energy efficient PM-VM mapping, their algorithm first sort the migrating VM list in decreasing order of their CPU utilization then find a PM which exhibit minimum power increase when a VM has been allocated to it. The Pa algorithm is depicted below:

Algorithm 3.1: VM placement using power aware best fit decreasing (Pa) [13]

```
Input: PM list, migrating VM list
Output: PM-VM mapping
Sort all migrating VMs in decreasing order of their CPU utilization
foreach VM in VM list do
        minPower ← MAX
        allocatedPM ← NULL
end for
foreach PM in PM list do
        if PM has sufficient resources for VM then
            power(PM,VM) ← calculatePower (PM, VM)
            diff ← power(PM,VM)- power(PM)
        if diff < minPower then
            minPower ← diff
            allocatedPM ← PM
end for
return PM-VM mapping
```

The Proposed Approach for VM Placement

The first step to effectively handling VM placement difficulties is mapping VMs onto PMs. We have developed VM placement solutions based on the GA meta-heuristic and compared it to alternative heuristic methods. The goal of our proposed strategy is to reduce EC and active PMs. When distributing *m* VMs to *n* PMs in a cloud environment, a method that strikes an ideal balance between EC reduction and Service Level Agreement (SLA) maximization must be developed. Many solutions exist for the problem mentioned above, and there are many ways to build them. In this work, GaMat and GaLin are developed to explore the possible solutions for VM placement in the datacenter. Initially, random solutions are populated despite their non-feasibility. Further crossover and mutation operations are performed to improve their quality. The solutions are also known as population, and each solution is termed as a chromosome. GaMat works on a matrix-based chromosome, and GaLin is for the linear chromosome. (Fig. 1).

Populating Solution

In GaMat, each chromosome is represented by matrix form. In this representation, rows and columns correspond

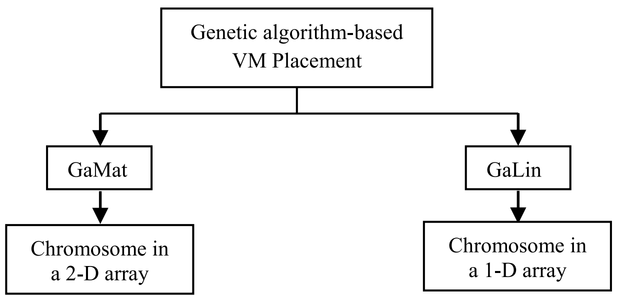

Genetic algorithm-based VM Placement
GaMat
GaLin
Chromosome in a 2-D array
Chromosome in a 1-D array

Fig. 1 Genetic algorithm based VM placement

to virtual machines (VMs) and physical machines (PMs), respectively. In our work, population coding has been done on a random basis because it provides a large search space domain to find a solution. In Eq. (3), x_{ij} represents the chromosome's gene whose value is 1, when a $\mathrm{VM_i}$ is successfully allocated to $\mathrm{PM_j}$ and 0 otherwise. VMs are randomly assigned to PMs; therefore, the solution may be feasible or infeasible depending upon successful VM placement.

$$x_{\mathrm{ij}} = \begin{cases} 1, \textit{if } \mathrm{Vm_i} \text{ is placed on } \mathrm{Pm_j} \\ 0, \textit{otherwise} \end{cases} \forall \mathrm{Vm} \in \mathrm{Vm_{list}} \quad \text{and} \quad \forall \mathrm{Pm} \in \mathrm{Pm_{list}} \tag{3}$$

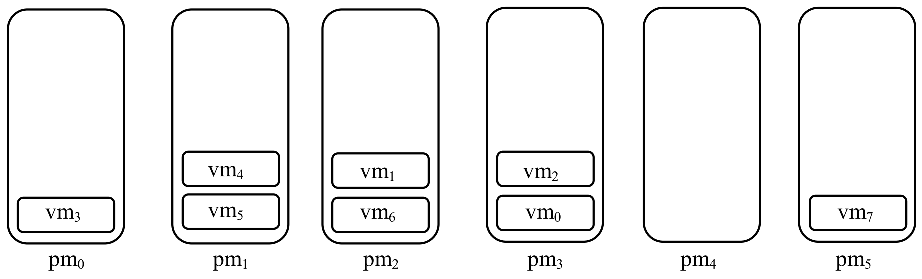
The chromosome representation is explained using an example (Fig. 2), where the VM allocation has been shown as $\mathrm{pm_0}$ $\{\mathrm{vm_3}\}$, $\mathrm{pm_1}$ $\{\mathrm{vm_4}, \mathrm{vm_5}\}$, $\mathrm{pm_2}$ $\{\mathrm{vm_1}, \mathrm{vm_6}\}$, $\mathrm{pm_3}$ $\{\mathrm{vm_2}, \mathrm{vm_0}\}$, $\mathrm{pm_4}$ and, $\mathrm{pm_5}$ $\{\mathrm{vm_7}\}$. The matrix in Fig. 2, shows the allocation map where 8 VMs are mapped to 6 PMs as mentioned in the above example. According to Eq. 3, the elements $(x_{0,3}, x_{1,4}, x_{1,5}, x_{2,1}, x_{2,6}, x_{3,2}, x_{3,0}, x_{5,7})$ of the matrix are assigned 1 and the rest are made 0.

For the same example, the linear chromosome is designed for GaLin as depicted in Fig. 3. The variable x_i, represents the $\mathrm{PM_j}$ if it hosts the $\mathrm{VM_i}$ as shown in Eq. 4.

$$x_{\mathrm{i}} = \mathrm{PM_j}, \textit{if } \mathrm{VM_i} \text{ is assigned to } \mathrm{PM_j} \tag{4}$$

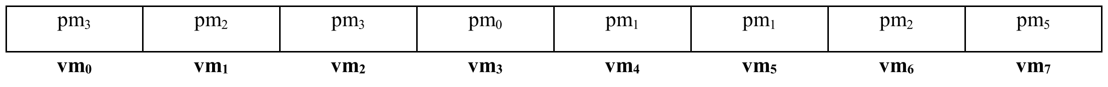
Fitness Function

The fitness function serves as a tool to assess solution's quality; a higher fitness value indicates more quality of the solution. In this work, Predicted usage (P_u) of PM is considered to derive the fitness value of each solution for GaMat and GaLin. Each PM's remaining capacity is calculated according to Eq. 5, and it is further used to find out the fitness value of the corresponding solution.



$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 2 Chromosome representation in matrix form



$\mathrm{pm_3}$	$\mathrm{pm_2}$	$\mathrm{pm_3}$	$\mathrm{pm_0}$	$\mathrm{pm_1}$	$\mathrm{pm_1}$	$\mathrm{pm_2}$	$\mathrm{pm_5}$
$\mathrm{vm_0}$	**$\mathrm{vm_1}$**	**$\mathrm{vm_2}$**	**$\mathrm{vm_3}$**	**$\mathrm{vm_4}$**	**$\mathrm{vm_5}$**	**$\mathrm{vm_6}$**	**$\mathrm{vm_7}$**

Fig. 3 Chromosome representation in linear form

In Eq. 5, Rem_{PM} is the available capacity of PM in a solution, Cap_{PM} is the overall capacity of PM, and Req_{VM_i} is the request of all VMs in PM. PM's P_u is calculated using the Local Regression method as shown in Eq. 6 [12]. Populated solution sets may contain some infeasible solutions, so some PMs' remaining capacity will be negative because the required resources are more than the total capacity. In this case, − 1 is assigned to the fitness value for that solution. The fitness value is the inverse of the predicted utilization (P_u) of PMs selected for VM placement in a solution in case of positive remaining capacity (Eqs. 7 and 8).

