Cluster Computing
https://doi.org/10.1007/s10586-020-03107-0

Resource provisioning using workload clustering in cloud computing environment: a hybrid approach

Ali Shahidinejad[1] · Mostafa Ghobaei-Arani[1] · Mohammad Masdari[2]

Received: 28 June 2019 / Revised: 3 February 2020 / Accepted: 6 April 2020

Abstract
In recent years, cloud computing paradigm has emerged as an internet-based technology to realize the utility model of computing for serving compute-intensive applications. In the cloud computing paradigm, the IT and business resources, such as servers, storage, network, and applications, can be dynamically provisioned to cloud workloads submitted by end-users. Since the cloud workloads submitted to cloud providers are heterogeneous in terms of quality attributes, management and analysis of cloud workloads to satisfy Quality of Service (QoS) requirements can play an important role in cloud resource management. Therefore, it is necessary for the provisioning of proper resources to cloud workloads using clustering of them according to QoS metrics. In this paper, we present a hybrid solution to handle the resource provisioning issue using workload analysis in a cloud environment. Our solution utilized the Imperialist Competition Algorithm (ICA) and K-means for clustering the workload submitted by end-users. Also, we use a decision tree algorithm to determine scaling decisions for efficient resource provisioning. The effectiveness of the proposed approach under two real workloads traces is evaluated. The simulation results demonstrate that the proposed solution reduces the total cost by up to 6.2%, and the response time by up to 6.4%, and increases the CPU utilization by up to 13.7%, and the elasticity by up to 30.8% compared with the other approaches.

Keywords Cloud computing · Workload clustering · Resource provisioning · Imperialist competition algorithm · Decision tree algorithm

1 Introduction

Cloud computing is an Internet-based computing paradigm that delivers Information Technology (IT) resources to end-users and businesses. Cloud computing enables on-demand access to a shared pool of IT resources, such as computing resources, development platforms, and applications as services to the end-users according to pay-as-per-use model. Cloud computing is very economical and saves a lot of money. A blind benefit of this computing is that even if we lose our laptop or due to some crisis our personal computer gets damaged, still our data and application will stay safe and secured as these are not in our local computer [1].

One of the main motivations for cloud providers is leveraging their cloud resources to run cloud applications for making a profit. The end users submit their requests derived by cloud applications with different Quality of Service (QoS) requirements in the form of Service Level Agreements (SLAs) to run by cloud providers. Hence, cloud providers want to obtain as many new requests as possible with the aim of maximizing profit; on the other hand, they must satisfy QoS requirements according to the agreed SLA with end-users. To achieve this goal, we need efficient resource provisioning mechanisms. Resource provisioning is the allocation of cloud resources to serve user requests in a way to reduce the response cost and time while considering SLAs such as availability, reliability, response time limit and cost limit [2].

Determining the right amount of cloud resources for executing the user requests depends on the incoming

✉ Ali Shahidinejad
a.shahidinejad@qom-iau.ac.ir

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

[2] Department of Computer Engineering, Urmia Branch, Islamic Azad University, Urmia, Iran

Published online: 23 April 2020

workloads. Resource provisioning can be performed using a long-term reservation or on-demand. Utilizing long-term reservation, there will be many idle cloud resources, and in the on-demand plan, too many requests may be executed on a cloud resource. Besides, users usually have irregular access to cloud resources, and there will be fluctuations in the workloads. Workload fluctuation issue leads to under-provisioning and over-provisioning problems, which leads to wastage of resources and time. In over-provisioning problem, too many workloads are executed on a single resource while in under-provisioning, there are many idle resources [3]. One solution is to predict the workload using patterns of the previous usage and the current state of the cloud resources. The patterns link user requests with cloud resources according to the request type. In [4], they applied log smoothing and a fuzzy logic-based method for workload prediction. Smoothing detects the workload trend by using log data and the fuzzy logic addresses uncertainty of the monitored workload. Another solution is to analyze and classify the workloads according to their SLA. Then the output of workload clustering is used to determine scaling decisions for resource provisioning [5]. One of the most common clustering methods for workload clustering in cloud computing is K-means [6]. The output of the K-means algorithm is depended on selecting initial cluster centers. To solve this problem, different clustering algorithms using evolutionary algorithms have been introduced. However, evolutionary approaches like Genetic Algorithm are very slow. Recent researches have investigated new evolutionary methods like Particle Swarm Optimization (PSO) [7] and Imperialist Competition Algorithm (ICA) [8], to have a better response and quicker convergence.

In this study, we proposed a hybrid ICA/K-means technique for workload clustering, and then resource provisioning is performed using a decision tree algorithm. The proposed approach includes workload preprocessing, workload clustering, and resource provisioning. In workload preprocessing, noisy and unnecessary requests are eliminated. Then the training workload is clustered by Imperialist Competitive algorithm and K-means clustering technique. Afterwards, the test workload is assigned to the nearest cluster center. Finally, resource provisioning is performed using a decision tree algorithm. In the hybrid ICA/K-means algorithm, after generating the initial population in ICA, K-means technique is used to improve the position of each colony. This combination results in a better response and also quicker convergence than other evolutionary algorithms.

The main contributions of this research include:

- Designing a framework inspired by the three cloud layer architecture for workload clustering-based resource provisioning.
- Proposing a hybrid workload clustering solution using Imperialist competition algorithm and K-means clustering technique.
- Considering the preprocessing of heterogeneous cloud workloads by eliminating noisy and unnecessary requests.
- Improving the convergence speed of the ICA clustering technique than other evolutionary algorithms to find for local and global optimum points.

The rest of this paper is organized as follows: In Sect. 2, we focus on a survey of related works. Section 3 describes the proposed solution in more detail. The experimental design and discuss the experimental results are provided in Sect. 4, and we finally present the conclusions and future works in Sect. 5.

2 Related work

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

Sukhpal Singh et al. [9] have studied recent researches in resource management mechanisms including resource scheduling, resource provisioning, and autonomous resource scheduling and provisioning approaches. They reviewed and categorized resource management mechanisms based on QoS metrics such as response time, resource utilization, cost, SLA violations and so on. Besides, they provided future research challenges about autonomic resource management mechanisms in the cloud environment.

Kaur et al. [10] have proposed an extended shuffled frog leaping method for resource provisioning of workflow-based applications in the cloud environment. The aim of their proposed solution is minimizing execution cost applications while satisfying the agreed deadline. Also, they validate their solution compared with the particle swarm optimization algorithm and shuffled frog leaping algorithm under well-known workflow-based applications. Singh et al. [6] presented a resource provisioning solution based on various Quality of Service (QoS) requirements of heterogeneous workloads. Their proposed solution provisioned cloud resources for serving heterogeneous workloads by clustering and analyzing workloads and classifying them using common patterns. Besides, the workload analysis has been done using K-Means based clustering technique using assigning weights to quality characteristic in each heterogeneous cloud workload through QoS requirements. Their simulation results indicated that the proposed solution is efficient in reducing execution time and execution cost of heterogeneous cloud

workloads while satisfying customers QoS requirements. Haghighi et al. [11] have presented a hybrid solution using k-means clustering method and the micro-genetic algorithm to reduce power consumption in the cloud datacenters. Their solution utilized the k-means clustering to map tasks to the cloud servers and dynamic consolidation of cloud servers to attain an energy-QoS solution. Besides, their solution is able to make a suitable trade-off between the number of SLA violations and the power consumption of cloud servers. Finally, they evaluated their solution using Cloudsim toolkit and indicated that it reduce the number of VM migrations and the make-span compared with particle swarm optimization and genetic algorithms.

Zhang et al. [4] proposed a prediction method that combines a workload pattern mining technique with a collaborative filtering solution to satisfy the accuracy and efficiency requirements. They utilized a fuzzy logic-based algorithm to capture uncertain information to make the prediction technique more robust against noise and uncertainty. Furthermore, they implemented their proposed prediction mechanism on the OpenStack cloud platform and illustrated that it outperforms in terms of the accuracy, efficiency, and robustness compared with other techniques. Singh et al. [12] have designed an SLA-aware autonomic resource provisioning method to reduce the SLA violation rate and satisfying the QoS requirements for serving heterogeneous workloads in cloud environments. Their method considers various QoS requirements such as cost, reliability, latency, availability, and execution time to calculate the SLA violation rate. Their simulation results indicated that their proposed method outperforms in terms of SLA violation rate and satisfying QoS requirements for delivering cloud services. In [13], a light-weight framework for predicting and resource provisioning of cloud services with sudden bursts of workload traffic have designed. Their framework is consist of three units namely: workload classification by characterization, traffic prediction, and adaptive dynamic resource provisioning. Besides, they validate their proposed framework under Google cluster and online game World of Warcraft workload traces and demonstrated that it handles efficiently workload fluctuation and it outperforms in terms of resource utilization, energy consumption, and guarantee ratio compared with other existing solutions. In [14], a particle swarm optimization-based solution to schedule of both heterogeneous and homogenous and 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.

