



An efficient resource provisioning approach for analyzing cloud workloads: a metaheuristic-based clustering approach

Mostafa Ghobaei-Arani¹ · Ali Shahidinejad¹

Published online: 23 April 2020

© Springer Science+Business Media, LLC, part of Springer Nature 2020

Abstract

With the recent advancements in Internet-based computing models, the usage of cloud-based applications to facilitate daily activities is significantly increasing and is expected to grow further. Since the submitted workloads by users to use the cloud-based applications are different in terms of quality of service (QoS) metrics, it requires the analysis and identification of these heterogeneous cloud workloads to provide an efficient resource provisioning solution as one of the challenging issues to be addressed. In this study, we present an efficient resource provisioning solution using metaheuristic-based clustering mechanism to analyze cloud workloads. The proposed workload clustering approach used a combination of the genetic algorithm and fuzzy C-means technique to find similar clusters according to the user's QoS requirements. Then, we used a gray wolf optimizer technique to make an appropriate scaling decision to provide the cloud resources for serving of cloud workloads. Besides, we design an extended framework to show interaction between users, cloud providers, and resource provisioning broker in the workload clustering process. The simulation results obtained under real workloads indicate that the proposed approach is efficient in terms of CPU utilization, elasticity, and the response time compared with the other approaches.

Keywords Cloud computing · Workload clustering · Resource provisioning · Gray wolf optimizer · Genetic algorithm · Fuzzy C-means

✉ Mostafa Ghobaei-Arani
m.ghobaei@qom-iau.ac.ir

¹ Department of Computer Engineering, Qom Branch, Islamic Azad University, Qom, Iran

1 Introduction

With rapid developments of Internet-based computing, the cloud computing model has emerged as one of the promising distributed computing technologies to offer the IT resources, such as computational servers, network, storage, and applications to meet the user's quality of service (QoS) constraints through the Internet. The usage of cloud-based applications is significantly increased for performing the activities of daily life in both personal and professional life [1–3]. Therefore, it is necessitated that cloud infrastructure automatically provisioned the cloud resources for executing these cloud-based applications. To this end, cloud resource management is one challenging issue to be addressed. The cloud resource management includes several issues such as resource scheduling, load balancing, resource provisioning, and resource discovery, and resource adaptation [4]. Since users frequently use the cloud-based applications and they may experience workload fluctuations, we focus on the resource provisioning issue to handle their workload changes. The number of resources and the number of users are two of the important factors that affect the provisioning of cloud resources to execute these applications. All the resource provisioning mechanisms are based on analyzing the characteristics and fluctuations the cloud workloads. The resource provisioning mechanisms can dynamically scale up to serve the burst workloads, whereas scale down when workload demands subside [5]. Examples of these workloads include financial services, web services, mobile computing services, graphics-based services, and online transaction processing services. On the other hands, the users submit their demands (i.e., workloads) with various QoS constraints in the form of service level objectives (SLOs) to execute on the cloud infrastructure. Since the submitted cloud workloads by users are heterogeneous in terms of QoS metrics, analysis and identification of them to meet QoS constraints agreed in SLOs can play an important role to provision the cloud resources in a cloud environment. Therefore, it requires allocating or de-allocating cloud resources to serve the heterogeneous cloud workloads for achieving the desirable elasticity at runtime. Although some resource provisioning mechanisms using workload clustering based on QoS metrics have already been investigated [6, 7], still more effort is necessitated for analyzing cloud workloads better.

In this paper, we propose an efficient resource provisioning solution based on metaheuristic-based clustering mechanism to analyze the cloud workloads. The proposed approach utilized a combination of the genetic algorithm (GA) and fuzzy C-means technique [8] for clustering the heterogeneous cloud workloads based on QoS metrics. First, we eliminate the abnormal user requests from incoming workload and then index the accepted user requests for clustering to make the training workloads. Afterward, the training workloads are compared with the test workloads to find the most similar cluster to the current user request. Finally, a gray wolf optimizer (GWO) metaheuristic technique [9] to identify appropriate scaling decisions to provide an efficient resource provisioning solution is utilized.

The main contributions of this study can be summarized as follows:

- Designing an extended framework inspired by the three-tier architecture of the cloud ecosystem to interact between users, cloud providers, and resource provisioning broker.
- Proposing a hybrid solution using the GA algorithm and fuzzy C-means technique for clustering the heterogeneous cloud workloads based on QoS metrics.
- Utilizing a GWO technique to determine scaling decisions for efficient resource provisioning.
- Simulating a set of experiments to validate the effectiveness of our proposed solution under real and synthetic workloads in terms of cost, response time, elasticity, and CPU utilization metrics.

The rest of this paper is organized as follows: In Sect. 2, we review studies related to the workload clustering-based resource provisioning mechanisms. We explain the proposed approach in more detail in Sect. 3. The experimental results through simulations are provided in Sect. 4, and we finally provide the conclusions and future directs in Sect. 5.

2 Related works

Several approaches have been proposed previously to handle the resource provisioning issue using workload analysis in cloud environments.

Gill et al. [10] have developed an extended framework to provision the cloud resource automatically for serving the heterogeneous clustered workloads. Their proposed framework utilized autonomic computing paradigm for self-managing resource to execute cloud-based applications for satisfying QoS requirements. Their solution used workload analyzer component for clustering of heterogeneous cloud workloads using k-means technique and provisioned the required resources according to their QoS requirements. Finally, they validated the proposed framework on the e-commerce application as a cloud-based application and demonstrated that their proposed framework outperforms in terms of execution time, energy consumption, throughput, SLA violation ration, and resource utilization compared with the existing framework. Erradi et al. [11] have proposed a new scheme to predict required resources according to access logs for meeting QoS requirements of web applications. Their proposed method used unsupervised learning to extract the workload latent features to estimate the hardware resource demands such as memory, CPU, and bandwidth utilization and response time for executing varying-time workloads. They validate the proposed scheme with RUBiS and Acme Air application benchmarks under repeated and increasing random workloads and indicated that their proposed scheme outperforms in terms of mean squared error metric compared with the existing schemes. Xu et al. [12] have investigated the outage probability forecasting in mobile multiuser communication systems. They extracted a closed form for the outage probability on the fading channels. Then, they combined gray wolf optimization and neural network to predict the performance of outage probability for generating training data. They validated the proposed solution using Monte Carlo simulation and indicated that their proposed solution outperformed in terms of accuracy metrics compared with

other machine learning-based mechanisms. Besides, in other work [13], they utilized cooperative communications to reduce the bit error probability in mobile IoT network. They also describe closed-form expressions for the direct link signal-to-noise ratio and end-to-end link and investigate the effect of fading channels on the bit error probability metric. Yi-Han Xu et al. [14] have studied the resource allocation issue to maximize energy efficiency in wireless body area networks. They take into account the relay selection, transmission power, and transmission mode to find an efficient allocation decision. Besides, they formulated their problem in the form of Markov decision process and utilized a reinforcement learning technique for reducing the state space and improving the convergence speed. Xu et al. [15] have presented a mobility management approach for device-to-device communication to meet QoS requirements such as latency, power consumptions on the heterogeneous network systems. Their proposed approach extends IEEE 802.21 to improve mobility experience of users on the heterogeneous network environment. Besides, they developed a load-aware mode selection mechanism to select the best target mode.

In [16], a particle swarm optimization-based solution to schedule of both heterogeneous and homogenous workloads on the cloud resources for minimizing the cost, execution time have proposed. The main aims of their proposed solution are: extracting QoS parameters of workloads, clustering workloads using patterns, and k-means-based clustering technique, and resource provisioning classified workloads according to their QoS parameters before resource scheduling. Also, they indicated that their proposed solution avoids over- and under-utilization of cloud resources, and it reduces queuing time, and energy compared with other existing methods. Mian et al. [17] investigated the data analytic workloads for provisioning resources in a public cloud environment. They introduce a framework that includes a cost model to predict the cost of serving a workload on a configuration to specify the most cost-effective configuration for a certain data analytic workload. They validated their proposed framework on Amazon EC2 with data-intensive workloads and demonstrated that their solution minimizes the resource costs while the QoS requirements associated with the workload are satisfied. Iqbal et al. [6] designed a framework for auto-scaling of web applications based on workload patterns prediction. They utilized an unsupervised learning technique to analyze the web application access logs using response time and document size metrics. Besides, they model the web application workload in the form of a probabilistic workload pattern to predict the future workload pattern of the web application using a nonnegative least square technique for future time intervals. They implemented their proposed framework under three real-world web application access logs and indicated that their solution could accurately predict future workload patterns compared to existing methods. Magalhães et al. [18] introduced a web application model to obtain the behavioral patterns of various user profiles for a cloud workload. Their solution models the workload patterns as statistical distributions to represent dynamic cloud environments for supporting and simulating of resources utilization in cloud data centers. Also, they validated their proposed web application model as an extension of the CloudSim toolkit and indicated that their model can generate data to accurately represent various user profiles.

Amiri et al. [19] proposed a prediction-based with capability online learning method for extracting knowledge about the application behavior changes for efficient

resource provisioning in the cloud environment. They utilized a consistency metric to extract the workload patterns to predict the behavior changes of the application. Their simulation results showed that their method learns the new workload behavioral patterns compared with linear regression and neural networks methods. Meenakshi et al. [20] presented an efficient resource provisioning method using k-means clustering and gray wolf optimization (GWO) partitioning technique. They utilized GWO for prioritization and k-means clustering to analyze QoS metrics to allocate cloud resources for serving user requests. Their numerical results illustrated that their method outperforms in terms of clustering accuracy, memory usage, and execution time compared with existing methods. Raza et al. [21] reviewed autonomic workload management in large-scale database management systems and data warehouses. They explore studies related to various domains of workload management, including workload performance prediction, workload adaptation, and workload classification. They used three characteristics autonomic computing, namely self-adaptation, self-prediction, and self-inspection, to select workload management studies on large-scale data repositories.