In this scenario, a fitness value of − 1 is assigned to the solution. The fitness value is determined as the inverse of the predicted utilization (P_u) of the selected PMs for VM placement, specifically in instances of positive remaining capacity, as described in Eqs. 7 and 8.

$$\mathrm{Rem_{PM}} = \mathrm{Cap_{PM}} - \sum_{i}^{m} \mathrm{Re}q_{\mathrm{VM_i}} \forall \mathrm{VM} \in \mathrm{Vm_{list}} \tag{5}$$

$$\mathrm{P_u} = \text{Local Regression(PM Utilization History)} \tag{6}$$

$$\text{Total } \mathrm{P_u} = \sum_{i}^{m} \mathrm{Pu}\left(\mathrm{PM_i}\right) \forall \mathrm{PM_j} \in \mathrm{Pm_{cand}} \tag{7}$$

$$\eta = \begin{cases} -1, \mathrm{Rem_{PM_i}} < 0 \\ \frac{1}{\text{Total } \mathrm{P_{u}},} \mathrm{Rem_{PM_i}} > 0 \end{cases} \forall \mathrm{PM_j} \in \mathrm{Pm_{cand}} \tag{8}$$

The same fitness function is also used for GaLin methods. In the case of GaLin, the linear chromosome is converted into a matrix form. Then fitness value is evaluated for the solution similar to GaMat.

Selection

In this process, the selection of parent solutions is made for the generation of the new solution. In our work, we select two consecutive chromosomes from the population. According to Eq. 9, if the fitness value is negative, then solutions are sent for the crossover to derive the new feasible offspring. In the case of the positive value of the solution's fitness function, mutation and crossover are performed according to Eqs. 10 and 11. The selection process is presented by equations below, where, s_i is a solution from the population, and $f\left(s_i\right)$ is the fitness function.

$$\mathrm{offspring_i} = \sum_{i=0}^{\text{length}} \text{crossover}\left(s_{\mathrm{i}}, s_{\mathrm{i-1}}\right), f\left(s_{\mathrm{i}}\right) = -1 \tag{9}$$

$$\mathrm{offspring_i} = \sum_{i=0}^{\text{length}} \text{mutation}\left(s_{\mathrm{i}}\right), f\left(s_{\mathrm{i}}\right) > 0 \tag{10}$$

$$\mathrm{offspring_i} = \sum_{i=0}^{\text{length}} \text{crossover}\left(s_{\mathrm{i}}, s_{\mathrm{i-1}}\right), f\left(s_{\mathrm{i}}\right) > 0 \tag{11}$$

In the case of GaLin, the selection process is similar to GaMat in which feasible solutions are taken for the mutation to derive the new solution, while a solution having a negative fitness value is selected for crossover to derive the new offspring.

Crossover

The genetic algorithm combines two individual solutions to form the next generation in the crossover process. In the case of GaMat, a new offspring is generated that inherits useful information from its parents. Two consecutive chromosomes ($s_i$and s_{i-1}) are selected from the population as shown in Eq. 9, where $x_{i,j}$, and $y_{i,j}$ are chromosome s_i and chromosome s_{i-1} genes. According to Eq. 12, swapping between these genes is done for the even index, and chromosome s_i is updated with new values. Finally, chromosome s_i is considered as a new offspring. We applied the multi-point crossover in the matrix representation of chromosomes.

$$x_{\mathrm{i,j}} = \begin{cases} \text{swap}\left(x_{\mathrm{i,j}}, y_{\mathrm{i,j}}\right), \text{i is even} \\ \text{do nothing, i is odd} \end{cases} 0 \le i \le m, 0 \le j \le n \tag{12}$$

In the case of GaLin, the random crossover is performed where a variable r is selected randomly (Eq. 13). Further, it is used to derive the crossover point according to Eq. 14.

$$\mathrm{r} = \text{radom selection between 1 to m} \tag{13}$$

$$\text{cspoint} = \mathrm{m/r} \tag{14}$$

$$x_{\mathrm{i}} = \text{swap}\left(x_{\mathrm{i}}, y_{\mathrm{i+cspoint}}\right) 0 \le i \le m \tag{15}$$

According to Eq. 15, the crossover process is applied on both feasible and non-feasible solutions to diversify the search; however, a crossover between two feasible solutions does not guaranty the feasibility of offspring generated after crossover. Thus, in our work, we have used the crossover to enhance the solutions' diversity by exploring more solutions in the solution space.

Mutation

The mutation is a process of modifying some genes randomly in a chromosome. A small change in the chromosome

Table 3 Comparison between GaMat and GaLin

Algorithm	Population coding	Selection	Crossover	Mutation	Fitness value
GaMat	Array based	Fitness based	Multi-point	Genes' shifting	PMs' predicted utilization
GaLin	Matrix based	Fitness based	Single point	Genes' shifting	PMs' predicted utilization

is done to implant the mutation in children. It allows the genetic operator to remove or edit one or more genes in a chromosome. GaMat takes gene $x_{i,j}$ from a chromosome (feasible solution) and replaces it with zero. If $x_{i,j}$ is one (Eq. 16), then it sets the value of the next gene at $x_{i,j+1}$ to one (Eq. 17). In the mutation process, VM's location is changed from one PM to another to reduce the active numbers of PMs (Eq. 18).

$$x_{\mathrm{i,j}} = 0, \quad \mathrm{if} x_{\mathrm{i,j}} = 1 \tag{16}$$

$$x_{\mathrm{i,j+1}} = 1 \tag{17}$$

$$f(\mathrm{PM}) = \min \left|\mathrm{PM}_{\mathrm{active}}\right| \tag{18}$$

In the case of GaLin, random mutation is performed in which the location of VM is changed from one PM to another which results in minimizing the active PMs in a solution, as shown in Eq. 19.

$$x_{\mathrm{i+1}} = x_{\mathrm{i}} 0 \leq i \leq m \tag{19}$$

The mutation has been applied to explore best solution which consists of minimum active PMs. It is similar to the exploitation process in meta-heuristic techniques.

Algorithm Description

GaMat

In this work, k numbers of solutions are developed where each solution is a chromosome. Each solution is represented as a two-dimensional array in which 1 represents a PM-VM mapping, and 0 represents no mapping. In the first phase, solutions are developed randomly without considering their quality and efficiency. So, initially populated solutions may be correct or not (Algorithm 4.1). Table 3 depicts the differences between GaMat and GaLin in terms of their genetic parameters.

Algorithm 4.1: Populating solution

```
1.  Input: m (Migrating VM list) and n (Candidate PM list)
2.  Output: Random Solution
3.  Each element of the Solution matrix is set to zero
4.  PmIndex is initialized with a random number (0 to n)
5.  for each PM_i from the PM list, do
6.          temputil (PM) ← 0
7.  end for
8.  for each VM ϵ VM list do
9.          VM is added to VM_candidate list
10. end for
11. allocatedVM is initialized to null
12. for each i to m do
13.         allocatedVM = candidateVM_i
14.         solution matrix_(allocatedVM id, PmIndex) ← 1
15. PmIndex = random number (0 to n)
16. end for
17. solution_i,j ← solution matrix_i,j
18. return solution
```

Algorithm 4.2 is developed to estimate the fitness value, which represents quality of the solution. High fitness means the high quality of the solution. If a solution is not feasible, − 1 is assigned to the fitness value for that solution (line 16). In GaMat and GaLin, The assessment of the fitness value depends on the predicted utilization of the PMs (line 18). In the case of the positive value of available capacity, the fitness value is the inverse of PMs' predicted utilization (line 20).