Suresh et al. [15] have proposed an extended resource provisioning approach using particle swarm optimization technique by selecting the optimal resource with minimum cost. Their algorithm classified the cloud resources using kernel fuzzy c-means technique into several clusters. They evaluated their proposed approach using Cloudsim toolkit and indicated that it reduces execution time, memory usage, and cost. Cheng et al. [16] have designed a deep learning-based approach with two-step for resource provisioning to reduce energy cost in cloud data centers. Their proposed approach is comprised of two steps: decomposition user workload model into several tasks and minimizing energy cost using the deep learning-based technique. Their proposed solution dynamically make the scaling decisions using learning from the user request patterns and realistic price model. Gong et al. [17] have developed an adaptive resource provisioning algorithm using control theory technique for reacting to the workload fluctuations. Their algorithm utilized an adaptive multi-input and multi-output control and radial basis function neural network for improving resource utilization and meeting QoS requirements. In their proposed algorithm, the CPU and memory allocated to cloud applications according to the workload changes and QoS requirements. Singh Gill et al. [18] have designed an extended framework for self-adaptive of cloud resource for serving clustered workloads. Their framework controls the cloud resources automatically using self-management characteristics including self-healing to find sudden faults, self-optimizing to minimize execution time, execution cost, the SLA violation rate, self-configuring to readjust cloud resources, and self-protecting to detect and protect of attacks.

Vozmediano et al. [19] have proposed a hybrid auto-scaling solution using machine learning technique and queuing theory for adaptive provisioning of cloud services. Their solution utilized the support vector machine regression method to predict the workload of web server according to the historical observations. Besides, their solution used a queueing model to specify the number of cloud resources that must be allocated based on the estimated workload. Their simulation results demonstrated that the SVM-based regression method to achieve higher prediction accuracy than other classical prediction models. Feng et al. [20] have designed an elastic resource provisioning framework using cloud layer model by a suitable threshold in a flexible way. They utilized the Grey relational analysis strategy to adjust the proper threshold based on CPU and memory usage. Besides, they implemented a fine-grained scaling mechanism to scale up the cloud servers in the physical machine level or virtual

machine level to satisfy QoS requirements. Finally, they proposed used the weighted moving average prediction model to scale down the cloud servers for reducing energy consumption. Erradi et al. [21] have proposed a novel approach to estimate the resource utilization and the response time for different workload patterns of web applications using latent workload features in the cloud datacenters. Their approach utilized unsupervised learning techniques on the historical access logs for identifying latent features to predict the resource demands. They validate the proposed approach under various benchmark applications and indicated that their proposed approach outperforms in terms of estimating CPU, memory, bandwidth usage, and response time compared with other approaches.

Generally, most of the current works used heuristic-based mechanism to determine scaling decisions and resource provisioning for serving cloud workloads. Since usually, workloads submitted by users to cloud providers are heterogeneous with different QoS requirements, most of the heuristic-based mechanisms are still not entirely suitable for provisioning of cloud resources to handle heterogeneous cloud workloads. Therefore, we combine the ICA technique as a metaheuristic-based mechanism with a decision tree algorithm to handle the resource provisioning problem using workload analysis in a cloud environment. Although some approaches [14, 15] have already been used the meta heuristic-based mechanisms for addressing heterogeneous cloud workloads, still more work is required for provisioning cloud infrastructure on-demand in an efficient manner.

Table 1 sums up some of the most relevant works related to resource provisioning techniques using workload clustering based on six characteristics: (1) utilized technique, (2) performance metrics, (3) policy, (4) scaling type (5) evaluation tool and (6) workload type.

3 Proposed approach

The framework of the proposed approach is illustrated in Fig. 1. When user requests are submitted to the cloud provider, they are forwarded to the Workload Recorder. Workload Information database is used to save the recorded workloads. Workload Preprocessing component captures the requests from Workload Information database and eliminates noisy and unnecessary requests. An ID is defined for each request, and the SLA table is formed. After that, a hybrid imperialist competitive/K-means based clustering algorithm is used to clustering the workloads for execution on a different set of resources. Finally, by using a decision tree algorithm, the desired resource provisioning is performed and delivered to the infrastructure layer.

In the following, we introduce a problem formulation for the proposed model, and we then present the proposed algorithm for workload clustering and resource provisioning in the cloud environment.

3.1 System model

In this section, we explain the system model and notations used in the proposed approach called Dynamic Resource Provisioning _Cluster ICA (DRP_CICA). Table 2 introduces the notations used in the structure of the DRP_CICA. Requests in a workload are defined as Request $= \{\mathrm{Req}_1.\mathrm{Req}_2.\ldots..\mathrm{Req}_N\}$ where N represents the number of requests in the workload. Let *Train_req* be requests for training data and let *Test_req* be requests for testing data. Each request has its special SLA. Each SLA structure contains four characteristics as conditions of each request: *Res* as the response time, *Cost* as cost, *AV* as availability, and *Rel* as reliability. *SLAV* represents SLA violation which is the difference between the real response time of a request ($Response_i$) and the user desired response time ($Deadline_i$).

Virtual machines are defined as VM $= \{\mathrm{VM}_1.\mathrm{VM}_2.\ldots..\mathrm{VM}_k\}$ where k is the number of virtual machines. Each virtual machine has different characteristics for providing the service. $VMCost_i$ is the cost of using the i^{th} VM, $VMMIPS_i$ is the amount of MIPS of the i^{th} virtual machine, $VMRAM_i$ is the amount of i^{th} virtual machine memory, and $VMStorage_i$ is the amount of storage space of the i^{th} virtual machine. If a virtual machine needs a new configuration, the cost of a new configuration VM_Init_i is also calculated. As shown in Table 2, *Cost*, and *Response* represent the sum of costs for all requests and sum of responding time.

Note that VM configuration can be either replication or resizing. In replication strategy (i.e., horizontal scaling), the resource unit is the server replica (i.e., running on a VM), and new replicas are added or released (i.e., scaling in/out) as needed. In contrast, resizing strategy (i.e., vertical scaling) consists of changing the resources assigned to an already running VM, for example, increasing or reducing the allocated CPU power or the memory (i.e., scaling up/down). Most common operating systems do not allow on-the-fly (i.e., without rebooting) changes on the machine on which it runs (even if it is a VM); for this reason, most cloud providers only offer replication strategy. In this study, we also utilized the replication strategy to configure cloud infrastructure.

Center (Center $= \{\mathrm{Center}_1.\mathrm{Center}_2.\ldots..\mathrm{Center}_\alpha\}$) is known as the center of the clusters where α is the number of formed cluster. *Dis* (i,j) represents Euclidean distance between i and j requests. Specifically, for each time that a violation of the term of service occurs, a penalty is

Table 1 Survey of works related to resource provisioning techniques using workload analysis

References	Utilized technique	Performance metric	Policy	Scaling type	Benchmark	Workload
Singh [6]	Heuristic-based+K-means clustering	Submission burst time, total execution time, network usage, serviceability	Proactive	Replication	CloudSim toolkit	Real workloads
Zhang [4]	Fuzzy logic+Pattern matching	Execution time, MAE	Proactive	Replication/ resizing	OpenStack+Azure	Web-based workloads
Singh [12]	Autonomic-based+K-means clustering	SLA violation, reliability, availability, latency, execution time, execution cost	Reactive	Replication	CloudSim toolkit+JADE Platform	Real web-based workloads
Chen [13]	Heuristic-based	Guarantee ratio, Energy consumption, Resource utilization	Proactive	Replication	MATLAB	Google cluster, WoW workloads
Gill [14]	PSO+K-means clustering	Execution time, execution cost, energy consumption, resource utilization, reliability, availability, latency	Reactive	Replication	CloudSim toolkit	Real web-based workloads
Suresh [15]	PSO+Fuzzy c-means clustering	Cost, Memory utilization, Execution time	Reactive	Replication	CloudSim toolkit	Synthetic workloads
Cheng [16]	Deep reinforcement learning+	Energy cost, Runtime, SLA violation	Reactive	Replication	Python	Google cluster workloads
Gong [17]	Adaptive control theory+Neural network	Response time, CPU utilization, memory utilization, throughput	Proactive	Resizing	MATLAB	FIFA workload
Gill [18]	Autonomic-based+K-means clustering	Execution time, execution cost, fault detection rate, Waiting time, energy consumption, resource utilization, SLA violation, throughput	Reactive	Replication	CloudSim toolkit+JADE Platform	Flipkart and Snapdeal workloads
Vozmediano [19]	SVM+queuing theory	SLA violation. MAE, RMSE	Proactive	Replication	CloudSim toolkit	Web-based real workloads
Feng [20]	GRA+weighted moving average	Cost, energy consumption, resource utilization, SLA violation	Reactive/ proactive	Replication/ Resizing	CloudStack+TPC benchmark	NASA and Synthetic workloads
Erradi [21]	Unsupervised learning-based clustering	Response time, CPU utilization, memory utilization, bandwidth utilization, MSE	Proactive	Replication	Amazon EC2 + RUBiS, Acme Air benchmarks	Web-based workloads
Proposed	ICA/K-means clustering+Decision tree	Response time, cost, CPU utilization, elasticity	Reactive	Replication	CloudSim toolkit	FIFA and ClarckNet workloads