Liu et al. [22] proposed an adaptive classified technique for workload prediction in a large-scale heterogeneity cloud environment. Their technique classifies the workloads into different patterns according to workload features and then assigned for various prediction models. They transform the workload clustering problem using linear programming model according to prediction accuracy and the predicting time metrics. Further, they validated their proposed technique under Google Cluster trace and demonstrated that their solution reduces prediction errors compared with existing time-series prediction techniques. Singh et al. [23] proposed a classification-based approach for predicting workload patterns of web applications in a cloud environment. Their solution utilized the support vector regression, linear regression, and ARIMA to select the prediction model according to workload features. They evaluated the effectiveness of the proposed solution on the ClarkNet and NASA as two real workload traces and indicated that their approach significantly reduces root-mean-squared error and mean absolute percentage error metrics compared with other time-series prediction approaches.

Generally, most of the current researches only focus on heuristic-based mechanisms with the k-means [10, 17], or unsupervised learning [6, 11] techniques for clustering the heterogeneous cloud workloads for satisfying the QoS requirements. Since the cloud workloads are heterogeneous, combination heuristic-based mechanisms with the other clustering techniques are still not entirely adequate for achieving high clustering accuracy. Therefore, we combine the GA as a metaheuristic approach with fuzzy C-means clustering technique to estimate the hardware resource demands for executing the cloud heterogeneous workloads. Besides, our approach uses preprocessing workload phase to eliminate the abnormal user requests from incoming workload for enhancing clustering accuracy. Although some studies [20] have already been utilized the metaheuristic-based clustering mechanism to address workload clustering based on QoS criteria, still more effort is necessitated for analyzing cloud workloads in an efficient manner.

Finally, we provide a summarization of the most relevant works related to resource provisioning techniques using workload clustering into Table 1 based on

six metrics: (1) utilized technique, (2) performance criteria, (3) policy, (4) method (5) evaluation tool and (6) workload type.

3 Proposed approach

In this section, we present our proposed approach in more details. First, we design an extended framework to interact with users, cloud providers, and resource provisioning broker in the cloud ecosystem. In the proposed solution, users send their requests, and then, their required resources will be allocated. The proposed solution is categorized into three main phases. In the first phase, preprocessing of the workload is fulfilled, which are mainly focused on eliminating noisy and abnormal requests. Then, it is defined as an ID for each request which is used for creating an SLA table according to these requests. In the second phase, the workload is clustered by the GA algorithm and fuzzy C-means technique. Then, the closest center of the cluster from the test workload is selected. Finally, resource scaling decisions are carried out by GWO algorithm in the third phase.

3.1 Proposed framework

The framework of the proposed solution is depicted in Fig. 1. As it is depicted in Fig. 1, the framework is categorized into three main entities: management of the workload resources, the users, and the cloud. Resource provisioning broker (RPB) is the intermediate level of the proposed method to provision required cloud resources and services of the users from data centers of cloud providers (CPs). These data centers include one or more virtual machines acting as the main processing elements, each of which has its particular processing resources. The results of each request are finally returned to the user. Each service is running on a particular VM, and incoming requests of these VMs and services are variable and depend on incoming workload traffic. Services to these VMs have one of the two statuses: allocation, not-allocation. Each CP which is arrived in the cloud ecosystem has to register itself in Cloud Information Service (CIS). Firstly, a query is created by RPB and is sent to CIS requesting CP's name to execute the user's requests. For example, if the user request is determined to be CP1, RPB sends requests to this provider. On each CP, there are predefined enough VMs to execute the incoming workload requests according to the policy. After completing the tasks, these VMs are deallocated, and the results are sent back to the requesters or users.

For solving the cloud resources provisioning problem in the proposed framework, if the required processing resources are estimated and calculated, it will be possible to identify the similarity between the current service request and the frequent cases for determining required VMs and processing resources to the manager of cloud computing resources. Specifically, the worst possible cases of demands could be considered to ignore the minor differences between the new incoming request and the most frequent requests.

As another important problem, since it is not possible to identify the amount of required resources for executing requests precisely while service requests are

Table 1 Survey of studies related to workload clustering-based resource provisioning techniques

Ref.	Utilized technique	Performance criteria	Policy	Method	Benchmark	Workload
Iqbal [6]	Unsupervised learning-based clustering + NNLS + PWP	Response time, occupancy probability, residual error	Proactive	Replication	CloudSim toolkit + RUBiS benchmark	Safety, News link, and DAR workloads
Gill [10]	Heuristic-based + K-means clustering	Submission burst time, total execution time, network usage, serviceability	Proactive	Replication	CloudSim toolkit	Real workloads
Erradi [11]	Unsupervised learning-based clustering	Response time, CPU utilization, memory utilization, bandwidth utilization, MSE	Proactive	Replication	Amazon EC2 + RUBiS, Acme Air benchmarks	Web-based workloads
Singh [16]	PSO + K-means clustering	Execution time, execution cost, energy consumption, resource utilization, reliability, availability, latency	Reactive	Replication	CloudSim toolkit	Real web-based workloads
Mian [17]	Heuristic-based search algorithm	Cost, cost/performance	Proactive	Replication/resizing	Amazon EC2 + benchmark factory	OLAP workload
Magalhães [18]	Statistical analysis	Response time, CPU utilization, memory utilization, disk utilization	Proactive	Resizing	CloudSim toolkit + RUBiS benchmark	Web-based workloads
Amiri [19]	Online learning + pattern mining	Precision, covering percentage	Proactive	Replication/resizing	CloudSim toolkit	Synthetic workloads
Meenakshi [20]	GWO + K-means clustering	Accuracy, execution time	Reactive	Replication	CloudSim toolkit	Synthetic workloads
Liu [22]	Mixed integer linear programming	CPU usage, prediction time, MRPE	Proactive	Replication	CloudSim toolkit	Google cluster workload
Singh [23]	ARIMA + SVR + LR prediction models	MAE, MSE, RMSE and MAPE	Proactive	Replication	MATLAB	NASA and ClarcNet workloads

Table 1 (continued)

Ref.	Utilized technique	Performance criteria	Policy	Method	Benchmark	Workload
Our work	GA/fuzzy C-means clustering + GWO	Execution time, execution cost, energy consumption, latency, SLA violation rate	Reactive	Replication	CloudSim toolkit	Web-based workloads

MAE mean absolute error; *MAPE* mean absolute percentage error; *RMSE* root-mean-squared error; *MSE* mean square error; *MRPE* mean relative percentage error; *OLAP* OnLine analytical processing; *NNLS* nonnegative least square; *PWP* probabilistic workload pattern; *DAR* daily activity reporting; *RUBiS* rice university bidding system; *PSO* particle swarm optimization; *GWO* gray wolf optimization; *GA* genetic algorithm; *LR* linear regression; *SVR* support vector regression; *ARIMA* auto-regressive integrated moving average

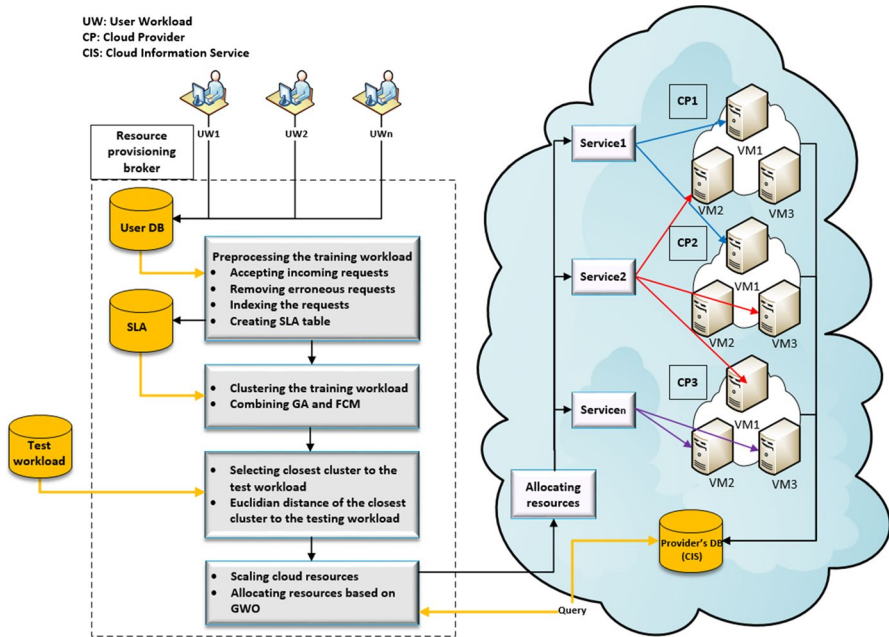


Fig. 1 The proposed Framework

coming, calculating the amount of processing resources is hard and impracticable. In addition, it is time-consuming and demanding huge and precise amount of calculation, which is out of the real-time cases. This problem leads us to utilize clustering in which the required calculation could be postponed to the other times in the future; therefore, the results could be saved to be used later.

According to the proposed framework, after accepting requests, the sequence of removing abnormal requests, indexing these requests, and creating SLA table is fulfilled. This table is next saved for future search and reference. Then, users' requests are clustered for the training requests. Finally, the test workload is compared with the training workload to find and select the most similar cluster for the current request using Euclidian distance approach. Based on the prior calculations, the amounts of required processing resources are determined using GWO algorithm. Noteworthy, to have a decent clustering and good samples, the state space (i.e., different requests) should be covered well as the history of clustering. More the request samples better and further the coverage of the state space.

3.2 Problem formulation

In this section, we present the required notations in the proposed approach, as shown in Table 2.