Algorithm 4.2: Computing the fitness function

```
1.  Input: Solution
2.  Output: Fitness value
3.  for each PM_i from PM list do
4.          temputil (PM_i) ← currentUtil (PM_i)
5.          availcap (PM_i) ← currentAvailcap(PM_i)
6.  end for
7.  for each VM_i from VM list do
8.      for each PM_j from PM list do
9.          if (Solution_{i,j} == 1)
10.             temputil (PM_j) ← temputil (PM_j) + VM_i request
11.         end if
12.     end for
13. end for
14. for each PM_j from PM list do
15.     if (temputil(PM_j) != currentUtil (PM_i))
16.         availutil (PM_j) ← PM_j capacity - temputil(PM_j)
17.     end if
18.     if (availutil (PM_j)<1)
19.         return -1
20.     end if
21.     else
22.         Pu_total = Pu_total + predicted utilization of PM_j
23. end for
24. fitness value ← 1/ Pu_total
25. return fitness value
```

In the selection phase (Algorithm 4.3), chromosomes whose fitness values are not equal to -1 are selected for mutation. The variable *solution length* expresses the total solutions to be developed in the selection step. The *solution list* consists of solutions that were randomly generated in the first phase. Initially, there is a low probability of selecting a feasible solution because most solutions may have a negative fitness value. In the selection process, a solution having positive fitness value is further chosen for the process of mutation, and then the mutated child is appended to fitted solution list (line 8). Further, it is sent for the crossover to replace it with new offspring (line 10). The solution having a negative fitness value (the infeasible solution) was selected for the crossover in anticipation that a feasible solution may be generated, and further, it is added to the population (line 12). If the algorithm cannot find a feasible solution, repairing is executed to handle such an issue. In this process, the existing population is modified to create a new population. Hence, this process returns all feasible solutions. The solution that achieves the maximum fitness is ultimately opted for the placement of VMs. (line15).

Algorithm 4. 3: Selection

```
1.  Input: solution length s, solution list
2.  Output: best solution
3.  variable c is initialized to zero
4.  while ( c < s )
5.      for each i to s (solution length) do
6.          double fitnessvalue ← fitnessvalue (solution_i)
7.          if ( fitnessvalue != -1)
8.              Fitsolution ← mutation (solution_i)
9.              c++
10.             solution_i ← crossover(solution_i, solution_{i-1})
11.         end if
12.         else
13.             solution_i ← crossover(solution_i, solution_{i-1})
14.     end for
15. end while
16. best solution = fitsolution having maximum fitness value
17. return the best solution
```

The third phase of GaMat is the crossover (Algorithm 4.4); variable $p1$ and $p2$ represent the parent solutions from the population, and *offspring* is the output variable resulting from the crossover between $p1$ and $p2$. We have used a multi-point crossover in this work where two consecutive solutions from the initial population list (generated in phase one) are selected for the crossover. Offspring is generated by the swapping rows of the even index of both arrays (line 6). These offspring replace the solution ($p1$) from the population in anticipation that the offspring is a feasible solution (line 9).

Algorithm 4.4: Crossover function (GaMat)

```
1.  Input:p1, p2
2.  Output: offspring
3.  for each i to the size of (VM list) do
4.      if (i%2 !=0)
5.          for each j to the size of (PM list) do
6.              offspring ← swap p1_{i,j}, and p2_{i,j}
7.          end for
8.  end for
9.  p1 ← offspring
10. return offspring
```

In Mutation phase (Algorithm 4.5), a chromosome is modified by changing its genes (lines 7, 8). The change is done if a gene is found 1 (mapping), then its value is converted to 0 (no mapping), and the value of the gene at the next column having the same row is made 1 (mapping). This act is only for reallocating VM from one PM to another to minimize PMs' active numbers in a chromosome. Further, the solution quality is tested, and if it is found feasible, then the solution is considered a mutated solution (line 11); otherwise, the chromosome is modified again.

Algorithm 4.5: Mutation (GaMat)

```
1.  Input: Fit solution (FS)
2.  Output: Mutated solution (MS)
3.  temp ← FS
4.  for each j to size of (PM list) do
5.      for each i to size of (VM list) do
6.          if ( temp_{i,j} == 1) && (j < size of (PM list) -2)
7.              temp_{i,j} ← 0
8.              temp_{i,j+1} ← 1
9.          end if
10.     end for
11.     if (fitness value (temp) != -1)
12.         MS ← temp
13.     else
14.         temp ← FS
15. end for
16. return MS
```

GaLin

Like GaMat, k numbers of solutions are developed where each solution is a chromosome of the population. In the first phase, solutions are developed randomly without considering their quality and efficiency. So, initially populated solutions may be correct or not (Algorithm 4.6). The third phase of GaLin is the crossover (Algorithm 4.7); variable $p1$ and $p2$ represent the parent solutions from the population, and *offspring* is the output variable resulting from the crossover between $p1$ and $p2$. In GaLin, single point crossover has been applied in which a variable is selected randomly between 0 and the total number of migrating VMs. Further, this variable is used to determine the crossover point (line 4). Two consecutive solutions from the initial population list (generated in phase one) are selected for the crossover. Offspring is generated by the swapping of elements between parent solutions according to line 6, and the resulted array is stored as offspring (line 9).

Algorithm 4.6: Populating solution

```
1.  Input: m (Migrating VM list) and n (Candidate PM list)
2.  Output: random solution
3.  Each element of the Solution matrix is set to zero
4.  PmIndex is initialized with a random number (0 to n)
5.  for each PM_i from the PM list, do
6.      temputil (PM) ← 0
7.  end for
8.  for each VM ϵ VM list do
9.      VM is added to VM_candidate list
10. end for
11. allocatedVM is initialized to null
12. for each i to m do
13.     allocatedVM = candidateVM_i
14.     solution matrix_(allocatedVM id) ← PmIndex
15.     PmIndex = random number (0 to n)
16. end for
17. solution_i ← solution matrix_i
18. return solution
```

Algorithm 4.7: Crossover function (GaLin)

```
1.  Input: p1, p2
2.  Output: offspring
3.  Randomly selected variable t between 0 and total no of
    migrating VMs
4.  csPoint ← numberOfVMs / t
5.  for each i to csPoint do
6.      offspring ← swap p1_i and p2_{i+csPoint}
7.  end for
8.  p1 ← offspring
9.  return offspring
```

The last process of GaLin is the mutation (Algorithm 4.8), in which active numbers of PMs are minimized in the solution. In this process, the feasible solutions are modified to maximize their fitness value. In this phase, the genes from one index are shifted to the next index, which results in the shifting of VM from one PM to another according to lines

5 and 6. Moreover, this may result in the generation of the infeasible solution; therefore, the fitness value is evaluated in each iteration (line 11). Like GaMat, this act is only for reallocating VMs from one PM to another to minimize PMs' active numbers in a chromosome.