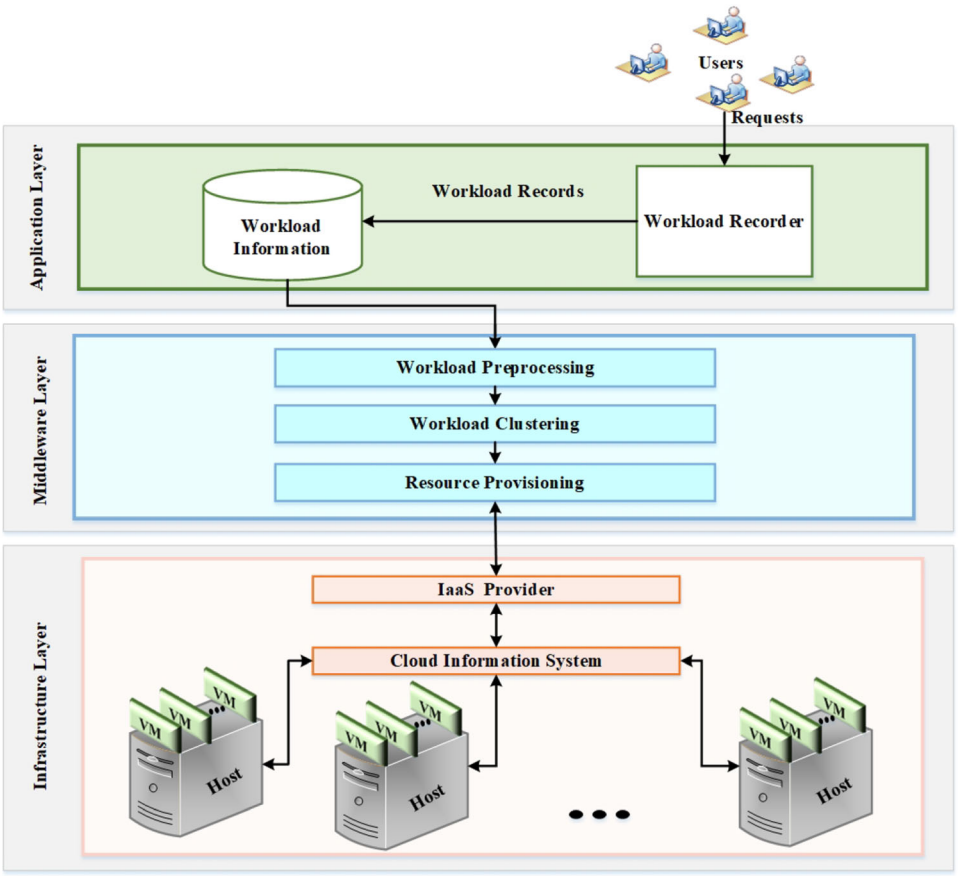
MAE mean absolute error, *SVM* support vector machine, *RMSE* root mean squared error, *SLA* service level agreement, *WoW* World of warcraft, *JADE* java agent development framework, *PSO* particle swarm optimization, *GRA* grey relational analysis, *RUBiS* rice university bidding system

assigned for each request and also, there is a coefficient for the progressive increase of the penalty rate ($\alpha penalty$).

3.2 The proposed algorithm

In this section, we explain our proposed approach for resource provisioning in more details. The flowchart of the proposed approach is illustrated in Fig. 2. The workflow of the proposed approach includes three major steps: preprocessing, workload clustering, and resource provisioning. The first step is preprocessing of workload that eliminates noisy and unnecessary requests. Then for each request, an ID is defined, and the SLA table is created using this ID. Finally, the SLA data is normalized. In the second step, the training workload is clustered by Imperialist Competitive algorithm and K-means. After that, by using the Euclidean distance, the nearest cluster center to the test

Fig. 1 Proposed framework



workload is selected. Finally, by using a decision tree algorithm, resource provisioning is performed.

3.2.1 Workload preprocessing

Workload preprocessing has a great impact on the performance of the proposed approach because the output of this phase is fed to the workload clustering. Workload preprocessing includes three steps. First, invalid requests are removed. Then an identifier is set for each request. Finally, the SLA table is created according to the requests.

3.2.1.1 Removing invalid requests The first step of workload preprocessing is requests refinement where unnecessary requests are removed. Unnecessary requests are caused by the following situations:

-One of the fields of the request is not recorded accurately (data contains NULL).
-The user is recognized as a scanning robot, spam, or intruder.

3.2.1.2 Setting identifier for each request In this step, a unique ID number is assigned to each request. This makes each request simply available for further calls.

3.2.1.3 Making SLA An SLA is a commitment between a user and a service provider. Particular features of the service, including cost, response time limit, availability, and reliability are agreed between the service user and the service provider, as shown in Table 3. A provisioning mechanism must have the capacity to estimate the desired resources to deal with workload fluctuations and satisfying these service level objective (SLO) requirements for avoiding over-provisioning or under-provisioning problems.

3.2.1.4 SLA normalization The parameters in the SLO requirement of each request do not have the same type. Therefore, to use these objectives in the next steps, they should be normalized. There are different normalization methods like decimal scaling min–max and Z-score. Here, a combination of Z-score normalization and Mean Absolute Deviation (MAD) is used. MAD is applied instead of standard deviation in the Z-score normalization method to control the side effect of outlier data [22]. MAD and Z-score normalization are presented in Eqs. (1) and (2), respectively.

Table 2 Variables and their definitions

Name	Definition
N	Number of requests
Req_i	i^{th} request
Train_req	Training workload requests
Test_req	Test workload requests
SLA	Serviec Level Agreement table
SLAV	SLA Violation
Response_i	Real response time of i^{th} request
Deadline_i	Desired response time of i^{th} request
$SLO_i.Cost$	Cost condition in i^{th} request
$SLO_i.Res$	Responsible time condition in i^{th} request
$SLO_i.Av$	Availability condition in i^{th} request
$SLO_i.Rel$	Reliability condition in i^{th} request
$NSLO$	Normalized Service Level Objective
C_i	i^{th} Cluster Center
Imp_i	i^{th} Imperialist
Penalty	Penalty of the SLA violation
αPenalty	Progressive increase coefficient of penalty
VM	Virtual machine
$VMCost_i$	Cost of using i^{th} VM
$VMRAM_i$	Amount of i^{th} VM memory
$VMMIPS_i$	Amount of MIPS of the i^{th} VM
$VMStorage_i$	Amount of i^{th} VM storing space
$VMPrice_i$	The price of i^{th} VM for a client request
$VMhour_i$	The time duration for responding to i^{th} requests by hour
VM_Init_i	The cost of the initial adjustment of the i^{th} VM
OF	Objective function
Cost	Sum of cost
Response	Sum of response time

$$MAD = \frac{\sum_{i=1}^{N} |X_i - \mu_x|}{N} \tag{1}$$

$$norm\_data = \frac{|X_i - \mu_x|}{MAD} \tag{2}$$

where μ_x is the average of data, N is the number of data, and X_i is the ith value of data. After the normalization, we call each SLO as NSLO.

In more details, the normalization is performed on all the arrived requests in a time interval and each request has four SLO (SLO_1, SLO_2, SLO_3, SLO_4,). First, the mean absolute deviation of an SLO is calculated using Eq. (1) where μ_x is the average of SLO_i of all requests arrived in a time interval and then each SLO is normalized based on Eq. (2).

3.2.2 Workload clustering

In the proposed approach, clustering is based on a combination of imperialist competitive and K-means algorithms. The main reason behind this combination is to improve the initial population of colonies. K-means forms the initial empires of the ICA. Applying the K-means algorithm prevents workload clustering falling into local optima because it converges more quickly. Flowchart of the proposed clustering structure is shown in Fig. 3.

The inputs of the algorithm are user requests and the number of clusters (K). To select the value of K, the elbow curve method is used. Clustering models are built as increasing the number of clusters. Then the distance between all cluster members and their cluster center is calculated for each new model. This will result in a series of clustering models with their distance regarding value. As the number of clusters increases, the distance value keeps decreasing until it settles in a point. In the elbow curve method, a huge drop point which means that increasing the number of clusters leads to an optimized model is considered as the K value of the k-means algorithm. The steps of the proposed clustering method are presented in the following.