Let's $Request = \{Req_1, Req_2, \dots, Req_N\}$ be the N incoming requests of the workload. The SLA represents the structure's name for accepting service of each request. This structure consists of response time, cost, availability, and reliability. $Train_new$

Table 2 Variables and their description

Variable	Description
N	Number of requests
M	Number of request's fields
Req_i	i -th request
$Train_req$	Training workload requests
$Test_req$	Test workload requests
$Train_New$	Processed workload requests
SLA	Service level agreement
$SLA_i \cdot Cost$	Cost of i -th request based on SLA
$SLA_i \cdot Res$	Response time of i -th request based on SLA
$SLA_i \cdot Av$	Availability of i -th request based on SLA
$SLA_i \cdot Rel$	Reliability of i -th request based on SLA
$Penalty$	Payment rate in the case of deviation from SLA
$Penalty_Rate$	Exponential coefficient of penalty
K	Number of VMs
$VM_k \cdot Cost$	The cost of using k -th VM
$VM_k \cdot MIPS$	MIPS of k -th VM
$VM_k \cdot RAM$	RAM of k -th VM
$VM_k \cdot Storage$	Storage of k -th VM
VM_Init_Cost	Configuration cost of new VM
VM_Boot_Cost	Initializing cost of new VM
VM_UP_Cost	Cost of adding extra resources to the new VM
$Req_x \cdot SLA$	Number of deviation from SLA for i th request
$Req_x \cdot Penalty$	Total penalty of request x
$Req_x \cdot benefit$	Total benefit of request x
$Cost$	Total cost
$Penalty$	Total penalty
$Response$	Total response time
$Availability$	Total availability
$Center$	Center of cluster

holds processed noiseless requests. The variable α holds the number of constructed clusters with the cluster center of $Center$ which is defined as $Center = \{center_1, center_2, \dots, center_\alpha\}$. RC_j^i shows the requests belonging to each cluster in which i is the number of cluster with request number of j . Also, $Dis(i, j)$ defines Euclidian distance between requests i and j .

For each violation from the SLA, different punishment is kept in $Penalty$ with the exponential coefficient for next violations, $Penalty_rate$. In addition, $VM = \{VM_1, VM_2, \dots, VM_k\}$ is k virtual machines with different service specifications such that $VM_k \cdot MIPS$, $VM_k \cdot Cost$, $VM_i \cdot RAM$, and $VM_i \cdot Storage$ represent Million Instruction Per Second (MIPS) of i -th VM, the cost of using i -th VM, RAM of i -th VM, and storage of i -th VM, respectively. When adding a new VM, the total cost consists of the summation of VM normal usage cost and booting cost, VM_Boot_Cost . Besides, in

the case of needing new configuration, this new configuration demands extra costs for memory, storage, or MIPS that is added to reconfiguration costs, VM_Init_Cost .

3.3 Proposed algorithm

In this section, the proposed algorithm is presented. In the proposed model, the structure of workloads is as follows: user's IP, the date and time of the request, the type of requested service identified by the user, the protocol of information transferring, cost, response time, availability, and reliability predefined in SLOs.

3.3.1 Workload preprocessing phase

Workload preprocessing is one of the most important steps for achieving the results of research more precisely and completely. In this respect, different techniques are used to remove noisy data.

Workload preprocessing is depicted in Fig. 2. As it is shown, the requests will be processed in the first step. In this step, the unnecessary requests are removed from the incoming workload, which includes lack of user access right for the requested service, the incomplete fields of request (e.g., null field), and the fake user as a robot to search, propagandize, and desolate purposes.

For the ease of Availability, a unique number named *Request ID* is defined for each request. This unique number will help in creating SLA table for each request, as shown in Table 3.

Fig. 2 The workload preprocessing phase

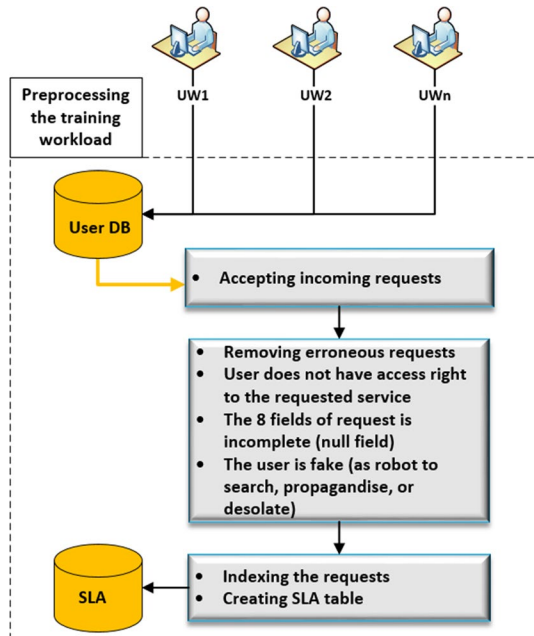


Table 3 SLA structure for requests

Request's ID	User SLA			
	SLO ₁	SLO ₂	SLO ₃	SLO ₄
	Cost	Response time	Availability	Reliability

The pseudo-code for workload preprocessing in the proposed algorithm is depicted in Algorithm 1. As it is shown, N training workloads are entered to the algorithm, each of which has M attributes (line 1). According to the above-mentioned reasons, incoming requests will be removed under certain conditions (lines 4–12). A new array is defined for indexing each workload (lines 13–16). Finally, the SLA table is created for all preprocessed workloads (line 17).

Algorithm 1: Pseudo code the workload preprocessing

```

1- Input: Train_req = {Req1, Req2, ..., ReqN} {elem1, elem2, ..., elemM} // set of Train_workload
2- Req_elements_Count ← M
3- for i ← 1 to N do
4-   if (Train_reqi is searcher_robot or propagandise_robot or desolate_robot)
5-     delete Train_reqi
6-   if (Train_reqi has not authorization_availability)
7-     delete Train_reqi
8-   for j ← 1 to Req_elements_Count do
9-     if (Train_reqi == null)
10-      delete Train_reqi
11-   end for
12- end for
13- for i ← 1 to len (Teach_req) do
14-   Train_newi = Train_reqi
15-   Train_newi.elem0 = i
16- end for
17- SLA ← Train_new // select many elements of each Train_new to create a SLA table

```

3.3.2 Workload clustering phase

In this section, we describe the workload clustering as one of the most important phases of the proposed approach to provide an efficient resource provisioning solution. To have proper clusters, a combining GA and fuzzy c-mean is utilized for clustering the training workload, as shown in Fig. 3. The workload clustering phase consists of four step, namely: (A) Generating initial population and identifying the number of clusters, (B) calculating the value of each request, (C) applying genetic operators, and (D) termination condition.

A. Generating initial population and identifying the number of clusters

In the proposed algorithm, each initial chromosome (i.e., request) includes two vectors: the first vector, H , which indicates the number of cluster center's requests, and the second one holds the number of sub-clusters for each cluster in α . Indeed,

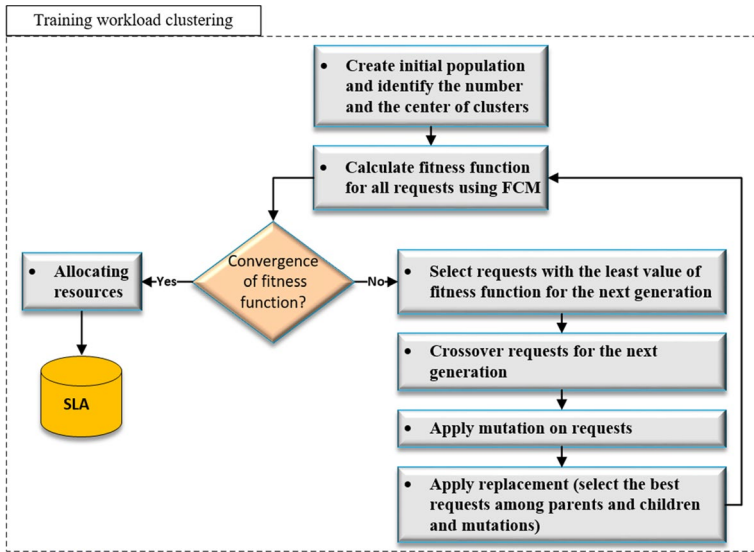


Fig. 3 The training workload clustering using GA and fuzzy c-mean

the second vector holds α elements under the centrality of clusters' requests. Each element of this vector has one ID of the request. Creating such vector and selecting ID of cluster center's candidate requests is random. Therefore, cluster C_i in the proposed algorithm with sub-cluster X_i has the following structure:

$$C_{One} = [C_1, C_2, \dots, C_H] [X_1, X_2, \dots, X_\alpha, \dots, XH_{\alpha*c}] \tag{1}$$

B. Calculating the value of each request

In this section, we select the best requests (i.e., the best parents for generating children) using an objective function. Obviously, the objective function in fuzzy c-mean clustering is as follows:

$$J_m(U, V) = \sum_{i=1}^C \sum_{k=1}^n u_{ik}^m x_k - v_i^2 \tag{2}$$

$$\sum_{i=1}^c u_{ik} = 1. \quad \forall k = 1, \dots, n$$

In Eq. (2), n data are divided into c clusters to meet the following condition.

$$M_{fcm} = \left\{ U \in R^{c*n} \mid \forall i \cdot k : 0 \leq \mu_{ik} \leq 1; \sum_{i=1}^c \mu_{ik} = 1; 0 < \sum_{k=1}^n \mu_{ik} \right\} \tag{3}$$

To optimize the function $J_m(U, V)$, optimization algorithm estimates U and V in two steps such that clusters' center in step r is calculated regarding U in step $(r - 1)$ th, as follows:

$$v_i = \frac{\sum_{k=1}^n u_{ik}^m x_k}{\sum_{k=1}^n u_{ik}^m} \tag{4}$$

$$u_{ik} = \sum_{j=1}^c \left(\frac{x_k - v_i}{x_k - v_j} \right)^{-2/(m-1)}$$

Since the algorithm continues until satisfying the condition $U_{i+1} - U_i$, the fuzzy c-mean algorithm always converges. Therefore, the fitness function is defined as follows:

$$F = \sum_{i=1}^H \sum_{k=1}^{\infty} u_{ik}^m \sum_{z=1}^4 (X_k \cdot SLO_z - C_i \cdot SLO_z), \quad m > 1 \tag{5}$$

Generally, the main purpose of this phase is to minimize the fuzzy c-mean function by utilizing GA, thereby identifying clusters of all requests. In other words, less distance from clusters' center, more qualification of the chromosome and more the probability of its selection.

C. Applying genetic operators

We use the standard operators such as selection, single-point crossover, mutation operators for changing clusters' center and replace requests in the clusters. For selecting the parent (i.e., superior request), the number of parents for the next generation should be generated.