Algorithm 4.8: Mutation (GaLin)

```
1.  Input: Fit solution (FS)
2.  Output: Mutated solution (MS)
3.  temp ← FS
4.  initialize variable counter to zero
5.  while (counter < size of (VM list) – 2)
6.          temp[counter + 1] ← temp[counter]
7.          if (fitness value (temp) != -1)
8.              MS ← temp
9.          else
10.             temp ← FS
11.         if (fitness value (temp) != -1)
12.             MS ← temp
13.         else
14.             temp ← FS
15. end while
16. return MS
```

In Fig. 4, the whole process flow is depicted according to the above algorithms.

Simulation Environment

In most of the study, tests are conducted in the simulation because datacentre is unavailable. CloudSim [30] presents a virtual cloud system where tests for this work have been conducted. The two types of PMs for the dynamic consolidation of VMs are planned and simulated in the cloud environment (Table 4). Due to its practical functionality, PlanetLab [31] dataset is used (Table 5), which contains CPU use of VMs in five-minute intervals and stored in separate files.

The main components that affect PMs EC in datacenters are the CPU, memory, disk storage, power supply, and cooling systems. Study done by Fan et al., that EC by PMs can be described by a linear relationship between EC and CPU usage.

SPEC[1] power benchmark is employed to assess the energy consumption (EC) in our algorithm. In Table 6, for the various CPU usage levels percentages associated EC is given in watts.

Figure 5 below depicted that the proposed solution has been incorporated in CloudSim, the function *optimizeAllocation* calls the *getNewVmPlacement* function. Which supply the VM list and Overloaded PM list, from these overloaded PMs, Migrating VM list has been prepared. Then the function *PrepareSolution* takes the migrating VM list and candidate PM list as input, this function initiates the population coding process, then the proposed method returns the suitable PM-VM mapping.

[1] http://www.spec.org/power_ssj2008/.

Simulation Results and Discussion

GaMat uses Local regression and Minimum migration time (LrMmt) techniques to find out overloaded PM and migrating VM from the overloaded PM. Further, GaMat and GaLin techniques are applied for the VM placement from overloaded PMs.

GaMat, GaLin, and Pa [13] VM positioning strategies are measured against each other in terms of below mentioned performance parameters.

Energy Consumption (EC)

Fan et al. defined in their work that there is a linear pattern between PM's EC and its CPU consumptions [9]. The CPU consumption of PM is then used in this paper to approximate the EC. Using the following equation, the relationship between EC and CPU usage of PM was expressed:

$$P_{\text{total}} = P_{\text{idle}} + \left(P_{\max} - P_{\text{idle}}\right) \times U \quad (20)$$

where P_{total} is the total EC, P_{idle} is the EC by PM; $P_{\max}$ is the EC when the host is totally used when it is idle, and U is the CPU usage [15]. GaMat and GaLin remove all PMs with zero expected usage, since in the future, such PMs will be underused. As a consequence, resource underutilization is minimized, which prevents excessive energy use. Table 7 depicts the average number of active PMs in each VM placement method.

To assess the energy consumption of single PM, a PM (PM Id: 119) was randomly selected from the pool of PMs, and further we have found its energy consumption and CPU usage in each time interval depending on the number of running VMs as depicted in Fig. 6a, b. The total number of VMs running in each time interval for the same PM has been depicted in Fig. 6c.

As it is cleared that with more active PMs, more energy will be consumed in the cloud datacenter. We have also calculated the average numbers of active PMs as shown in Table 7. It shows that GaMat and GaLin use the same numbers of active PMs in their optimal solution, and is the reason for the similar EC. Figure 7a illustrates that Ga-based VM positioning has more effect in comparison to the heuristic algorithm in EC terms. In the proposed VM placement

Fig. 4 The overall process flow using proposed approach

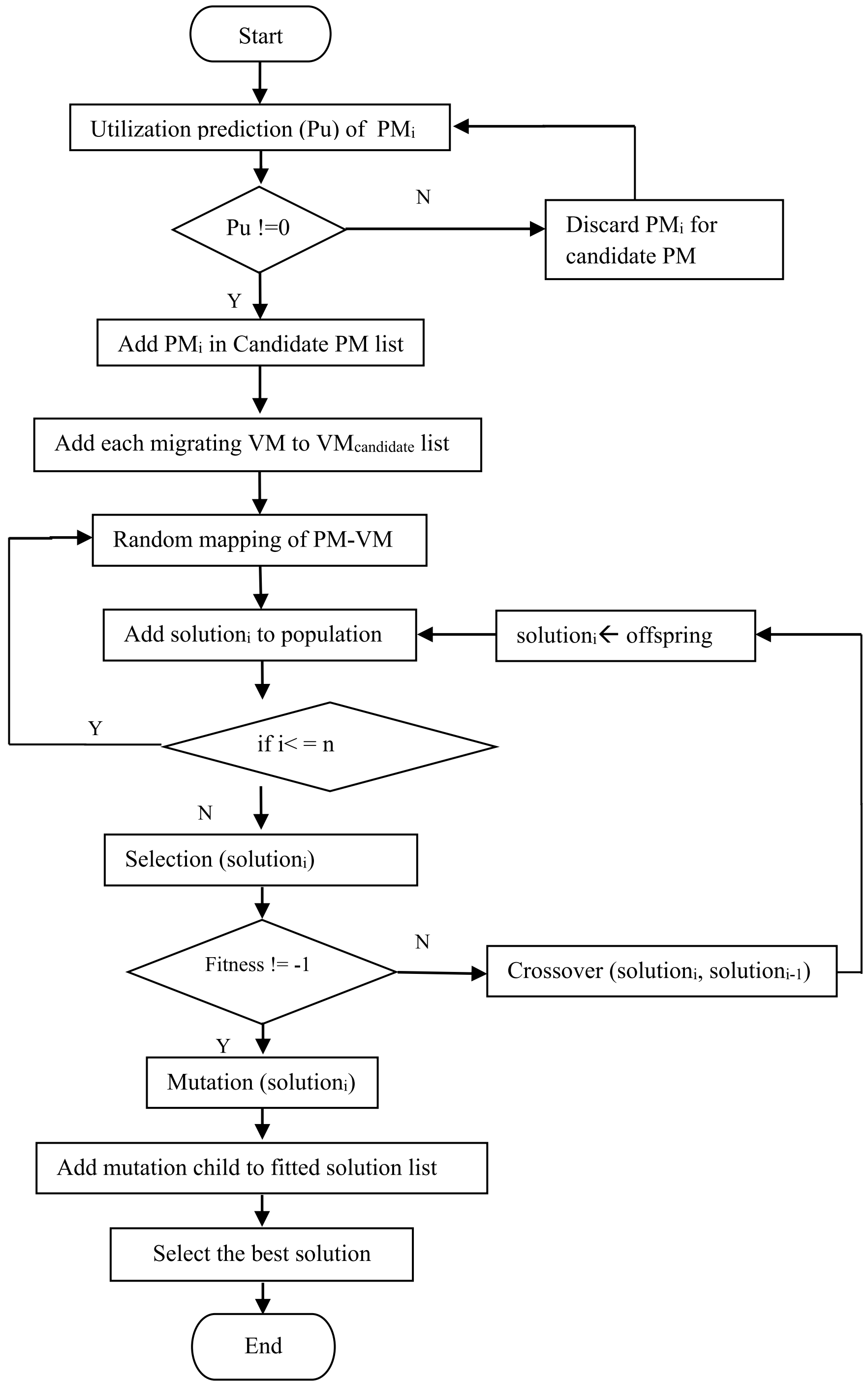

Start
Utilization prediction (Pu) of PMi
Pu !=0
N
Discard PMi for candidate PM
Y
Add PMi in Candidate PM list
Add each migrating VM to VMcandidate list
Random mapping of PM-VM
Add solutioni to population
solutioni ← offspring
Y
if i< = n
N
Selection (solutioni)
Fitness != -1
N
Crossover (solutioni, solutioni-1)
Y
Mutation (solutioni)
Add mutation child to fitted solution list
Select the best solution
End