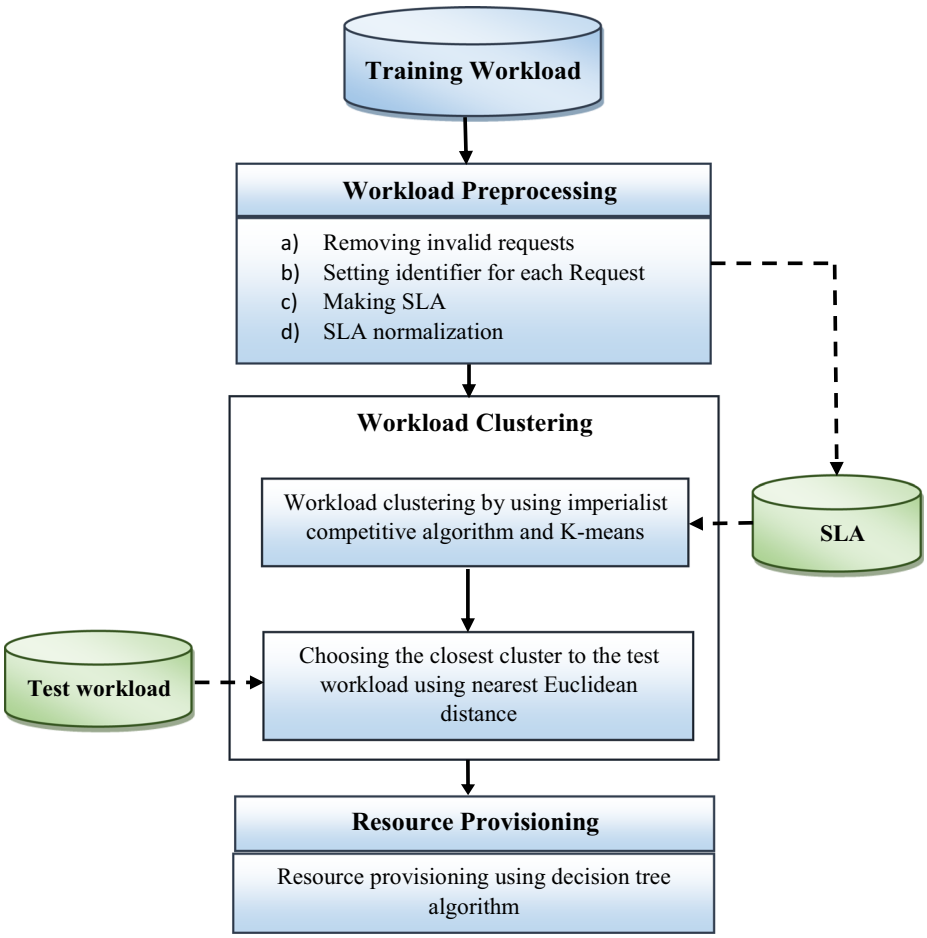


Fig. 2 Flowchart of the proposed approach

Table 3 SLA structure for each request

Request ID	User SLA			
	SLO_1	SLO_2	SLO_3	SLO_4
	Cost	Response time	Availability	Reliability

3.2.2.1 Formation of the initial population

The initial population of input requests is considered as Eq. (3).

$$\text{Population} = [X_1,\ X_2,\ ...,\ X_\alpha,\ ...\ X_N] \tag{3}$$

Here X_i represents Country_i which includes K clusters and each cluster contains d members. The development of this vector is performed randomly.

3.2.2.2 Calculating the objective function value and then sorting the population

Considering N number of input data and K number of clusters, the objective function (OF) is given by Eq. (4).

$$OF = \sum_{m=1}^{N} \min \left(\sqrt{\sum_{S=1}^{4} (Req_m.NSLO_S - Center_i.NSLO_S)^2}, i = 1toK \right), \tag{4}$$

where Req_m is the mth request, $NSLO_S$ is the sth feature of the NSLO table, and $Center_i$ is the ith cluster center. After calculating the OF, the initial population is sorted according to their value.

3.2.2.3 Choosing the imperialist states and distributing colonies among them

Countries with the minimum value of OF are considered as imperialists and others are selected as the colonies of these imperialists. The colonies are distributed among imperialists base on their power. The normalized power is calculated as Eq. (5).

$$Power_n = \left| C_n / \sum_{i=1}^{N_{imp}} C_i \right| \tag{5}$$

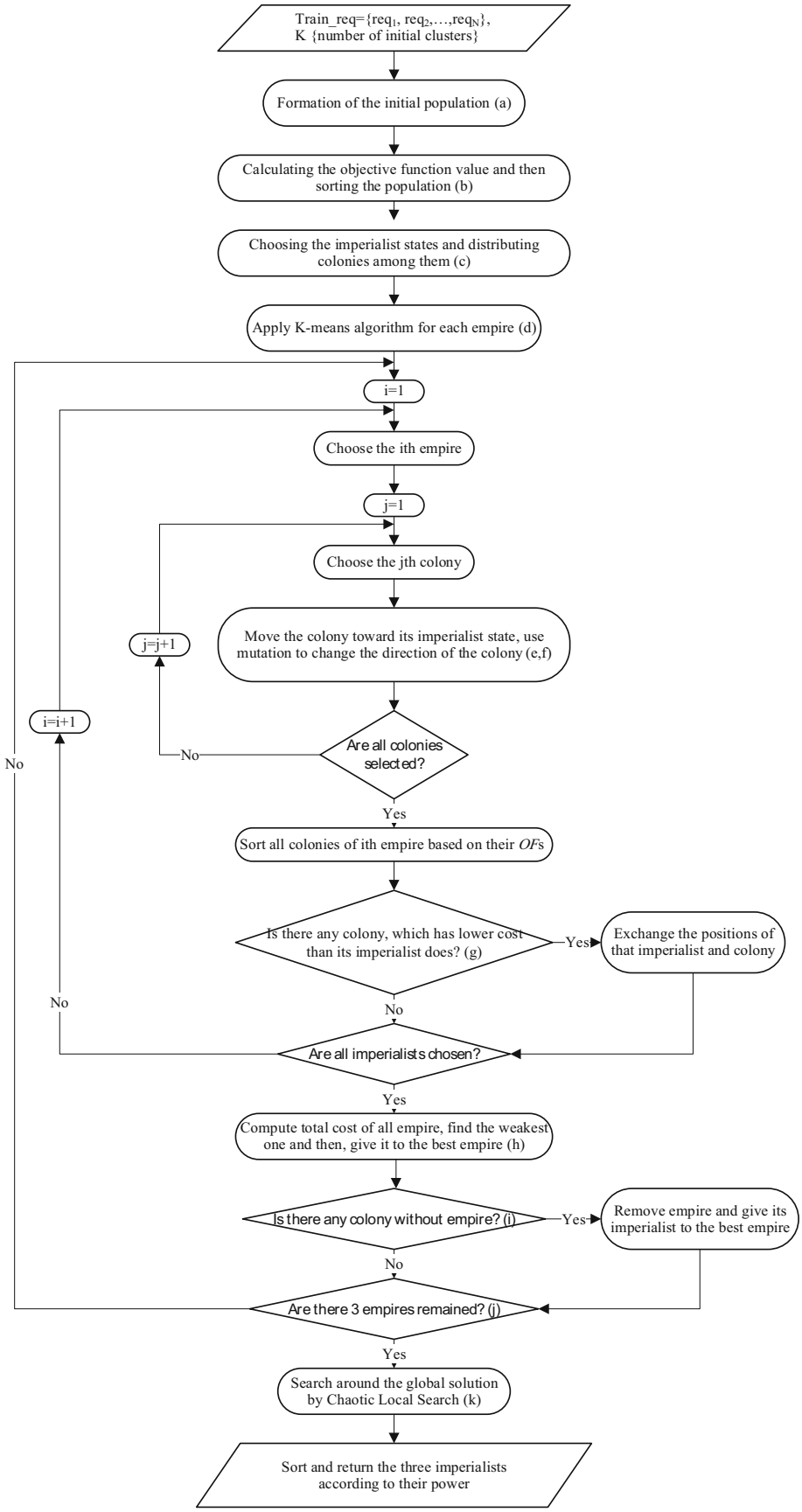


Fig. 3 Flowchart of the proposed clustering algorithm

where N_{imp} is the number of imperialists and C_n is the normalized objective function value (OF) of each imperialist which is defined by the following equation.

$$C_n = \min\{OF_i\} - OF_n, \quad (6)$$

where OF_n is the objective function value of nth imperialist.

3.2.2.4 Applying K-means for each empire K-means algorithm is applied for each empire. At first, K requests are chosen and set as cluster centers. After that, other requests are assigned to the closest centroid's cluster. When all requests have been assigned, the positions of the centroids are recalculated. This process is repeated unless the centroids are not changing. The pseudo code for the K-means algorithm is presented in Algorithm 1.

Algorithm 1. Pseudo code for K-means

```
            Input: K (The number of Clusters), N (a set of requests)
            Output: a set of K clusters
            Method:
1:          Begin
2:            do{
3:              Randomly choose K requests as cluster centers
4:              do{
5:                (Re) assign each object to the cluster to which the object is the
                  most similar, based on the mean value of the object in the cluster
6:                Recalculate the positions of the K centroids
7:              }
8:             while (the centroids move no longer)
9:              }
10:          while (all colonies are selected)
11:         End
```

3.2.2.5 The movement of colonies toward the imperialists After the formation of initial empires, the assimilation policy is executed. In this policy, the colonies move toward their relevant imperialist based on social-political metrics (objective function value). Based on Eq. (7), the movement toward the more powerful empires is performed.

$$X_{\text{new}} \sim (0, \beta * \text{d}) \quad (7)$$

A colony moves toward the imperialist by X_{new} units where X_{new} is a random variable within uniform distribution between 0 and β∗d, where β is a positive number greater than one, and d is the distance between the imperialist and the colony.