Crossover operation For this purpose, the single-point crossover method is utilized such that a random point for the crossover is selected as follows:

$$P_{\text{new}} = \beta P_{m\alpha} + (1 + \beta) P_{d\alpha} \tag{6}$$

Mutation operation At first, a random number between 1 and 4 is generated, and based on this generated random number, the values of SLO_i will be changed. For example, if this generated number is 3, SLO_3 of the selected request will be changed.

Since GA improves its generation by repetition, it is better to reduce mutation by increasing repetitions. This process is simulated by applying an exponential function as follows:

$$\mu = \mu_0 e^{-\text{Counter}} \tag{7}$$

where $\mu_0 = 0.2$ and $e = 2.718218284$.

D. Termination condition

If the condition $U_{i+1} - U_i$ is satisfied, then the algorithm is terminated; otherwise, a new population derived from previous population, children, and mutations will be selected. Next, the fitness value for the new population will be calculated.

Pseudo-code for the proposed algorithm is depicted in Algorithm 2. Generating initial population and identifying the number and also the center of clusters is carried out in lines 2–9. For each workload, the belonging matrix is constructed (lines 10–13). Then, the fitness value of each workload is calculated using the objective function of fuzzy c-mean clustering (lines 14–24). After calculating fitness for all requests, determined numbers of those requests with higher fitness values are selected (line 25). Crossover of workloads, mutations, and as the final step, selecting the best cases among parents, children, and mutations workload are fulfilled (lines 26–28). Finally, the termination condition is checked. If the result of checking is positive, then the loop will be repeated (lines 29–31); otherwise, the clustering training workload phase is terminated.

Algorithm 2: Pseudo code of GA_FCM-Cluster

```

1- Input:  $Train\_new = \{Req_1, Req_2, \dots, Req_k\} \{elem_1, elem_2, \dots, elem_m\}$  // set of  $Train\_new$  workload to
   be clustered
2-  $cl$  // number of clusters
3-  $a$  // count of sub-cluster in clusters
4-  $m$  // parameter of fuzzy_cluster
5-  $R\_lim$  // limitation of repeat
6- Output:  $C = \{c_1, c_2, \dots, c_{cl}\}$  // set of cluster centroids
7- for each  $Train\_new_i \in Train\_new$  do
8-    $c_j \leftarrow Train\_new_i, c_j \in C$  //  $c_j$  is selected randomly
9- end for
10- for each  $Train\_new_i \in Train\_new$  do
11-   Evaluate  $U_i = \{u_1, u_2, \dots, u_k\}$  // matrix of the population
12-    $u = \{(u | 0 \leq u \leq 1, \sum(u_{k,c}) = 1)\}$ 
13- end for
14- do {
15-   for  $i = 1$  to  $cl$  do
16-     for  $k = 1$  to  $a$  do
17-        $Sum_1 += U_{ik} \wedge m$ 
18-     end for
19-      $Sum_2 += Train\_new_k - c_i$ 
20-     for  $z = 1$  to  $M$  do
21-        $Sum_3 = Sum_1 + Sum_2$ 
22-        $Sum = Sum + Sum_3$ 
23-     end for
24-   end for
25-   Select count of  $Train\_new$  for Crossover
26-   Crossover_N_Point of  $Train\_new$ 
27-   Mutation_Inversion_Permutation N_ount of  $Train\_new$ 
28-   Composition k number of crossover and mutation and basic population of  $Train\_new$ 
29-   Evaluate  $U_{i+1}$  matrix of the population
30-    $U\_value = \|U_{i+1} - U_i\|$ 
31- While ( $U\_value \leq R\_lim$ )

```

3.3.3 Resource provisioning phase

In this section, we utilized a gray wolf optimizer (GWO) metaheuristic technique to identify appropriate scaling decisions to provide an efficient resource provisioning solution. After clustering training workloads, it requires to select the closest cluster to the test workload. To this end, each request of test workload is compared with each cluster centroid using Euclidian distance. The distance of the request i from the cluster centroid j is calculated as follows:

$$Dis_{i,c_j} = \sqrt{\sum_{k=1}^4 (Req_i \cdot SLO_k - C_j \cdot SLO_k)^2} \quad (8)$$

After finding the closest cluster to the user request, resource provisioning process should be carried out. In this regard, three resource scaling decisions are considered as follows:

- (a) *Adding resources* In this decision, the numbers of VMs are not adequate for serving user requests. Therefore, new VMs should be turned on.
- (b) *Removing resources* In this decision, the numbers of VMs are more than the need for serving user requests. Therefore, some VMs should be turned off.
- (c) *Balanced resources* In this decision, the numbers of VMs are adequate for serving user requests, and there is no need to change the numbers of VMs.

In the following, we describe the resource provisioning algorithm using GWO technique in more details, as shown in Fig. 4. The resource provisioning (i.e., resource scaling) phase consist of four steps, namely: A) initialization, B) generating initial population, C) calculating the value of wolves, and D) replacing wolves and termination condition.

A. Initialization

As it is mentioned earlier, it should be identified the constant coefficient of vectors A and D , random value r , iteration numbers of the algorithm, and linear value a , then it is selected some incoming test workload randomly.

B. Generating initial population

Since there are cl clusters, the best cluster should be selected for the test workload. First, each cluster is identified as a wolf utilizing WGO. It is proposed an array of bits as the place array of the wolves, X . Initial population is filled randomly, and each wolf has some 0 s or 1 s. This array is defined as follows:

$$X = [bit_1, bit_2, \dots, bit_{Nvar}] \quad (9)$$

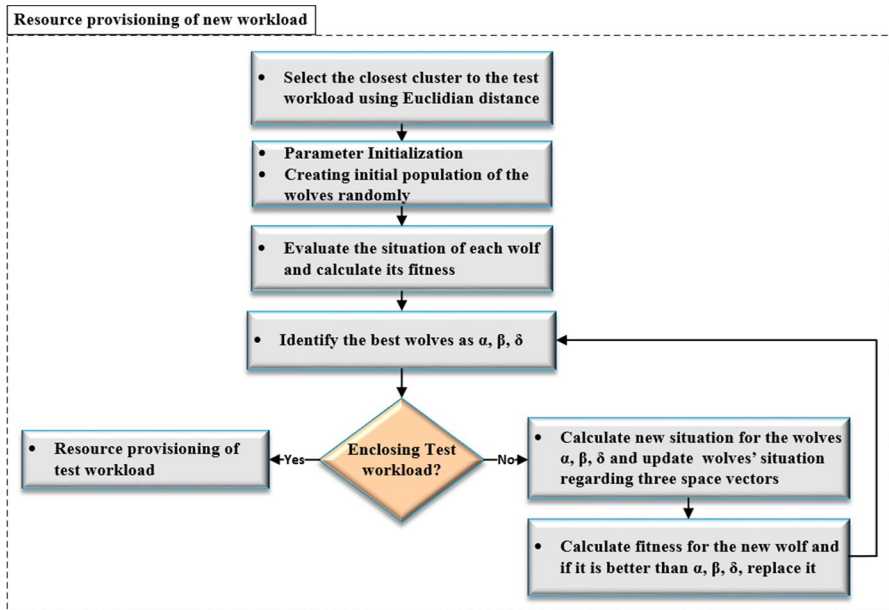


Fig. 4 Resource provisioning of test workload using GWO

Table 4 The selection of requests for the wolves

Request's ID	Request 1	Request 2	...	Request Nvar
Bit selection	0	1	...	1
	Unselected	Selected	...	Selected

Table 5 The example of the wolf's structure with 14 requests

1	1	0	0	1	0	1	0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

where *Nvar* is the number of solutions for the problem such that if the value 0 is produced, the request with this index is ignored, else if the value of the index is 1, the request will be selected. Table 4 illustrates the selection of requests for the wolves.

As an example, if the cluster has 14 requests, the initial population includes 14 bits filled with 0 s or 1 s. If the bit has the value 1, the requests with this index will be selected and will be considered in the selection operator; otherwise, the request will be ignored. The requests are selected randomly. If the requests, for example, 1, 2, 5, 7, 11, and 14, are selected, the created wolf will be as shown in Table 5.

Table 6 An example of the movement of the wolves and identifying their locations

Current location of the wolf	1	1	0	0	1	0	1	0	0	0	1	0	0	1
The wolf α	0	0	1	1	1	0	1	0	0	0	0	1	1	0
The wolf β	1	0	1	1	0	1	0	0	0	1	0	0	1	1
The wolf δ	1	1	0	1	1	0	1	0	1	0	1	0	0	0
New location of the wolf	0	1	0	0	1	0	1	0	0	0	1	0	0	0

C. Calculating the value of the wolves

In this step, the average Euclidian distance of the test workload’s request is calculated, as follows:

$$F = \sum_{j=1}^T \sqrt{\sum_{k=1}^4 (Req_i \cdot SLO_k - C_j \cdot SLO_k)^2} \tag{10}$$

where T is the elements of the fitness function for each wolf.

Regarding the calculated fitness function (i.e., average Euclidian distance of the test workload and each wolf), the first three ones are selected as the wolves α, β, δ in the sorted list.

D. Replacing wolves and termination condition

Evaluating and identifying the position of each wolf (i.e., request) are calculated as follows. Besides, the position of the wolves (i.e., clusters) keeps changing until the victim (i.e., test workload) is enclosed.

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_p(t) - \vec{X}(t) \right| \tag{11}$$

$$\vec{X}(t + 1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D} \tag{12}$$

where \vec{A} and \vec{C} are the coefficient vectors, and \vec{X}_p, \vec{X} , and t are the cluster’s location, location vector of each request, and iteration number, respectively. Also, \vec{A} and \vec{C} are calculated as follows:

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \tag{13}$$

$$\vec{C} = 2\vec{r}_2 \tag{14}$$

where vectors of \vec{a} are decreased from 2 to 0 linearly and iteratively, and \vec{r}_1 and \vec{r}_2 are random vectors in the interval [0, 1].

An example of the movement of the wolves and identifying new location are depicted in Table 6.