Table 4 Simulation environment

Specification	Physical machine type (HP Proliant)		Virtual machine type			
	ML110 G5	ML110 G4	Type I	Type II	Type III	Type IV
CPU	1860 MIPS (2 cores)	2660 MIPS (2 cores)	500 MIPS	1000 MIPS	2500 MIPS	2500 MIPS
RAM	4096 MB	4096 MB	613 MB	1.7 GB	1.7 GB	0.85 GB

Table 5 Workload data

Day	03/03/11	06/03/11	09/03/11		22/03/11	25/03/11	03/04/11	09/04/11	11/04/11	12/04/11	20/04/11
Total VMs	1052	898		1061	1516	1078	1463	1358	1233	1054	1033

Table 6 Average EC (watt) at different computing load

Computing Load	Active Ideal	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP Proliant G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP Proliant G5	93.7	97	101	105	110	116	121	125	129	133	135

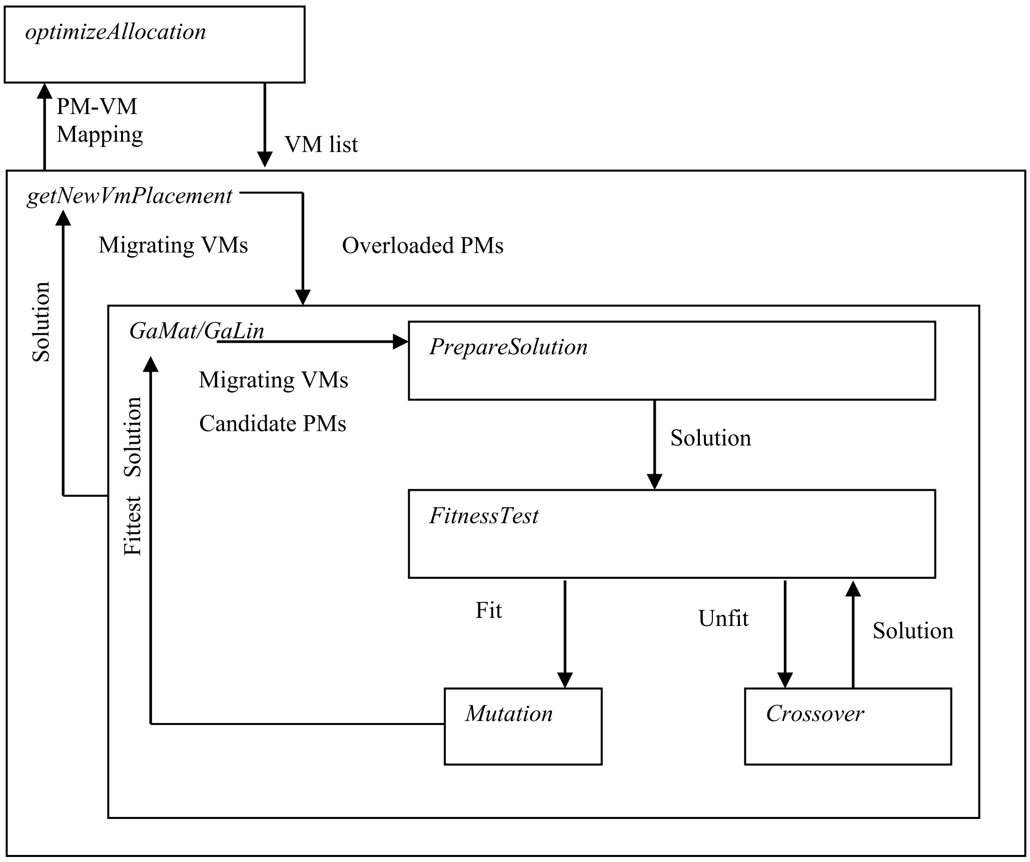


Fig. 5 Integration of the model into CloudSim

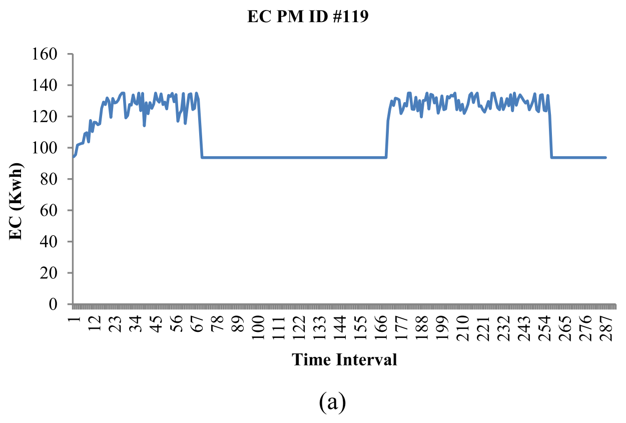
Table 7 Average number of active PMs in case of each VM placement method

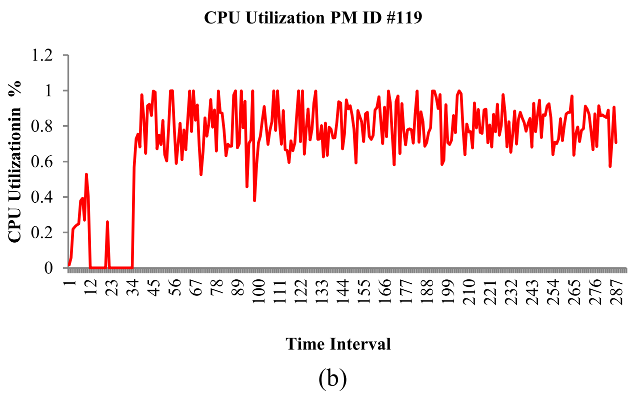
Day	03/03/11	06/03/11	09/03/11	22/03/11	25/03/11	03/04/11	09/04/11	11/04/11	12/04/11	20/04/11
PABFD	61	47	54	67	59	81	68	67	58	51
GaMat	45	35	39	47	41	60	48	47	41	35
GaLin	45	35	39	46	41	59	48	47	41	35

method, the utilization of resources is improved and positively impacts EC. Both of the methods, GaMat and GaLin, reduce the EC compared with the Pa placement method. Here, it can be seen that the GA system-based placement technique can play the lead role to minimize the EC.

VM Migration

The key challenge for achieving DCVM is virtual machine migration; the migration strategy helps the device to move VMs without interference from one PM to another [15]. The relocation of VMs also affects the system's efficiency [14]. 10 percent of CPU usage in our work is considered an average reduction in the output of the system, leading to further SLA breaches. In the migration process, SLA breaches are faced by all PMs before the migration process is finished. Therefore, with minimal VM migration, a successful DCVM consolidates VMs. Equations 21 and 22 were used to test the VM migration duration and the deterioration of efficiency due to migration. When VM migration occur more, system's efficiency degrades [14, 15]. Therefore, the total VM migrations are regarded as an output parameter in the suggested solution. However, migration is also expected for the DCVM, but it is important to prevent needless VM migration. The foregoing DCVM mechanism was implemented for the actual workload and the VM migration numbers were analyzed. Experiment findings revealed that in terms of migration counts, GaLin outperformed the DCVM techniques based on GaMat and Pa, as seen in Fig. 7b.