3.2.2.6 Changing the direction of colonies using mutation Inspired by the Genetic Algorithm, the mutation is applied to colonies to diversify the population of imperialists. The mutation is useful in circumstances in which the restoration of population diversity is required. Mutation improves the performance of the imperialist competitive algorithm by preventing premature convergence to local minima. In the mutation process, a random value of the uniform distribution is added in the interval [− 1,1] to a specific variable in the colony. The colonies are chosen at the predetermined mutation probability. When mutation makes a bad colony, we back to the old colony before mutation.

3.2.2.7 Checking the objective function value of all colonies During the mutation process, the objective function value of some colonies might have changed. In this step, the objective function value of all colonies of each empire is checked. If there is a colony that has a lower objective function than its imperialist, the position of the colony is exchanged with its relevant imperialist.

3.2.2.8 Checking the total objective function value of each empire The total objective function of each empire depends on the power of both colonies and their imperialist, which is calculated as Eq. (8).

$$TOF_n = OF(imperialist_n + \mu mean\{OF(coloniesofempire_n)\}) \quad (8)$$

TOF_n is the total objective function value of the nth empire and μ is an attenuation coefficient (0 to 1) to decrease the effect of colonies objective function value.

3.2.2.9 Eliminating the weakest empires When the weakest empires losses all of its colonies, they will collapse in the imperialistic competition because they have no colony. In this step, the weakest empire is eliminated and replaced by the next weakest empire.

3.2.2.10 Checking the number of empires If the number of empires is more than three do the imperialist competition. Else, the three imperialists are sorted and returned according to their power.

3.2.2.11 Running chaotic local search algorithm Imperialist Competition algorithm often converges to local optima. In order to avoid this weakness, a Chaotic Local Search algorithm is used to search around the global solution where one country chosen randomly is replaced with the best solution among them.

After clustering, the evaluation of its function should be done by test data. So, a workload as the data test is used in this phase. Each request from the test workload is compared with three imperialists by the Euclidean distance. The following equation shows how the distance between the ith test request ($Test\_req_i$) and the jth imperialists (Imp_j).

$$Dis_{i.Imp_j} = \sqrt{\sum_{k=1}^{4} \left(Test\_Req_i.NSLO_k - Imp_j.NSLO_k\right)^2} \quad (9)$$

In this way indexes of the requests' SLA are compared with the imperialists' SLA to define their distances. In the end, the ith request belongs to the empire with the minimum distance. The pseudo code for choosing the closest empire is presented in Algorithm 2.

Algorithm 2. Pseudo code for choosing the closest empire

Input: Test_req$_i$, $Imp=(Imp_1$ *or* Imp_2 *or* $Imp_3)\}$
Output: An Imperialist (Imp_1 *or* Imp_2 *or* Imp_3)

Method:
Begin
  For j = 1 to 3 **do**
    $Dis(j) = 0$
    For k = 1 to 4 **do**
      $Dis(j) = Dis(j) + (Test\_Req_i.NSLO_k - Imp_j.NSLO_k)^2$
    End for
    $Dis(j) = \sqrt{Dis(j)}$
  End for
Dis_{min}= min($Dis(1)$, $Dis(2)$ and $Dis(3)$)
If $Dis_{min}= Dis(1)$ **then** return (Imp_1)
  Else if $Dis_{min}= Dis(2)$ **then** return (Imp_2)
    Else if $Dis_{min}= Dis(3)$ **then** return (Imp_3)
End if
End

3.2.3 Resource provisioning

Resource provisioning is performed based on the workload volume type and CPU utilization type. The incoming workload volume is distinguished by its imperialist's type (Imp_1, Imp_2 *and* Imp_3). The average CPU utilization of the cloud platform in the time interval Δt is calculated by Eq. (10):

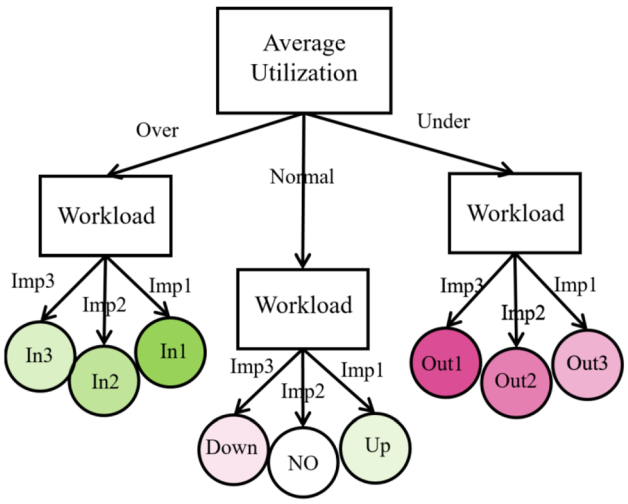


Fig. 4 Resource provisioning using the decision tree algorithm

$$AvgCPUUtilization(\Delta t) = \frac{\sum_{i=1}^{ActiveVMs} CPUUtilization(VM_i)}{ActiveVMs} \tag{10}$$

where $CPUUtilization(VM_i)$ is the CPU utilization of VM_i and the $ActiveVMs$ is the total number of operational VMs which have booted the start-up process and are being employed by the cloud platform. The CPU utilization of each VM is expressed by Eq. (11):

$$CPUUtilization(VM_i) = \frac{\sum_{j=1}^{CurrentCloudlets} \left(CloudletsPEs_j * CloudletLengths_j\right)}{PEs * MIPS} \tag{11}$$

where $CurrentCloudlets$ is the number of user requests running by the VM_i, the $CloudletsPEs_j$ is the number of processors required for a user request, and the $CloudletLengths_j$ is the number of instructions of a running user request. Besides, PEs (processing elements) is the number of the VM's processing elements and $MIPS$ (Million Instruction Per Second) is the processing power of each processing element [23, 24].

The CPU utilization is categorized into three types, Under_utilization, Normal_utilization and Over_utilization. U_t represents the utilization type, which is defined as Eq. (12). If utilization is less than the lower threshold (lt), U_t is considered under_utilization, and if the CPU utilization is higher than the upper threshold(ut), U_t is considered over_utilization. Otherwise, U_t is considered normal_utilization.

$$U_t = \begin{cases} Under\_utilization & U < lt \\ Normal\_utilization & lt \le U < ut \\ Over\_utilization & U \ge ut \end{cases} \tag{12}$$

Most research works and cloud providers use only two thresholds (i.e., lower threshold, upper threshold) for CPU Utilization metric. Thresholds are the key indicator for the correct working of the system performance. In some of cloud providers, the upper threshold is determined 85% [25, 26], and 80% [27], and RightScale's auto-scaling algorithm is set upper threshold = 0.7 and lower threshold = 0.3 [28, 29], and even in [30] a set of four thresholds is considered. In this study, we set ut = 0.80 and lt = 0.30 for improving system performance.

There are two categories for scaling cloud resources, vertical scaling, and horizontal scaling. Horizontal scaling is referred to adding/removing VMs from a cloud platform environment while vertical scaling is referred to reconfiguration at runtime the assigned resource capacities (e.g., CPUs and RAM) to a running VM. The Pseudo code of resource provisioning using a decision tree algorithm is shown in Algorithm 3.

Algorithm 3. Pseudo code for resource provisioning

Input: Test_req$_i$, *Imp=(Imp$_1$ or Imp$_2$ or Imp$_3$)}*
Output: Scaling decision (NO, In1,In2, In3, Out1, Out2, Out3, Up, Down)

Method:
Begin
Calculate the Utilization according to Equation (11)
Import the Imperialist type from algorithm 3
If ($U <$ lt) and (Test_req$_i \in$ *Imp$_3$*) **then** return (*Out$_1$*)
Else if ($U <$ lt) and (Test_req$_i \in$ *Imp$_2$*) **then** return (*Out$_2$*)
Else if ($U <$ lt) and (Test_req$_i \in$ *Imp$_1$*) **then** return (*Out$_3$*)
Else if (lt $\le U <$ ut) and (Test_req$_i \in$ *Imp$_3$*) **then** return (*Down*)
Else if (lt $\le U <$ ut) and (Test_req$_i \in$ *Imp$_3$*) **then** return (*Down*)
Else if (lt $\le U <$ ut) and (Test_req$_i \in$ *Imp$_2$*) **then** return (*NO*)
Else if (lt $\le U <$ ut) and (Test_req$_i \in$ *Imp$_1$*) **then** return (*Up*)
Else if ($U \ge$ ut) and (Test_req$_i \in$ *Imp$_3$*) **then** return (*In$_3$*)
Else if ($U \ge$ ut) and (Test_req$_i \in$ *Imp$_2$*) **then** return (*In$_2$*)
Else if ($U \ge$ ut) and (Test_req$_i \in$ *Imp$_1$*) **then** return (*In$_1$*)
End if
End

Figure 4 shows the decision tree formation for resource provisioning. As can be seen in the figure, "Up" and "Down" represent vertical resource scaling and "out" and "In" represent horizontal resource scaling. There are three

Table 4 Datacenter structure

Architecture	Operating system	Virtual machine manager
X64	Cloud Linux	XEN

Table 5 Host specification

Bandwidth	Main memory (GB)	Frequency (MIPS)	Name
10 Gbit/s	128	4096	Host1

levels for vertical resource scaling, Out1 > Out2 > Out3 and In1 > In2 > In3. No Operation (NO) scaling decision is used when there is no need to scale the resources. For example, when the system state is Over_utilization, and the workload belongs to the most powerful imperialist (Imp_1), the resource provisioning will be "*In1*" which result in a significant increase in horizontal resource scaling. The requests are considered to be of three types because the relevant studies, which we compare the proposed study with, also clustered the requests into three types [4, 6]. However, the number of request types could potentially be extended to higher numbers.