Among the sorted requests, the first three ones are selected and labeled as α , β and δ . These three requests can estimate cluster's location at each iteration, as follows:

$$\vec{D}_\alpha = |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}|, \quad \vec{D}_\beta = |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}|, \quad \vec{D}_\delta = |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}| \tag{15}$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot (\vec{D}_\alpha), \quad \vec{X}_2 = \vec{X}_\alpha - \vec{A}_2 \cdot (\vec{D}_\alpha), \quad \vec{X}_3 = \vec{X}_\alpha - \vec{A}_3 \cdot (\vec{D}_\alpha)$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}$$

In every iteration, the locations of the other wolves are updated after identifying the location of α , β and δ . Besides, \vec{A} , \vec{C} , and \vec{a} are updated as well. Finally, the location of the wolf α is considered as the optimum point.

Algorithm 3 depicts the pseudo-code for the GWO-based resource provisioning algorithm. After initializing the parameters (lines 1–11), the cluster with the least Euclidian distance compared with the test workload is selected to provision the resources for the test workload using GWO algorithm (lines 12–15). Afterward, the fitness value for the population and the location of each wolf is calculated, and finally, this location will be sorted (lines 16–24), and the best wolves are identified as α , β and δ . (line 25). By varying directions and speed of wolves, three new locations are generated, and fitness value for the new wolves is recalculated, and the results are re-sorted (lines 26–39). If the fitness value for the new wolf is better than the wolf α , the new wolf is replaced with the wolf α ; otherwise, it is compared with the wolves β , and δ and replacement is performed in its case (line 40). Then, updating the required variables for fitness function, changing the location, and updating the counters are fulfilled (lines 41–43). If the termination condition is not satisfied, the above-mentioned steps will be repeated (line 44) which is held in *Max-Iter*. Finally, the algorithm returns the first wolf as the best one (line 45) and thereby selected resource scaling decisions (i.e., adding, removing, and balanced) for the test workload on VMs will be run.

Algorithm 3: Pseudo code for resource provisioning using GWO

1. **Input:** $Test_req = \{Req_1, Req_2, \dots, Req_k\} \{elem_1, elem_2, \dots, elem_M\}$ //set of test workload to be clustered
2. cl //number of clusters
3. a //counting sub-clusters in clusters
4. $C = \{C_1, C_2, \dots, C_{cl}\}$ //set of cluster's centroids
5. **Output:** $Scale = \{Up, Nor, Down\}$ //set of scales
6. $l = one\ of\ the\ big\ numbers$
7. **Initialize** a, A, r_1, r_2 and S
8. $MaxIter$ //limitation of iterations
9. $Agent = \{Agent_1, Agent_2, Agent_{agent_count}\}$
10. $t = 0$
11. $C_test = \{C_1, C_2, \dots, C_{cl}\}$ // cluster of workload
12. **foreach** $Test_Req_i \in Test_Req$ **do**
13. $minDist \leftarrow MinDistance(Test_Req_i, c_j) \ j \in \{1 \dots cl\}$
14. $C_Test_Req_i \leftarrow c_j$
15. **end for**
16. **foreach** $Test_Req_i \in Test_Req$ **do**
17. **for** $i \leftarrow 1$ to $Agent_count$ **do**
18. $Agent[i].pos \leftarrow Random(1, size(C_Test_Req_i))$
19. $Agent[i].rank \leftarrow Fitness(Agent[i].pos, req)$
20. **end for**
21. **end for**
22. $P_F \leftarrow Sort(Agent)$
23. $P_F \leftarrow Move(p_F)$
24. $P_F \leftarrow Sort(p_F)$ with rank
25. $X = \{1, 2, 3\} = the\ best\ first\ three\ P_F$
26. $i = 1$
27. **Do**{
28. $A = 2a, r_1 - a$
29. $S = 2, r_2$
30. **foreach** X
31. $D_vector \leftarrow Change_D(X, t, S)$
32. **end for**
33. **for** $j = 1$ to $Agent_count$
34. $P_F[j].pos \leftarrow position_newj(D_vector, t, A)$
35. $P_F[j].rank \leftarrow Fitness([P_F[i].pos], req)$
36. **end for**
37. $P_F \leftarrow Sort(P_F)$
38. $P_F \leftarrow Move(p_F)$
39. $P_F \leftarrow Sort(P_F)$ with rank
40. $X = \{1, 2, 3\} = the\ three\ of\ best\ P_F$
41. **Update** a, r_1, r_2
42. $t = t + 1$
43. $i = i + 1$
44. **While** $(i \leq MaxIter)$
45. **Return** $P_F[1].Scale$

3.3.4 An example of the use of the proposed approach

In this section, an example for better understanding the proposed approach in more details is provided.

A. Workload preprocessing phase

At first, the request structure and its correctness is checked. The structure of the incoming workload is depicted in Table 7.

(a) Removing the abnormal requests

As it is mentioned earlier, the unnecessary and abnormal requests are removed from the incoming workload based on the predefined rules. As it was shown in Table 6, the user with ID 82 has not the access right to the requested service, and the users with ID 74856 and also 510 have not filled the requested fields correctly and the users 74856 is recognized as a robot. The results of this process are depicted in Table 8.

(b) Creating the SLA table

For each request, the SLA table is created. An example of the SLA table for the requests of Table 6 is shown in Table 9.

B. Workload clustering phase

In the following, the clustering process for this example is described briefly.

(a) Generating an initial population and identifying the number of clusters

The SLA table is considered as the initial population. If the numbers of clusters are 3 each of which containing three sub-clusters, the first and the send vectors will contain 3 and 9 elements. It means that the second vector contains $3 * 3$ elements of the requests. Three elements are selected randomly as the candidates of the clutters' centroid, such that each X_i is its sun-cluster as well.

$$C_{\text{One}} = [C_1, C_2, C_3] [X_1, X_2, \dots, X_6, \dots, X_9]$$

Table 10 depicts the example of 13 requests and 3 clusters.

(b) Calculating the value of each request

By applying the parameter $m=2$ in Eq. (10) and utilizing the second vector of the requests (i.e., initial population), the belonging matrix U is initialized as randomly as it is depicted in Table 11.

Table 7 An example of incoming workload

User ID	The date and the time of the request	Type of user's requested service	Data transferring protocol	The required cost	The required response time	The required availability	The required reliability
1000	-20182004 14:25	3	4	150	50	20	30
2458	-20182004 14:50	2	2	300	100	33	50
3352	-20182004 15:34	1	4	250	150	44	30
424	-20182004 10:33	5	3	360	330	20	40
155	-20182004 10:22	12	4	500	400	100	60
66	-20182004 08:20	5	4	400	420	70	70
3000	-20182004 15:30	1	4	230	140	42	33
74856	-20182004 11:17	NULL	2	1500	350	5	10
140	-20182004 10:20	12	4	490	395	99	62
82	-20182004 10:20	6	2	100	30	10	80
9	-20182004 06:30	7	1	120	40	20	60
401	-20182004 10:30	5	3	340	320	25	42
142	-20182004 10:15	12	4	520	390	110	58
56	-20182004 08:18	5	4	410	400	65	60

Table 7 (continued)

User ID	The date and the time of the request	Type of user's requested service	Data transferring protocol	The required cost	The required response time	The required availability	The required reliability
510	-20182004 10:52	1	NULL	150	55	60	30

Table 8 After removing the abnormal requests

User ID	The date and the time of the request	Type of user's requested service	Data transferring protocol	Intended cost	Intended response time	Intended availability	Intended reliability
1000	-20182004 14:25	3	4	150	50	20	30
2458	-20182004 14:50	2	2	300	100	33	50
3352	-20182004 15:34	1	4	250	150	44	30
424	-20182004 10:33	5	3	360	330	20	40
155	-20182004 10:22	12	4	500	400	100	60
66	-20182004 08:20	5	4	400	420	70	70
3000	-20182004 15:30	1	4	230	140	42	33
140	-20182004 10:20	12	4	490	395	99	62
9	-20182004 06:30	7	1	120	40	20	60
401	-20182004 10:30	5	3	340	320	25	42
142	-20182004 10:15	12	4	520	390	110	58
56	-20182004 08:18	5	4	410	400	65	60

Table 9 Indexing each request and creating the SLA table

Request's ID	User SLA			
	SLO ₁	SLO ₂	SLO ₃	SLO ₄
	Cost	Response time	Availability	Reliability
1000	150	50	20	30
2458	300	100	33	50
3352	250	150	44	30
424	360	330	20	40
155	500	400	100	60
66	400	420	70	70
300	230	140	42	33
140	490	395	99	62
9	120	40	20	60
401	340	320	25	42
142	520	390	110	58
56	410	400	65	60

Table 10 The example of 13 requests and 3 clusters

Vector of the cluster's centroid	C1			C2			C3		
	424			3000			9		
Sub-cluster's vector	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
	2458	66	3352	1000	155	140	401	142	56

Table 11 Evaluating each request of the example

Chromosome	Belonging matrix U								
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
Cluster 1	0.1	0.3	0.4	0.3	0.6	0.1	0.5	0.7	0.8
Cluster 2	0.6	0.2	0.3	0.6	0.2	0.5	0.2	0.1	0.1
Cluster 3	0.3	0.5	0.3	0.1	0.2	0.4	0.3	0.2	0.1

Table 12 The minimization function of least distance from the clusters' center for all parent requests

Request	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
Fitness function	1039	1620	951	1338	1950	1970	881	2103	1497

Table 13 Selected requests for crossover

Request's ID	User SLA			
	SLO ₁	SLO ₂	SLO ₃	SLO ₄
	Cost	Response time	Availability	Reliability
3352	250	150	44	30
401	340	320	25	42

As it is mentioned earlier, each request's belonging degree of all clusters is presented by matrix U . Therefore, the minimization function of least distance from the clusters' center for all parent requests is shown in Table 12.

The objective is to minimize the fuzzy c-means function using GA. It means that smaller the distance of each request from the clusters' center will result in better fitness of the request and greater the probability of its selection.

(c) Applying genetic operators

First, the requests are sorted by their fitness values in ascending order; some identified numbers of requests are selected for crossover operation to achieve the next generation. The requests third and eighth of Table 8 are selected to make Table 13.