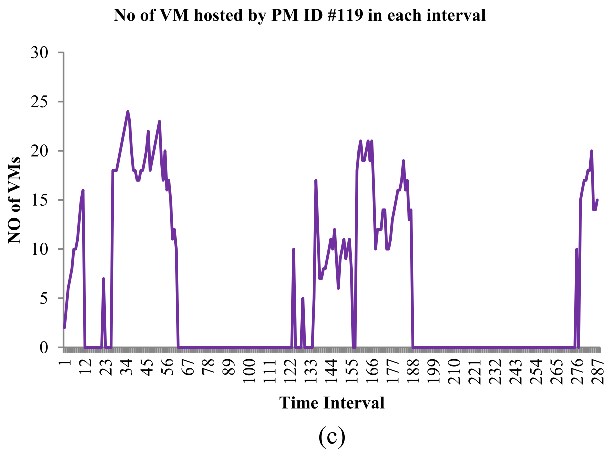


Fig. 6 Single PM (ID #119) analysis in each time interval in terms of **a** energy consumption **b** CPU usage **c** no. of VM hosted

$$\mathrm{Tm}_j = \frac{M_j}{B_j} \tag{21}$$

$$\mathrm{Ud}_j = 0.1 \times \int_{t_0}^{t_0+Tm_j} u_j(t)dt \tag{22}$$

where Tm_j shows total migration duration, M_j shows memory utilized by VM_j, and B_j shows available network bandwidth. Ud_j shows total degradation in the performance of the system, t_0 shows migration initiation time and $u_j(t)$ shows CPU utilization by VM_j as explained in [15] the 10% performance degrade due to VM migration.

SLA Violations

As seen in Eq. 23, SLA violation is also evaluated the efficiency of the operation, which is the result of performance loss caused by VM migration (PDM) and SLA time per active host (SLATAH). In a study done by Beloglozov et al. [15] that in PM overloading scenario, the source PMs experience SLA violation while a VM is in the migration process, thus reducing system output during the migration process. The following are described by the PDM and SLATAH:

$$\mathrm{SLAV} = \mathrm{SLATAH}\ x\ \mathrm{PDM} \tag{23}$$

$$\mathrm{PDM} = \frac{1}{N}\sum_{i=1}^{M}\frac{C_{dj}}{C_{rj}} \tag{24}$$

$$\mathrm{SLATAH} = \frac{1}{N}\sum_{i=1}^{N}\frac{T_{Si}}{T_{ai}} \tag{25}$$

where due to migrations, *Cdj* is the expected performance loss of the *VMj*, and *Crj* is the total requested *VMj* CPU power. *N* shows number of PMs, *Tsi* shows duration in which 100 percent of *PMi* has been used leading to a SLAV, *Tai* shows active PMs counts, *M* represents VMs [15]. It can be seen in the graph (Fig. 7c, d) that the proposed approaches greatly decrease the violation of the SLA and works better in terms of service efficiency.

Overload Counts

Investigation of PM overloading counts has been done in the proposed work to assess the system performance. Overloading counts for all PlanetLab workloads were tested in the simulation for the entire period. It impacts both the EC and SLA, as soon as overloading increases. It is then discussed in the work presented as well in Fig. 7e, it was found that, GA needs further development to minimize the overloading of PMs.

Comparative Analysis Between GaMat, GaLin and Pa

Table 8 summarizes a comparative examination of the proposed work.

SLATAH needs further reduction, as seen in Table 8, since it reflects the larger amount of overloading PMs, resulting in SLA violations (Fig. 7).

Table 8 Average results of VM placement policies for 10 days workload

Algorithm	EC Kwh	SLA %	VM Migration	PDM	SLATAH	Overload count
PaBFD	161.87	0.00497	28,174	0.08	6.213	2948
GaLin	120.58	0.00227	13,582	0.04	5.158	2688
GaMat	121.32	0.00209	16,252	0.04	5.144	2845

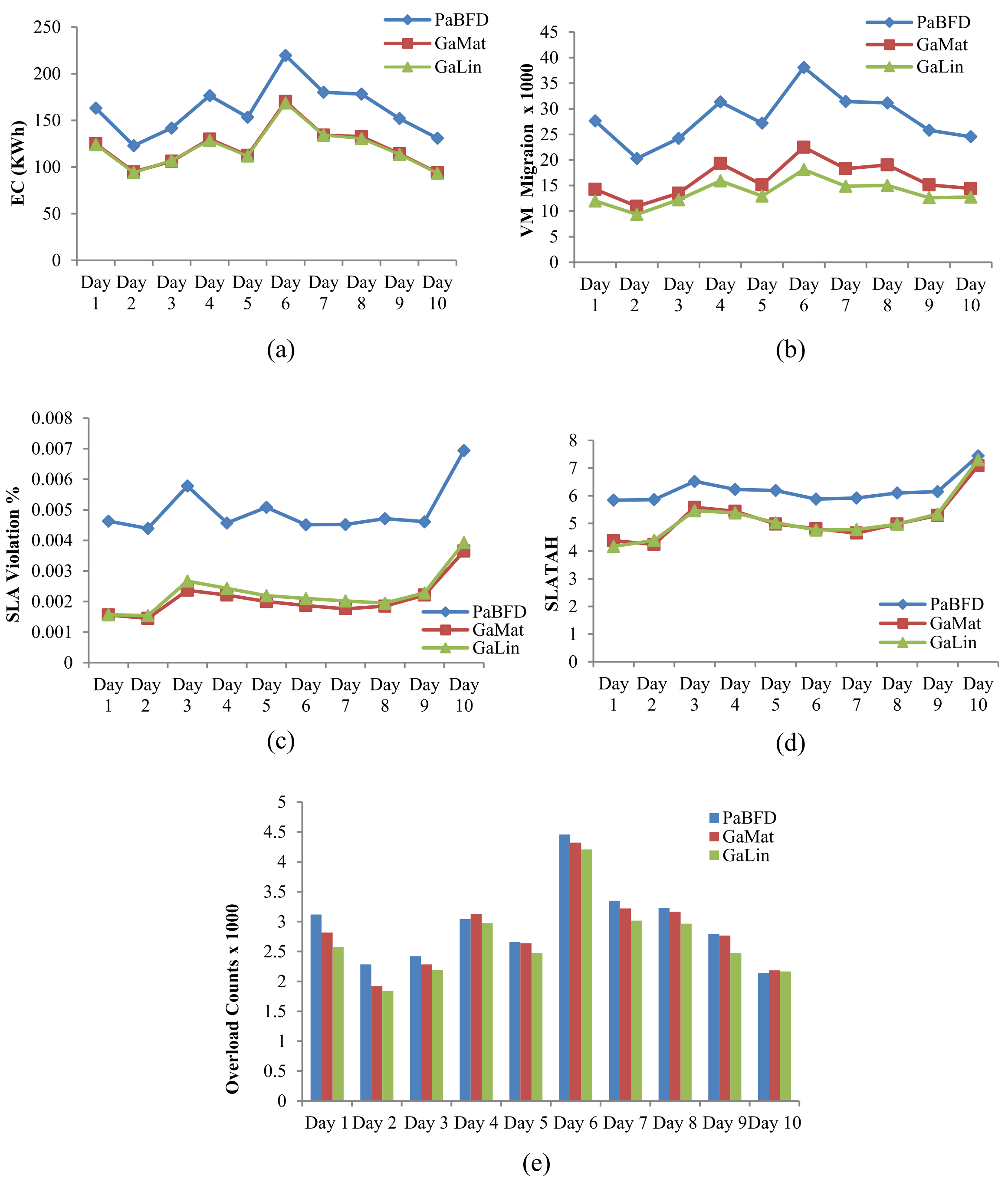


Fig. 7 Results of GaMat, GaLin and PaBFD techniques **a** EC, **b** VM Migration, **c**, **d** SLAV and **e** overload counts