4 Performance evaluation

In this section, we evaluate the proposed approach (DRP_CICA) for dynamic resource provisioning based on K-means clustering and Decision Tree in a cloud computing environment. We first explain the experimental setup and performance metrics, and then the experimental results are discussed.

4.1 Experimental setup

In this section, we explain the simulation setup in more details. The Cloudsim toolkit [31] as a simulation framework is utilized for modelling and developing the cloud computing infrastructures. In the analyses, all cloud servers have the same specification, as shown in Tables 4 and 5.

Table 7 Parameters related to RUBiS benchmark

Parameter	Web tier	App tier	DB tier
Single VM capacity(req/sec)	90	150	180
VMs number	16	10	8
Max throughput(%)	0.85	0.85	0.85

Also, we can consider three types of VMs offered by any cloud provider: extra-large, medium, and small. The configuration specification of different types of VMs with different capabilities inspired by Amazon EC2 [32], as shown in Table 6.

Furthermore, we utilized the RUBiS application benchmark [33] for modelling and autonomous managing multi-tier cloud application on the cloud infrastructure. The RUBiS as an open-source benchmark is a prototype of an auction web application modelled by eBay which provides three functionality namely, selling, browsing and bidding for supporting various user roles namely buyer, seller, and visitor, respectively. It can control the number of user requests per second and the number of cloud servers using XML files for each tier separately. Also, we utilized PHP, Sun Java System application, and MySQL for implementing the web tier, application tier, and database tier of RUBiS application benchmark, respectively. In this study, we set different values for the parameters of the RUBiS benchmark in order to validate the proposed solution, as shown in Table 7.

To evaluate the proposed approach, two real workload traces are used: the FIFA traces [34], and the NASA traces [35]. These workload traces include real load variations over time from recognized websites, and they have distinctive attributes, which make the outcomes more realistic to be applied in real cloud environments. FIFA and the NASA traces for 24 h are shown in Fig. 5a and b, respectively. In this paper, the simulation time is 24 h, and

Table 6 Virtual machines specification

Machin Name	CPU(MIPS)	RAM (GB)	Storage (GB)	BW (Gbps)	Price ($ per Hour)
t2.small	10,200	2	1 GB–16 TB	100Mbps	0.023
m3.Medium	12,000	3.75	1 × 4 GB	1Gbps	0.070
m4.4Xlarge	15,000	64	1 GB–16 TB	1Gbps	0.862
r3.4Xlarge	80,000	122	1 × 320 GB	10Gbps	1.330
m4.10Xlarge	97,000	160	1 GB–16 TB	10Gbps	2.155
d2.4Xlarge	105,000	122	12 × 2000 GB(24 TB)	10Gbps	2.76
m4.16Xlarge	280,000	256	1 GB–16 TB	100Gbps	3.447
r4.16Xlarge	350,000	488	1 GB–16 TB	100Gbps	4.256
d2.8Xlarge	500,000	244	24 × 2000 GB(48 TB)	100Gbps	5.52

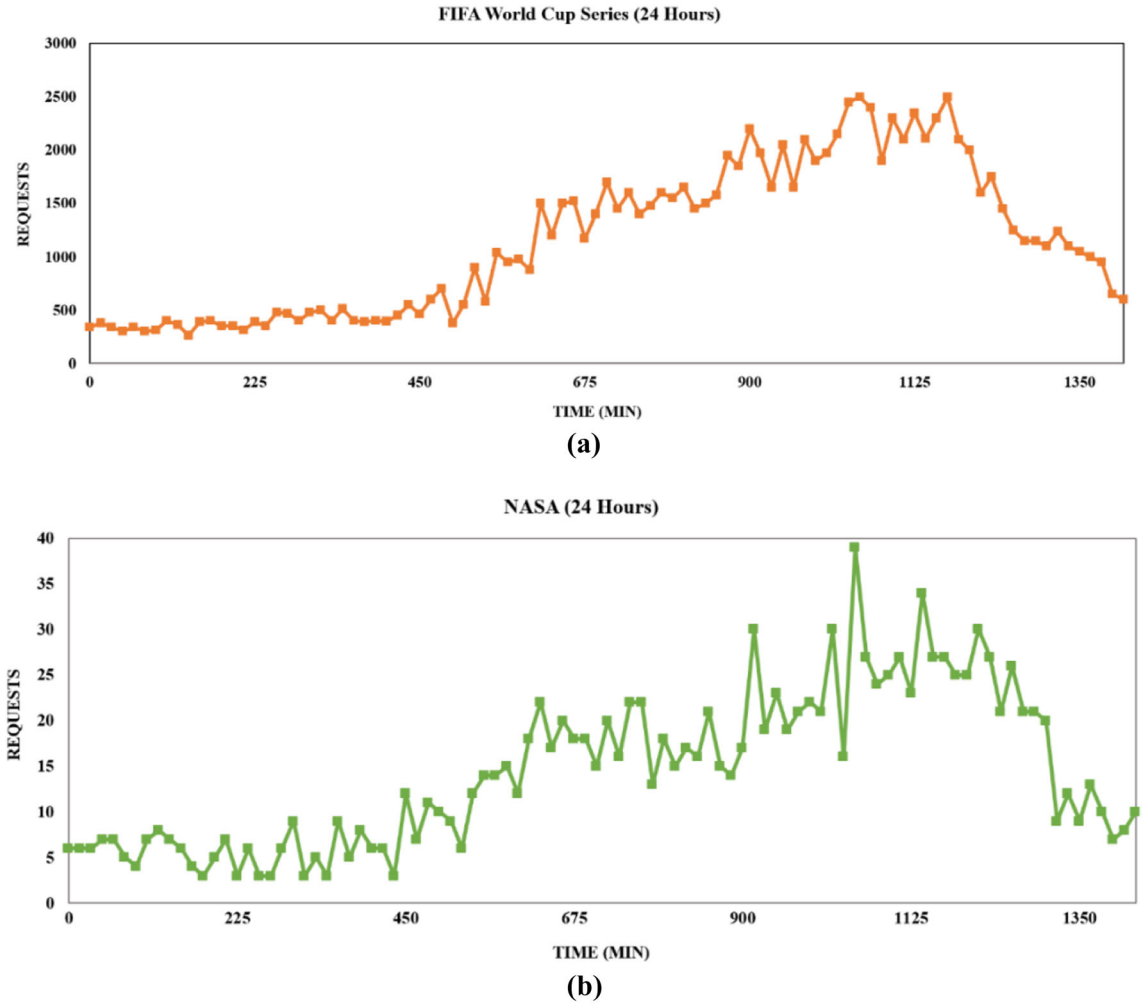


Fig. 5 Real workload traces **a** FIFA and **b** NASA

the time intervals are considered in 15-min intervals. Thus, each day includes 96-time intervals.

4.2 Performance metrics

We use the following metrics for comparing the proposed approach with other studies; costs, response time, CPU utilization, and elasticity.

Cost: Total Cost incurred by the cloud provider to serve all requests, and as shown in Eq. (13), it includes the VM Costs and the Penalty Costs [36].

$$TotalCost = VMCost + PenaltyCost \tag{13}$$

The VM Cost is defined by Eq. (14), which includes the cost of all VMs.

$$VMCost = \sum_{i=1}^{M} VMPrice_i * (VM\_hour_i + VM\_Init_i) \tag{14}$$

VMPrice is the price of a virtual machine for a client request, which is measured by \$/h. *VM_hour* is the time duration for responding to the requests measured by the hour. VM_Init_i is the cost of the initial adjustment of the virtual machines.

The Penalty Costs is the total penalty cost for all user requests and is expressed by

$$Penaltycost = \sum_{i=1}^{M} R_i.\alpha penalty \times R_i.penalty \times SLAV_i \tag{15}$$

$$SLAV_i = \begin{cases} Response_i - Deadline_i & Response_i > Deadline_i \\ 0 & otherwise \end{cases} \tag{16}$$

when calculating the penalty cost for M user requests (R), $R_i.\alpha penalty$ is the penalty increase coefficient per repeated SLA violation and $R_i.penalty$ is the penalty rate. $SLAV_i$, SLA violation, is the difference between the real response time of a request ($Response_i$) and the user desired response time ($Deadline_i$).