By selecting a point randomly for crossover operation and utilizing Eq. (6) with $\beta=0.6$ and intended parameter SLO_3 , crossover the two requests for the next generation is depicted in Table 14.

To improve the next generation by iteration, Eq. (7) is solved. In this example, three requests are selected randomly in which the least distance from the clusters' center is selected. For mutation operation, we set iteration number $\mu = 1$. Therefore, a random number between 1 and 4 is produced and based on this produced number, and the SLO_i is changed. For example, if the produced number is 3, then the SLO_3 of the selected requests will be changed. Based on this description, the results of the mutation are depicted in Tables 15 and 16.

As the example, ten requests from the parent population, the next generation, and mutants are selected and depicted in Table 17.

(d) Termination condition

The quality of clustering depends on increasing the most similarity among the members of a cluster and the least similarity among the members of a cluster compared with the other clusters. In each iteration, the belonging matrix U is calculated, and then, this matrix is compared with the matrix of the last step regarding

Table 14 The next generation by crossover the parents' requests

Request's ID	User SLA			
	SLO_1	SLO_2	SLO_3	SLO_4
	Cost	Response time	Availability	Reliability
3352	250	150	37	30
401	340	320	32	42

Table 15 Selecting requests for mutation

Request's ID	User SLA			
	SLO_1	SLO_2	SLO_3	SLO_4
	Cost	Response time	Availability	Reliability
66	400	420	70	70
142	520	390	110	58
56	410	400	65	60

Table 16 Mutant requests

Request's ID	User SLA			
	SLO_1	SLO_2	SLO_3	SLO_4
	Cost	Response time	Availability	Reliability
66	400	420	60	70
142	520	390	95	58
56	410	400	60	60

Table 17 Obtained population by running the algorithm in the first step

Request's ID	User SLA			
	SLO ₁	SLO ₂	SLO ₃	SLO ₄
	Cost	Response time	Availability	Reliability
1000	150	50	20	30
2458	300	100	33	50
3352	250	150	37	30
424	360	330	20	40
155	500	400	100	60
66	400	420	60	70
3000	230	140	42	33
140	490	395	99	62
9	120	40	20	60
401	340	320	25	42
142	520	390	95	58
56	410	400	60	60

$U_{i+1} - U_i$. If this condition is satisfied, the algorithm is finalized; otherwise, it is continued until satisfaction.

C. Resource provisioning using GWO algorithm

In this phase, each request of test workload is compared with each cluster's center using Euclidian distance. This process is fulfilled by utilizing Eq. (8). Therefore, SLA indices of requests are compared with each other to obtain the distance. Suppose test request is as depicted in Table 18.

After calculating the distance of these four values from four values of clusters' center of the previous step, it is identified that test request becomes the member of which cluster. If it is a member of cluster 3, the results are depicted in Table 19.

To fulfill this process, the following steps should be followed, respectively.

(a) Initializing

Table 18 A simple test request

Request's ID	SLO ₁	SLO ₂	SLO ₃	SLO ₄
5001	200	120	40	30

Table 19 The cluster's members that the test request belongs to

Request's ID	SLO ₁	SLO ₂	SLO ₃	SLO ₄
155	500	400	100	60
66	400	420	60	70
56	410	400	60	60

Table 20 The wolves' structure for the cluster with three requests

Wolves	Request ₁	Request ₂	Request ₃
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0

Table 21 The average Euclidian distance of test workload request of each wolf

Wolves	Fitness function
1	351
2	363
3	714
4	415

Constant coefficients such as vectors A and D , random values r , numbers of iterations of the algorithm, and the linear value of a should be initialized.

(b) Generating initial population

In the presented example, it is necessary to find the best case for the cluster identified for the test workload. Therefore, four cases are selected for the requests using GWO algorithm in which each cluster is identified as a wolf. The wolf's location is identified by the array X of Eq. (9) filled by 0 s and 1 s such that if the produced value is 0, then the request with the index 0 is ignored; otherwise, it is selected. The solution of this problem needs finding three answers. As an example, if the requests 1 and 2 are selected, the results are depicted in Table 20.

(c) Calculating the value of wolves

By utilizing Eq. (10), the average Euclidian distance of test workload request of each wolf, as the fitness function, is calculated. Therefore, the SLA indices of the requests are compared to find their distance, as shown in Table 21.

(d) Calculation of the wolves α , β and δ

The results of the calculated fitness function are sorted, and the first three values are identified as the wolves α , β and δ . The results are depicted in Table 22.

(e) Replacing wolves and termination condition

By utilizing Eqs. (11), (12), (13), and (14) with $\vec{a}=2$, and $\vec{r}_1=0.2$, and $\vec{r}_2=0.1$, will result in $\vec{A}=0.8$ and $\vec{C}=0.4$. Therefore, the values of \vec{D}_α , \vec{D}_β , and \vec{D}_δ are

Table 22 Identifying the wolves α , β , and δ

Wolves	Fitness function
1	(α) 351
2	(β) 363
4	(δ) 415
3	714

calculated, and thereby, the new location of the wolves will be identified, as shown in Table 23.

According to the sorted values of all requests, the first three requests are selected as the best ones. This process is continued until satisfying the termination condition. Finally, the location of the wolf α is returned as the optimum point.

4 Performance evaluation

In this section, we validate the effectiveness of our solution through simulation under different workload traces. In the following, we will explain the simulation setup and performance metrics. Then, the simulation results will be discussed.

4.1 Experimental setup

We used the Cloudsim toolkit [24] as a Java-based library in NetBeans IDE for modeling cloud ecosystem entities such as virtual machines, data centers, cloudlets, service broker and resource scheduling, and provisioning strategies. The configuration three types of hosts utilized in the simulation process is shown in Table 24.

We configure a number of hosts according to Table 24, randomly. Besides, the simulation setting used for every experiment in this research is presented in Table 25. We generate 90 cloud workloads randomly in the form of cloudlets using CloudSim library. A cloudlet includes all information related to cloud workload such as memory size, cloud workload size, file size, output size, and cost per workload. The heterogeneous cloud workloads are described in [10, 16] to validate the proposed solution under the same simulation setting.

Table 23 Replacement of the wolves and identifying their location

Current location of the wolf	1	0	1
Wolf α	0	0	1
Wolf β	0	1	0
Wolf δ	1	0	0
The new location of the wolf	0	1	1

Table 24 Host specifications [16]

Host Type	Processor configuration	Memory (GB)	Operating system	Number of VMs	Number of ECs	Prices (\$/EC unit)
Host 1	Intel Core2-Duo-2.4 GHz	1 GB RAM 160 GB HDD	Windows	6	18	2
Host 2	Intel Corei5-2310-2.9 GHz	1 GB RAM 160 GB HDD	Cloud Linux	4	12	3
Host 3	Intel XEONE-52407-2.2 GHz	2 GB RAM 320 GB HDD	Cloud Linux	2	6	4

Table 25 Simulation setting [16]

Parameter	Value
Number of resources	50–250
Number of workloads (cloudlets)	90
Bandwidth	1000–3000 B/S
Size of cloud workload	10,000 + (10–30%) MB
Number of PEs per machines	1
MIPS rating	100–4000 MIPS
Cost per cloud workload	3\$–5\$
Memory size	2048–12,576 MB
File size	300 + (15–40%) MB
Cloud workload size	300 + (15–50%) MB

4.2 Performance metrics

We use the following metrics for comparing the proposed approach with other studies; energy consumption, execution cost, execution time, latency, and SLA violation rate. Each metric is described as the following.

Energy consumption The energy consumption has a linear relationship with resource utilization which can be expressed by Eq. (16):

$$E = E_{Tr} + E_{DC} + E_{Mem} \tag{16}$$

E_{Tr} , E_{DC} , and E_{Mem} represent the energy consumption of switching equipment, datacenter, and storage device, respectively.

Execution cost It is the cost of workload execution which is measured in cloud dollars (C\$) as shown by Eq. (17)

$$Exe_{Cost} = \text{Min}(Cost(W_i, R_k)) \quad \text{for } 1 \leq i \leq m \text{ and } 1 \leq k \leq n \tag{17}$$

here $Cost(W_i, R_k)$ is the cost of executing workload i on resource k .

Execution time Execution time is the finishing time of the last workload.

Latency Latency is a time delay between expected execution ((Exp_i) time and actual execution time (Act_i) which is defined by Eq. (18)

$$\text{Latency} = \sum_i^n (Exp_i - Act_i) \tag{18}$$

where n is the number of workloads.

SLA Violation Rate SLA violations rate ($SLAVR$) is defined by Eq. (19), where FR is the failure rate and w_i is the weight of each $SLA = (SLA_1, \dots, SLA_z)$.

$$SLAVR = FR * \sum_{i=1}^z (w_i) \tag{19}$$

Failure rate (FR) is calculated by Eq. (20).

$$FR = \sum_{i=1}^z \left(\frac{\text{Failure}(SLA_i)}{z} \right) \quad (20)$$

When SLA_i is violated the $\text{Failure}(SLA_i) = 0$ and when SLA_i is not violated the $\text{Failure}(SLA_i) = 1$.

4.3 Results and discussion

To evaluate the performance of the proposed approach, we design various scenarios to examine performance metrics presented in the previous section. The proposed approach is compared with BULLET [16] and SCOOTER [10] methods. The reasons for choosing these two methods are as follows. (1) BULLET and SCOOTER outperform their counterparts. (2) Both methods and the proposed method operate on the multilayer cloud. (3) The same workload and parameter setting are used for the proposed, BULLET, and SCOOTER. BULLET applies a particle swarm optimization-based solution to schedule both heterogeneous and homogenous workloads on the cloud resources. The main aims of their proposed solution are: extracting QoS parameters of workloads, clustering workloads using patterns, and k-means-based clustering technique, and resource provisioning classified workloads according to their QoS parameters before resource scheduling [16]. SCOOTER utilized autonomic computing paradigm for a self-managing resource to execute cloud-based applications for satisfying QoS requirements. Their solution used workload analyzer component for clustering of heterogeneous cloud workloads using k-means technique and provisioned the required resources according to their QoS requirements [10].