Table 9 The simulation results (comparison of different algorithms of VM placement)

Performance metrics	GaMat versus Pa (%)	GaMat versus GaLin (%)	GaLin versus Pa (%)
Energy	25.5	0.6	25.04
VM migration	42.31	16.42	51.8
SLA	57.92	7.71	40.9
Overload count	3.4	5.51	8.81

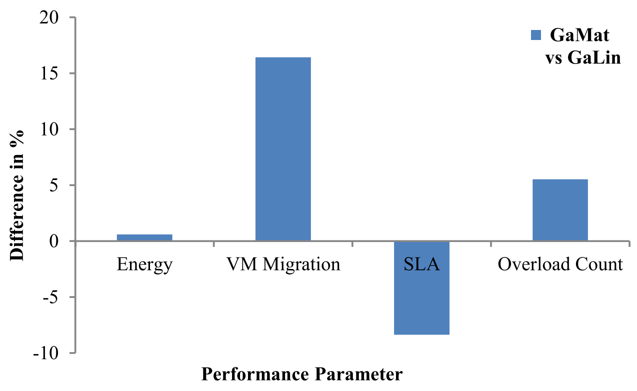


Fig. 8 GaMat versus GaLin

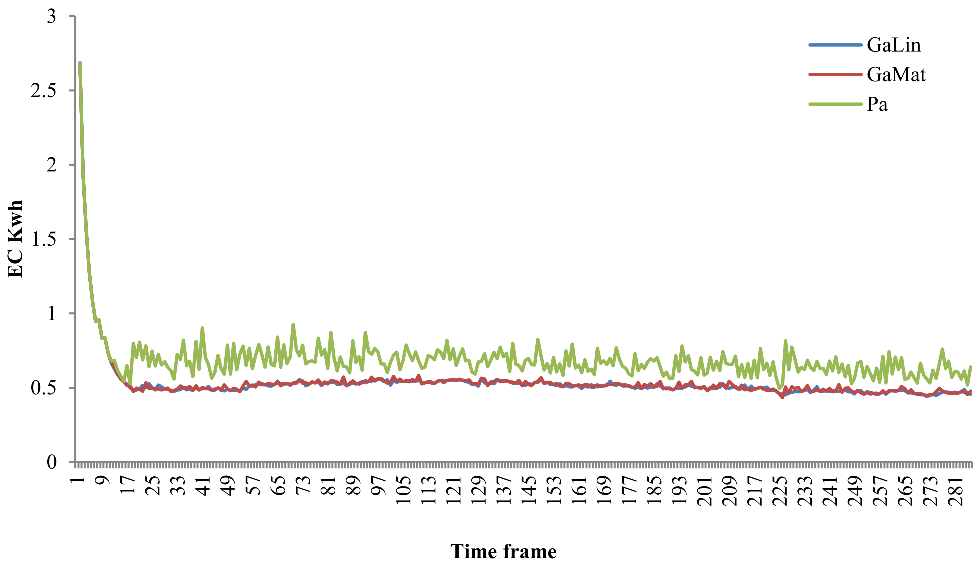
LrMmt algorithm [15] has been applied in the proposed work for the identification overloaded PMs and selection of VMs for migration. The average values of the performance metrics for 10 days of the workload are collected for GaMat based VM placement, where it improved the EC by 25.5% against Pa and GaLin outperformed GaMat by 0.6%. When GaLin is compared with Pa, an improvement of 25.04% in EC is observed. GaMat reduces SLA violation by 57.92% and 7.71% against Pa and GaLin VM placement policies. GaLin improved the SLA by 40.9% against Pa. It has been observed in the simulation that 42.31 percent of VM migration counts are lowered using the GaMat process, while GaLin lowers migration counts to Pa by 51.8 percent. In terms of overload counts, GaMat-based VM placement was found to be less productive since it decreases these counts by 3.4 percent. When GaLin was applied as a VM positioning strategy, the overload count to Pa resulted in a reduction of 8.81 percent. However, the proposed work needs further enhancement in terms of migration and overload counts (Table 9).

In this article, GaMat is based on 2-D chromosome design, while GaLin is applied by considering the chromosome in a linear manner, as shown in Table 2; it was used in most of the VM placement GA-based solutions. As seen in Fig. 8, in case of lowering VM migration and PM overloading, GaMat is higher than GaLin, although in the case of EC, GaMat varies significantly from GaLin. In the case of GaMat, though, it's a little more, as seen in Fig. 8. In terms of the output parameters described above, the Genetic based technique is found to be better that can lead to better outcomes. It is also inferred that GaMat (2-D chromosome) may become an alternative approach for the positioning of the VMs in the Datacenter.

We took the energy consumed values in each time interval after applying both approaches with a similar workload (Day 04/03/2011). Figure 9 illustrates that the EC is less than 1.5 kWh for every time frame of 24-h period. GaMat has smooth patterns compared to Pa, which proves the better performance of dynamic consolidation in terms of EC.

VM migrations in each time frame are also collected for the same workload and shown in Fig. 10. The plot shows that

Fig. 9 EC in each time frame: Day 04/03/2011



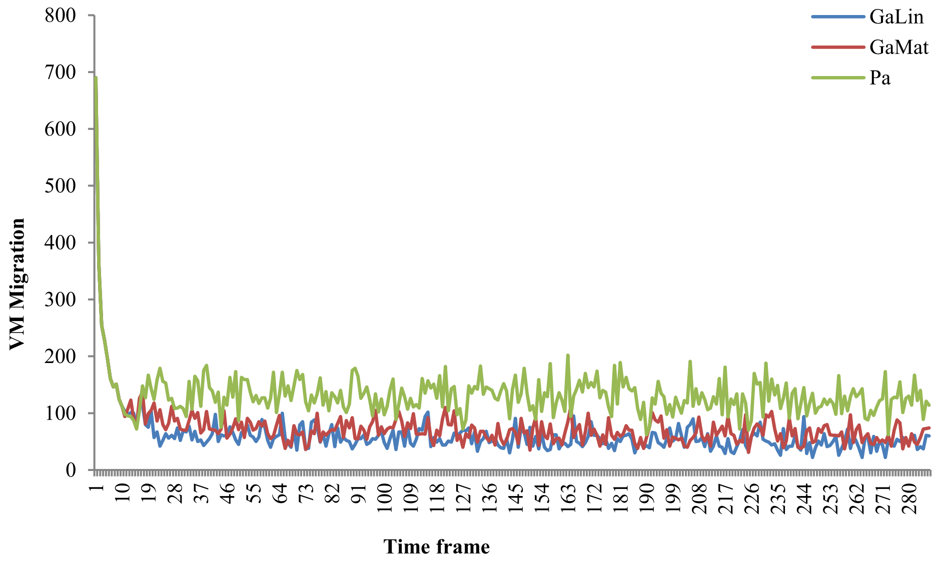


Fig. 10 VM migration in each time frame: Day 04/03/2011

the VM migrations in GaLin are lower than GaMat and Pa in each time interval.