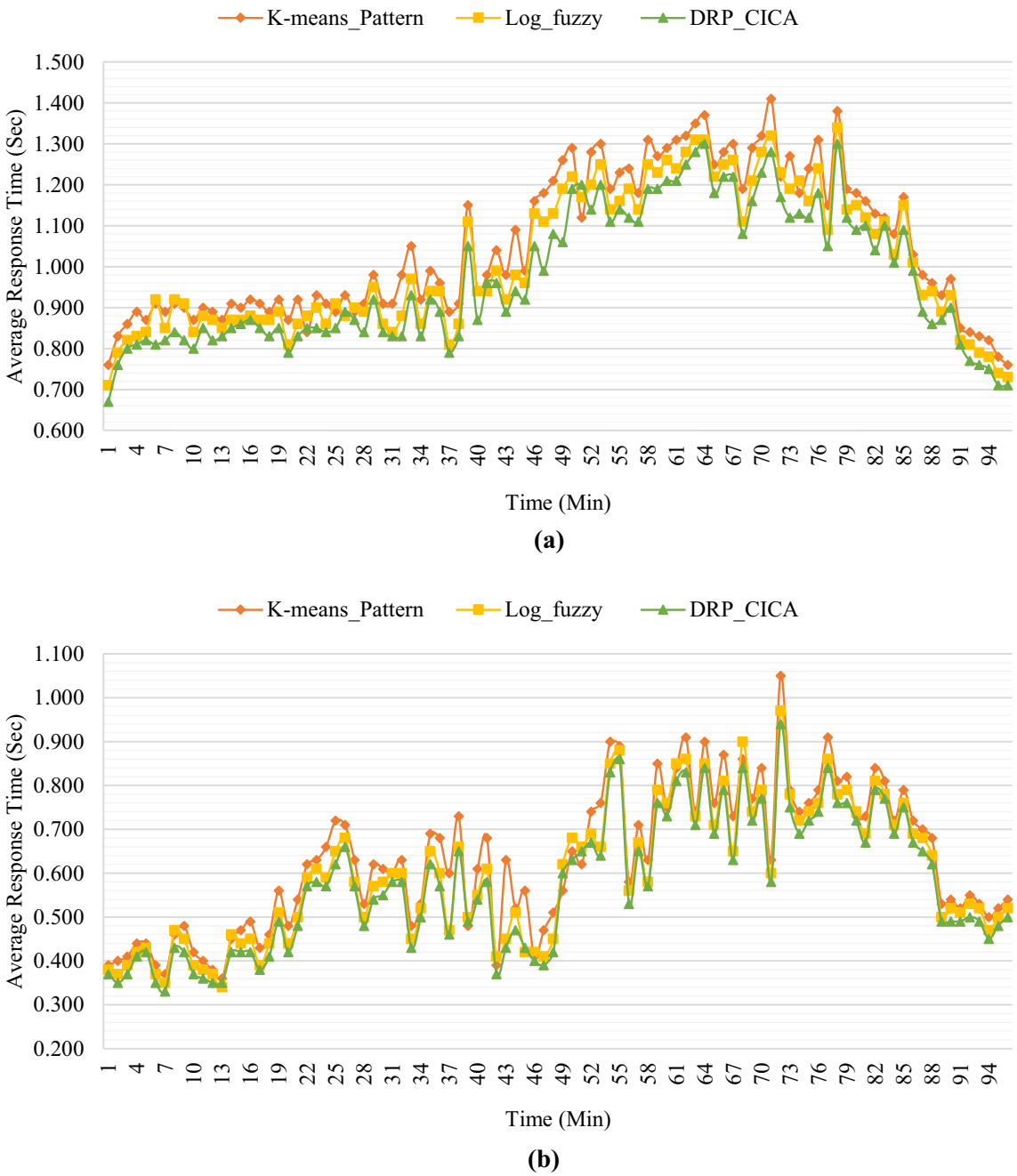


Fig. 6 Average response time of different approaches for **a** FIFA and **b** NASA

Response time: The response time of a user request is defined as the amount of time taken to process the user request by the cloud platform. Indeed, the actual response time is the time difference between the user request arrival time and the first response time received from the user by the cloud platform. The response time (RT) of user request is calculated by Eq. (17):

$$RT = FT - PT - AT, \tag{17}$$

where FT is the first time the user request is responded, PT is the duration time required to perform the user request by the VM, and AT is the time of user request arrive into the cloud platform. More information about the desired response time of user request submitted to the cloud platform has been discussed in [37].

Utilization: The CPU utilization is defined as the ratio of the average amount of the allocated MIPS of VMs for serving user requests to the average amount of the total

Table 8 The average response time for FIFA and NASA workloads

	DRP_CICA	K-means_Pattern	Log_fuzzy
FIFA	0.972	1.052	1.011
NASA	0.574	0.627	0.596

MIPS that is potentially offered by VMs into the cloud platform, and is calculated by Eqs. (10) and (11).

Elasticity: Let E be the elasticity of a cloud computing platform which is measured by the summation of time when the cloud platform is in normal_provisioning states. T denotes the total time that a system is working for a sufficiently long time period, T_o be the total time period that the system is in the over_provisioning state (i.e., the amount of provided resources are higher than the resource required for serving user requests), T_u be the total time period that the system is in the under_provisioning state (i.e., the amount of provided resources are lower than the resource required for serving user requests), and T_n be the total time period that the system is in the normal_provisioning state. Therefore, T includes all the periods in the normal, over_provisioning, and under-provisioning states; that is, $T = T_o + T_u + T_n$. Therefore, the *Elasticity* value is expressed by Eq. (18) [38]:

$$E = \frac{T_n}{T} = 1 - \left[\frac{T_o + T_u}{T}\right] \tag{18}$$

On the other hand, let P_o, P_u, and P_n denote the probability that the system is in the over_provisioning state, under_provisioning state, and normal_provisioning state, respectively and are expressed by Eq. (19):

$$P_o = \frac{T_o}{T}, P_u = \frac{T_u}{T}, P_n = \frac{T_n}{T} \tag{19}$$

In this case, the *Elasticity* value is calculated by Eq. (20):

$$E = P_n = 1 - (P_o + P_u) \tag{20}$$

4.3 Results and discussion

To evaluate the performance of the proposed approach, we design various scenarios based on two real workload traces and performance metrics that were discussed in the previous subsections.

The proposed approach is compared with K-means_Pattern [5] and Log_fuzzy methods [3]. Both methods operate on the multi-layer cloud. In K-means_Pattern method, the workloads are categorized based on common patterns using the K-means clustering algorithm. They present a QoS-aware resource provisioning technique based on the QoS requirements (SLOs) of cloud workloads to find the best workload and resource match with reliable cloud services without SLA violation. In Log_fuzzy method, they used log smoothing and a fuzzy logic-based approach to make workload prediction. Smoothing deals with log data variability and allows detecting trends and uncertainty of the monitored workload is addressed by a fuzzy logic enhanced prediction. For the implementation, CloudSim toolkit is used where datacenters, VMs, and resource provisioning policies are supported by this toolkit. Extending the toolkit, we developed the K-means-Pattern and Log_fuzzy techniques to test their performance. Compared to the proposed method, the same workloads were considered as the input for both simulations; however, SLOs were used as the QoS metrics.

In the following, the proposed approach (DRP_CICA) is compared with K-means_Pattern and Log-fuzzy in terms of response time, cost, utilization, and elasticity under two different workloads.

4.3.1 Response time

Response time is one of the essential goals for SLA to shrink costs and provide user satisfaction. Response time has a great impact on the selection of virtual machines. If the requested response time that is known as the response time limit is not achieved, scaling should be done. To investigate the average response time, different approaches were fed with two workloads. As shown in Fig. 6, the response time of the proposed approach is less than K-means_Pattern and Log_fuzzy algorithms under different workloads. For each variation of input workload, the related response time changes between lower and upper bounds depending on the number of available VMs and current load balance. The reason is that the hybrid proposed algorithm has a better response and also converges faster than ordinary evolutionary algorithms. However, the *OF* of the K-means_Pattern method is not convex, and therefore, it might contain many local minima. The output clusters of the K-means_Pattern algorithm is highly depended on the initial selection of the cluster centers. The average response time of different methods is shown in Table 8. Under the FIFA workload, the proposed approach reduces the response time by up to 7.5% and 3.8% compared with K-means_Pattern and Log_fuzzy, respectively. And under the NASA workload, the proposed method reduces the response time by up to 9.1 and 3.7% Nasa compared with K-means_Pattern and Log_fuzzy, respectively.