In the following, the proposed approach (GFA) is compared with BULLET and SCOOTER in terms of energy consumption, execution cost, execution time, latency, and SLA violation rate.

4.3.1 Energy consumption

In this section, the impact of changing the number of workloads on energy consumption and the impact of changing the number of resources on energy consumption are investigated. Energy Consumption for GFA, BULLET, and SCOOTER against the number of workloads are shown in Fig. 5. It can be seen that the energy consumption is increasing by increasing the number of workloads because of utilizing the GWO technique to determine scaling decisions for efficient resource provisioning.

Energy consumption of GFA is less than its counterparts for all workloads ranging from 15 to 90. The average energy consumption for GFA, BULLET, and SCOOTER are 73.2 kWh, 80.8 kWh, and 78.3 kWh, respectively. GFA performs better than BULLET and SCOOTER in terms of energy consumption by 9.4% and 6.5%.

Energy Consumption of GFA, BULLET and SCOOTER against the number of resources are shown in Fig. 6. Here the number of workloads is considered to be 90.



Fig. 5 Energy consumption by changing the number of workloads

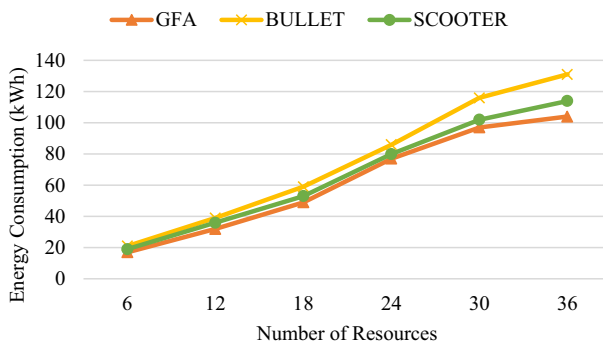


Fig. 6 Energy consumption by changing the number of resources

It can be seen that the energy consumption is growing by increasing the number of resources.

Energy consumption of GFA is less than its counterparts for resources ranging from 6 to 36. The average energy consumption for GFA, BULLET and SCOOTER are 62.6 kWh, 75.3 kWh, and 67.3 kWh, respectively. GFA performs better than BULLET and SCOOTER in terms of energy consumption by 16.8% and 6.9%.

4.3.2 Execution cost

In this section, the impact of changing the number of workloads on execution cost is examined. Then the impact of changing the number of resources on execution cost is investigated. Execution cost for GFA, BULLET, and SCOOTER against the number of workloads are shown in Fig. 7. It can be seen that the execution cost increases by increasing the number of workloads because of applying the hybrid solution using the GA algorithm and fuzzy C-means technique for clustering the heterogeneous cloud workloads based on QoS metrics.

Execution cost of GFA is less than its counterparts for all workloads ranging from 15 to 90. The average execution cost for GFA, BULLET, and SCOOTER are 252.5

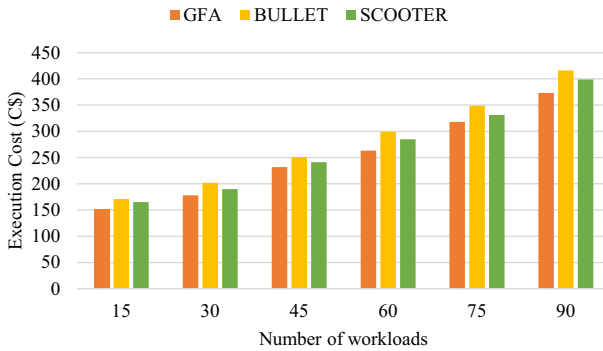


Fig. 7 Execution cost by changing the number of workloads

C\$, 281.3 C\$, and 268.5 C\$, respectively. GFA performs better than BULLET and SCOOTER in terms of execution cost by 10.2% and 6.1%.

Execution cost of GFA, BULLET, and SCOOTER against the number of resources are shown in Fig. 8. Here the number of workloads is considered to be 90. It can be seen that the execution cost is increasing by increasing the number of resources.

Execution cost of GFA is less than its counterparts for resources ranging from 6 to 36. The average execution cost for GFA, BULLET, and SCOOTER are 83.3 C\$, 93.8 C\$, and 89.2 C\$, respectively. GFA performs better than BULLET and SCOOTER in terms of execution cost by 11.1% and 6.6%.

4.3.3 Execution time

In this section, the impact of changing the number of workloads on execution time is investigated. Then the impact of changing the number of resources on execution time is studied. Execution time for GFA, BULLET, and SCOOTER against the number of workloads are shown in Fig. 9. It can be seen that the execution time is

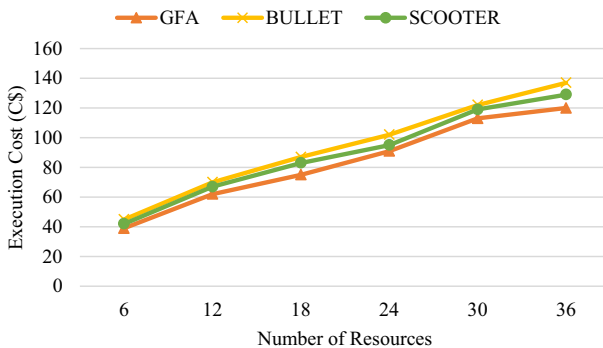


Fig. 8 Execution cost by changing the number of resources

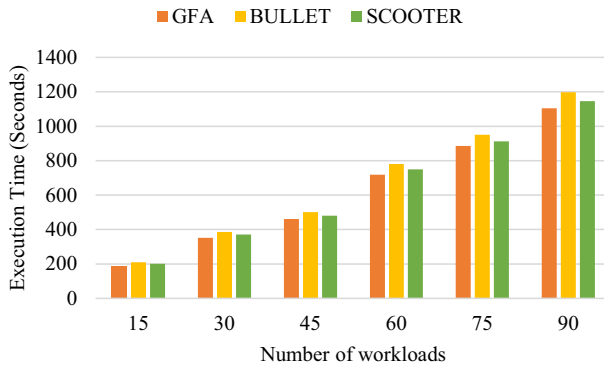


Fig. 9 Execution time by changing the number of workloads

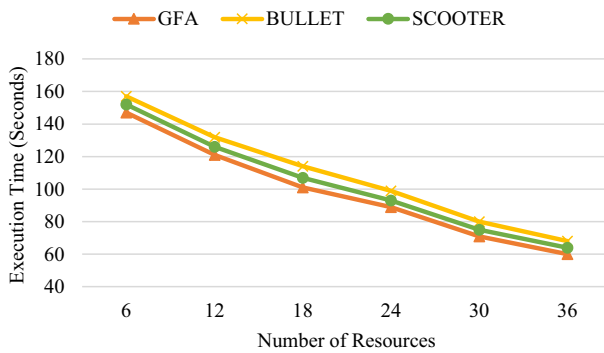


Fig. 10 Execution time by changing the number of workloads

rising by increasing the number of workloads. Applying the hybrid GA and fuzzy C-means technique for clustering workloads based on QoS metrics and then utilizing the GWO technique for scaling decisions lead to a better execution time performance. Another reason is that, the GFA algorithm converges to the global optimum faster than other approaches, as discussed in Sect. 4.3.6.

The execution time of GFA is less than its counterparts for all workloads ranging from 15 to 90. The average execution time for GFA, BULLET, and SCOOTER are 618 S, 670.8 S, and 643.3 S, respectively. GFA performs better than BULLET and SCOOTER in terms of execution time by 7.8% and 3.9%.

The execution time of GFA, BULLET, and SCOOTER against the number of resources are shown in Fig. 10. Here the number of workloads is considered to be 90. It can be seen that the execution time is decreasing by increasing the number of resources.

The execution time of GFA is less than its counterparts for resources ranging from 6 to 36. The average execution time for GFA, BULLET, and SCOOTER are

98.2 S, 108.3 S, and 102.8 S, respectively. GFA performs better than BULLET and SCOOTER in terms of execution time by 9.3% and 4.5%.

4.3.4 Latency

Latency for GFA, BULLET, and SCOOTER against the number of workloads are shown in Fig. 11. It can be seen that the latency is increased by increasing the number of workloads.

The latency of GFA is less than its counterparts for all workloads ranging from 15 to 90. The average latency for GFA, BULLET, and SCOOTER are 5.09 S, 5.65 S, and 5.35 S, respectively. GFA performs better than BULLET and SCOOTER in terms of execution time by 9.9% and 4.9%. By applying a proper workload clustering algorithm and then an enhanced resource provisioning decision, the proposed approach has a better performance in terms of latency, compared to its counterparts.

4.3.5 SLA violation rate

In this section, the impact of changing the number of workloads on the SLA violation rate is investigated. Then the impact of changing the number of resources on the SLA violation rate is examined. SLA violation rate varies between 0 and 100%. SLA violation rate for GFA, BULLET, and SCOOTER against the number of workloads are shown in Fig. 12. It can be seen that the SLA violation rate is increased by increasing the number of workloads. The GFA approach proposed a better execution time and latency and consequently a lower SLA violation rate.

The SLA violation rate of GFA is less than its counterparts for all workloads ranging from 15 to 90. The average SLA violation rate for GFA, BULLET, and SCOOTER are 7.3%, 7.7%, and 7.55%, respectively. GFA performs better than BULLET and SCOOTER in terms of execution time by 5.2% and 3.3%.

SLA violation rate of GFA is less than its counterparts for resources ranging from 6 to 36, as shown in Fig. 13. The average SLA violation rate for GFA, BULLET, and SCOOTER are 34.8%, 37.8%, and 36.5%, respectively. GFA performs better than BULLET and SCOOTER in terms of SLA violation rate by 7.9% and 4.6%.