Conclusion

Dynamic consolidation of VMs was done using genetic meta-heuristics in the proposed work. It finds an optimal solution using genetic operators (mutation and crossover) from random solutions. The primary purpose of the proposed work is to investigate use of the GA approach for controlling the EC and service efficiency of cloud datacenter. In addition, GaLin, influenced by research performed in most of the previous works, was also implemented in this work. EC, PM overloading, SLA and VM migration constraints to increase efficiency are mainly addressed by GaMat and GaLin. The fitness criterion is based on predicted utilization which results with significant improvement. However the efficiency may be enhanced by employing other factors, e.g. memory, disk usage, and network resources. Simulation findings indicate that the GA-based solutions have been found better for EC and SLA's autonomous management. These methods can be implemented as future work to address the other resource allocation issues in the cloud datacenter.

Declaration

Conflict of interest. All authors declare that they have no conflicts of interest.

References

1. AazamM, Khan I, Alsaffar AA, Huh E-N. Cloud of things: integrating internet of things and cloud computing and the issues involved. In: Proceedings of IEEE international Bhurban conference on applied sciences & technology (IBCAST), vol. 11; 2014. p. 414–9.
2. Khalaj AH, Scherer T, Halgamuge SK. Energy, the environmental and economical saving potential of data centres with various economizers across Australia. Appl Energy. 2016;183:1528–49.
3. Belady C. Projecting annual new datacenter construction market size. Technical Report. Microsoft Corp., US; 2011.
4. Fiona B, Ballarat C. International review of energy efficiency in data centres acknowledgements; 2021.
5. Koot M, Wijnhoven F. Usage impact on data center electricity needs: a system dynamic forecasting model. Appl Energy. 2021;291:116798.
6. Vijarania M, Gupta S, Agrawal A, Adigun MO, Ajagbe SA, Awotunde JB. Energy efficient load-balancing mechanism in integrated IoT-fog-cloud environment. Electronics. 2023;12(11):2543. https://doi.org/10.3390/electronics12112543.
7. Padmapriya N, Tamilarasi K, Kanimozhi P, Kumar AT, Rajmohan R, Ajagbe SA. A secure trading system using high-level virtual machine (HLVM) algorithm. In: 2022 international conference on smart technologies and systems for next generation computing (ICSTSN). IEEE; 2022. p. 1–4. https://doi.org/10.1109/ICSTSN53084.2022.9761326.
8. Adeniji OD, Ayomide MO, Ajagbe SA. A model for network virtualization with open flow protocol in software define network. In: 4th international conference on intelligent communication technologies and virtual mobile networks: proceedings of ICICV 2022, 10–11 Feb 2022. Springer Lecture Notes on Data Engineering and Communications Technologies, pp 723–33.
9. Fan X, Weber WD, Barroso LA. Power provisioning for a warehouse-sized computer. In: Proceedings of the 34th annual international symposium on computer architecture. New York, USA: ACM; 2007. p. 13–23.
10. Madni SHH, Latiff MSA, Coulibaly Y, Abdulhamid SM. Resource scheduling for infrastructure as a service (IaaS) in cloud computing: challenges and opportunities. J Netw Comput Appl. 2016;68:173–200.

11. Falkenauer E, Delchambre A. A genetic algorithm for bin packing and line balancing. In: Proceedings of the IEEE international conference on robotics and automation, Nice, France; 1992. p. 1186–92.
12. Holland J. Adaptation in natural and artificial systems. Ann Arbor/Cambridge: University of Michigan Press/MIT press; 1992.
13. 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(5):755–68.
14. Voorsluys W, Broberg J, Venugopal S, Buyya R. Cost of virtual machine live migration in clouds: a performanceevaluation. In: Proceedings of the I international conference on cloud computing (CloudCom), vol. 2009. Beijing: Springer; 2009.
15. 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. Concurr Comput. 2012;24(13):1397–420.
16. Hu J, Gu J, Sun G, Zhao T. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In: 2010 3rd international symposium on parallel architectures, algorithms and programming, Dalian; 2010. p. 89–96.
17. Tang M, Pan S. A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers. Neural Process Lett. 2015;41:211–21.
18. Joseph CT, Chandrasekaran K, Cyriac R. A novel family genetic approach for virtual machine allocation. Proced Comput Sci. 2015;46:558–65. ISSN:1877-0509.
19. DengL, Li Y, Yao L, Jin Y, Gu J. Power-aware resource reconfiguration using genetic algorithm in cloud computing. Mobile Inf Syst. 2016;2016:Article ID 4859862, 9 p.
20. Arianyan E, Taheri H, Sharifian S. Multi target dynamic VM consolidation in cloud data centers using genetic algorithm. J Inf Sci Eng. 2016;32:1575–93.
21. Lopez-Pires F, Baran B. Many-objective virtual machine placement. J Grid Comput. 2017. https://doi.org/10.1007/s10723-017-9399-x.
22. Sharma O, Saini H. Performance evaluation of VM placement using classical bin packing and genetic algorithm for cloud environment. Int J Bus Data Commun Netw. 2017;13:45–57. https://doi.org/10.4018/IJBDCN.2017010104.
23. Mosa A, Sakellariou R. Virtual machine consolidation for cloud data centers using parameter-based adaptive allocation. In: ECBS 2017, 5th European conference on the engineering of computer based systems, Larnaca, Cyprus, 31 August–1 September 2017.
24. Yousefipour A, Rahmani AM, Jahanshahi M. Energy and cost-aware virtual machine consolidation in cloud computing. Softw Pract Exp. 2018. https://doi.org/10.1002/spe.2585.
25. Askarizade M, Maeen M, Haghparast M. An energy-efficient dynamic resource management approach based on clustering and meta-heuristic algorithms in cloud computing IaaS platforms: energy efficient dynamic cloud resource management. Wirel Personal Commun. 2018. https://doi.org/10.1007/s11277-018-6089-3.
26. Tseng F, Wang X, Chou L, Chao H, Leung VCM. Dynamic resource prediction and allocation for cloud data center using the multi-objective genetic algorithm. IEEE Syst J. 2018;12(2):1688–99.
27. Kaaouache MA, Bouamama S. An energy-efficient VM placement method for cloud data centers using a hybrid genetic algorithm. J Syst Inf Technol. 2018;20(4):430–45.
28. Abohamama A, Hamouda E. A hybrid energy-aware virtual machine placement algorithm for cloud environments. Expert Syst Appl. 2020;150:113306. https://doi.org/10.1016/j.eswa.2020.113306.
29. Parvizi E, Rezvani M. Utilization-aware energy-efficient virtual machine placement in cloud networks using NSGA-III meta-heuristic approach. Clust Comput. 2020. https://doi.org/10.1007/s10586-020-03060-y.
30. Calheiros RN, Ranjan R, Beloglazov A, Rose CAFD, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. J Softw Pract Exp. 2011;41:23–50.
31. Park KS, Pai VS. CoMon: a mostly-scalable monitoring system for PlanetLab. In: ACM SIGOPS operating systems review; 2006. p. 65–47.