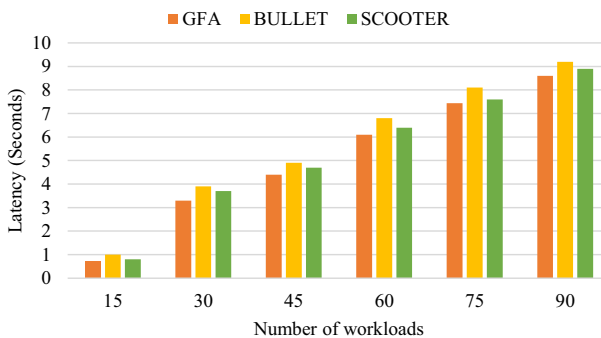


Fig. 11 Latency by changing the number of workloads

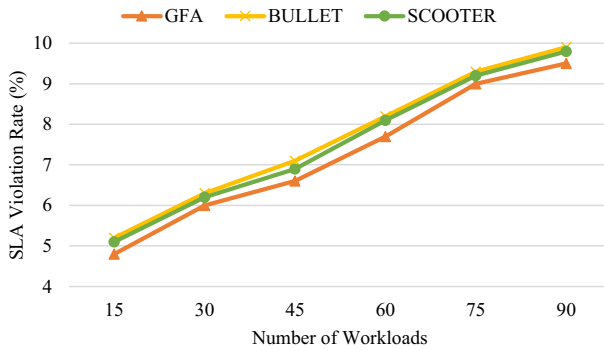


Fig. 12 SLA violation rate by changing the number of workloads

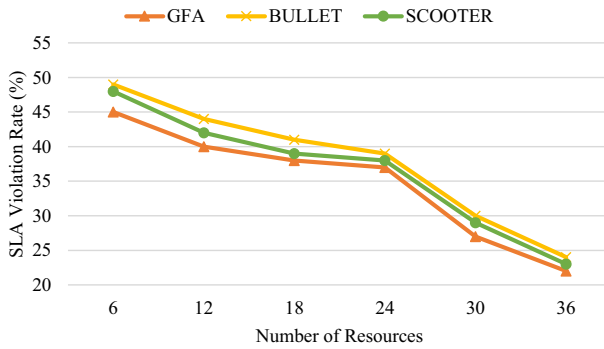


Fig. 13 SLA violation rate by changing the number of resources

Table 26 Parameters of each algorithm

GAF		ICA		PSO		GA	
Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
Population	90	N_{Pop}	90	$C_1 = C_2$	2	Population	90
Mutation rate	0.001	N_{Imp}	3	ω_{min}	0.5	Mutation rate	0.001
Crossover	0.8	β	5	ω_{max}	1	Crossover	0.8
		μ	0.05				

4.3.6 Convergence speed results

In this section, the convergence speed of GAF, ICA, PSO, and GA algorithms over iterations is presented. Parameter settings of all utilized algorithms are shown in Table 26. Besides, the value range of each request, including response time, cost, availability, and reliability are shown in Table 27.

Table 27 Values of four SLOs

SLO	Value
Response time	Between 20 and 1500
Cost	Between 2 and 15
Availability	Between 0.95 and 1
Reliability	Between 0.4 and 1

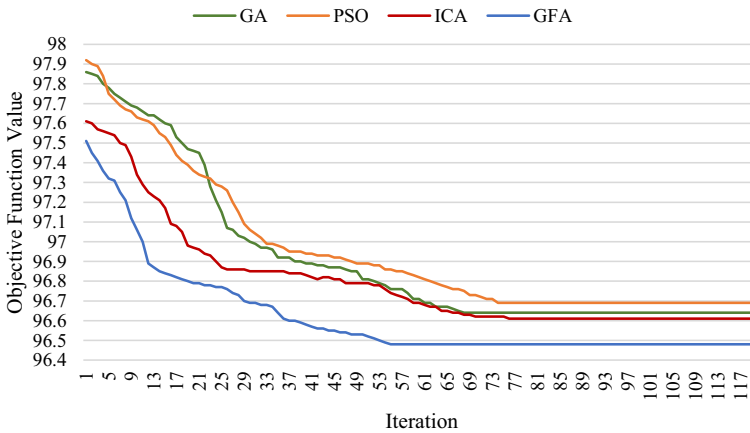


Fig. 14 Convergence characteristic of the proposed method, ICA, PSO, and GA for the best solutions

Figure 14 demonstrates the convergence characteristics of the GAF, ICA, PSO, and GA for the best solutions on 90 workloads where the y-axis represents iterations and the x-axis represents the objective function value defined by Eq. (5).

According to the simulation results, the GFA algorithm converges to the global optimum after 55 iterations while ICA, PSO, and GA algorithms converge to the global optimum after 71, 74, and 68 iterations, respectively. Besides, the GAF algorithm shows better results on the best solution. The best global solutions for GAF, ICA, PSO, and GA are 96.48, 96.61, 96.78, and 96.64. It can be concluded that the proposed algorithm (GFA) converges faster than other methods.

5 Conclusion

Recently, the usage of cloud computing as a promising computing model for delivering and hosting the applications over the Internet has increased significantly in recent years. In this paper, we studied the resource provisioning issue using workload clustering for cloud-based applications. Due to the heterogeneity of workloads submitted, the analysis and identification common workload patterns based on the user’s QoS requirements can play an important role to provision of the cloud resources in a cloud environment. We utilized the genetic algorithm with fuzzy C-means technique for clustering the heterogeneous cloud workloads based on QoS

metrics. Moreover, GWO as a metaheuristic technique to identify the appropriate scaling decisions to provide an efficient resource provisioning solution for satisfying user's QoS requirements has used. We validate the proposed approach under real-world workloads, and the obtained simulation results indicated that it outperforms in terms of the CPU utilization, elasticity, response time, and total cost compared with the other mechanisms. For future work, we plan for validation the scalability of proposed approach on the real benchmark application such as Amazon web service (AWS) cloud benchmark and its extension using container-based virtualization technology. Besides, we will utilize the pattern mining models to extract user behavioral patterns to access the cloud-based applications and also use the deep reinforcement learning algorithm to determine the resource scaling decisions for serving of cloud workloads.

References

1. Buyya R, Vecchiola C, Selvi ST (2013) Mastering cloud computing: foundations and applications programming. In: Newnes
2. Chandrasekaran K (2014) Essentials of cloud computing. CRC Press, Boca Raton
3. Ghobaei-Arani M, Khorsand R, Ramezanpour M (2019) An autonomous resource provisioning framework for massively multiplayer online games in cloud environment. *J Netw Comput Appl* 142:76–97
4. Manvi SS, Shyam GK (2014) Resource management for Infrastructure as a Service (IaaS) in cloud computing: a survey. *J Netw Comput Appl* 41:424–440
5. Shahidinejad A, Ghobaei-Arani M, Esmaili L (2019) An elastic controller using Colored Petri Nets in cloud computing environment. *Cluster Comput* 1–27. <https://doi.org/10.1007/s10586-019-02972-8>
6. Iqbal W, Erradi A, Mahmood A (2018) Dynamic workload patterns prediction for proactive auto-scaling of web applications. *J Netw Comput Appl* 124:94–107
7. Singh S, Chana I (2015) Q-aware: Quality of service based cloud resource provisioning. *Comput Electr Eng* 47:138–160
8. Wang X, Wang H (2020) Driving behavior clustering for hazardous material transportation based on genetic fuzzy C-means algorithm. *IEEE Access* 8:11289–11296
9. Mirjalili S, Saremi S, Mirjalili SM, Coelho LDS (2016) Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization. *Expert Syst Appl* 47:106–119
10. Gill SS, Buyya R (2019) Resource provisioning based scheduling framework for execution of heterogeneous and clustered workloads in clouds: from fundamental to autonomic offering. *J Grid Comput* 17(3):385–417
11. Erradi A, Iqbal W, Mahmood A, Bouguettaya A (2019) Web application resource requirements estimation based on the workload latent features. *IEEE Trans Services Comput*. <https://doi.org/10.1109/TSC.2019.2918776>
12. Xu L, Wang H, Lin W, Gulliver TA, Le KN (2019) GWO-BP neural network based OP performance prediction for mobile multiuser communication networks. *IEEE Access* 7:152690–152700
13. Xu L, Wang J, Wang H, Gulliver TA, Le KN (2019) BP neural network-based ABEP performance prediction for mobile Internet of Things communication systems. *Neural Comput Appl* 1–17. <https://doi.org/10.1007/s00521-019-04604-z>
14. Xu Y-H, Xie J-W, Zhang Y-G, Hua M, Zhou W (2020) Reinforcement Learning (RL)-based energy efficient resource allocation for energy harvesting-powered wireless body area network. *Sensors* 20(1):44
15. Xu YH, Liu ML, Xie JW, Zhou J (2019) An IEEE 802.21 MIS-based mobility management for D2D communications over heterogeneous networks (HetNets). *Concurr Comput Pract Exp* 32:5. <https://doi.org/10.1002/cpe.5552>

16. Gill SS, Buyya R, Chana I, Singh M, Abraham A (2018) BULLET: particle swarm optimization based scheduling technique for provisioned cloud resources. *J Netw Syst Manag* 26(2):361–400
17. Mian R, Martin P, Vazquez-Poletti JL (2013) Provisioning data analytic workloads in a cloud. *Fut Gener Comput Syst* 29(6):1452–1458
18. Magalhães D, Calheiros RN, Buyya R, Gomes DG (2015) Workload modeling for resource usage analysis and simulation in cloud computing. *Comput Electr Eng* 47:69–81
19. Amiri M, Mohammad-Khanli L, Mirandola R (2018) An online learning model based on episode mining for workload prediction in cloud. *Fut Gener Comput Syst* 87:83–101
20. Meenakshi A, Sirmathi H, Ruth JA (2019) Cloud computing-based resource provisioning using k-means clustering and GWO prioritization. *Soft Comput* 23(21):10781–10791
21. Raza B et al (2018) Autonomic workload performance tuning in large-scale data repositories. *Knowl Inf Syst* 1–37. <https://doi.org/10.1007/s10115-018-1272-0>
22. Liu C, Liu C, Shang Y, Chen S, Cheng B, Chen J (2017) An adaptive prediction approach based on workload pattern discrimination in the cloud. *J Netw Comput Appl* 80:35–44
23. Singh P, Gupta P, Jyoti K (2018) TASM: technocrat ARIMA and SVR model for workload prediction of web applications in cloud. *Clust Comput* 22(2):619–633
24. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exp* 41(1):23–50

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.