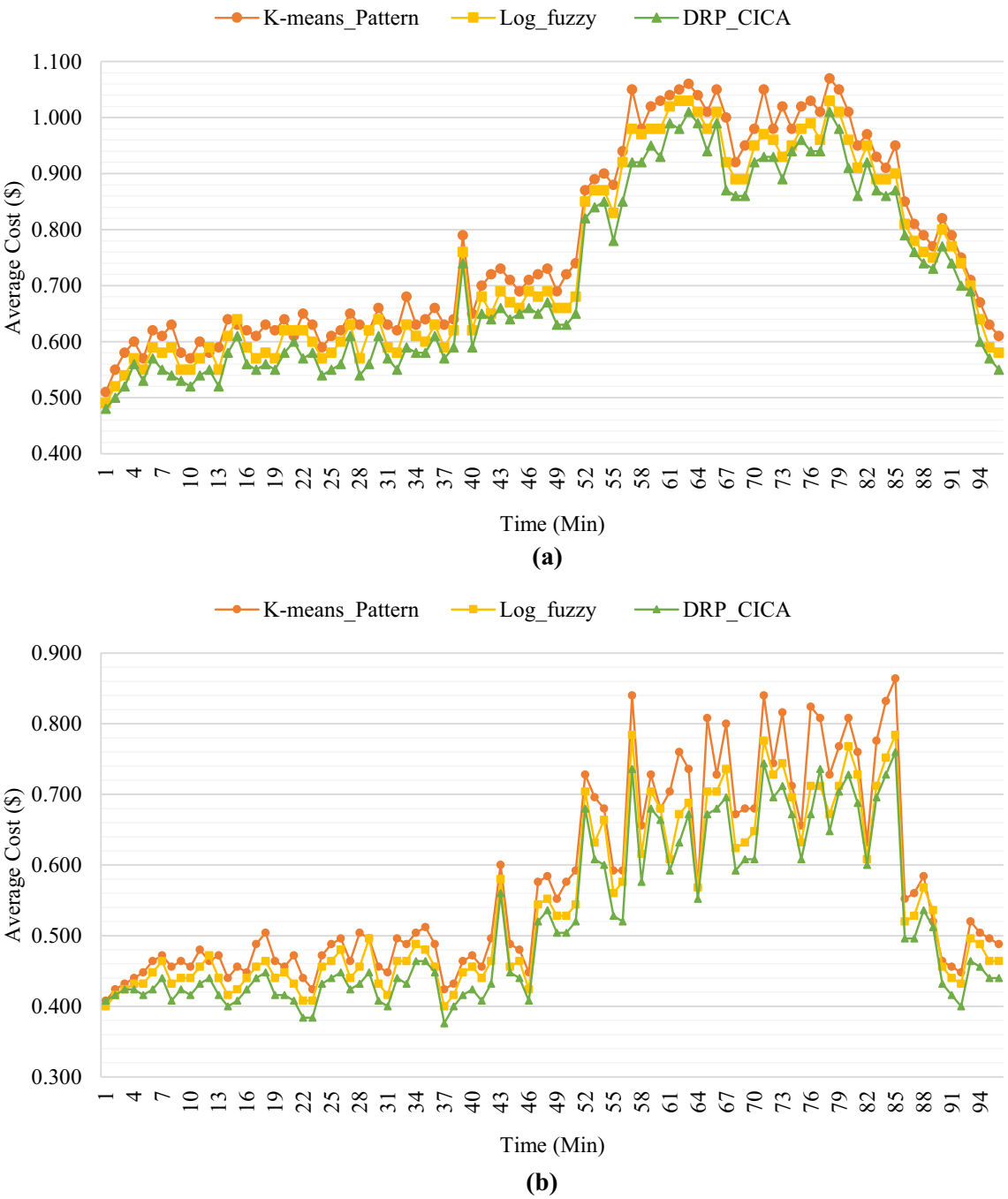


Fig. 7 The average cost of different approaches for **a** FIFA and **b** NASA

Table 9 The average cost for FIFA and NASA workloads

	DRP_CICA	K-means_Pattern	Log_fuzzy
FIFA	0.712	0.776	0.743
NASA	0.516	0.572	0.540

4.3.2 Cost

In this section, the average cost of the proposed approach is evaluated and compared with ADRP_AL and ADRP_RL algorithms. The main condition of the service is the user compromised cost. If the cost of service increases, the benefit of the service provider is reduced and to the same extent the degree of user discontent. The results of the comparison of the average cost for different approaches are

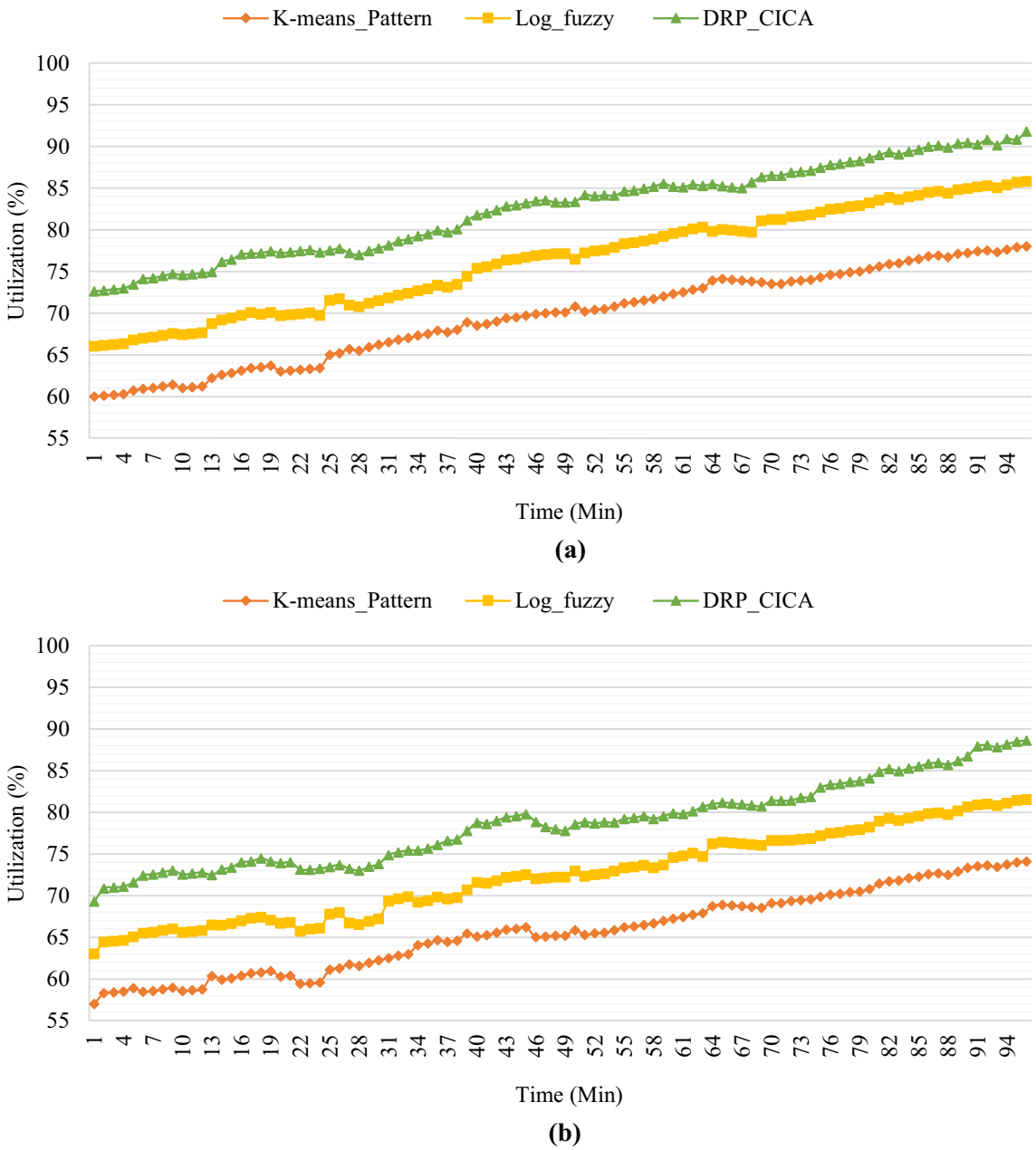


Fig. 8 The average CPU Utilization at different intervals for **a** FIFA and **b** NASA

Table 10 The average CPU Utilization for FIFA and NASA workloads

	DRP_CICA	K-means_Pattern	Log_fuzzy
FIFA	82.5	69.5%	76.3%
NASA	78.6%	65.6%	72.4%

shown in Fig. 7. The figure displays the impact of changing the workload on the average cost. As can be seen in the figure, the proposed approach is more cost-efficient than the other approaches. Clustering the workload based on K-means and imperialist competitive algorithm and resource provisioning using the Decision Tree algorithm results in a more accurate resource provisioning compared to previous studies. Consequently, fewer requests will be in the waiting queue of the cloud interface. Therefore, response time will be reduced, and fewer penalties occur, and the cost of providing services will be reduced. The average cost of different approaches, under two real workloads, is shown in Table 9. Under the FIFA workload, the proposed approach reduces the cost by up to 8.2% and 4.2% compared with K-means_Pattern and Log_fuzzy,

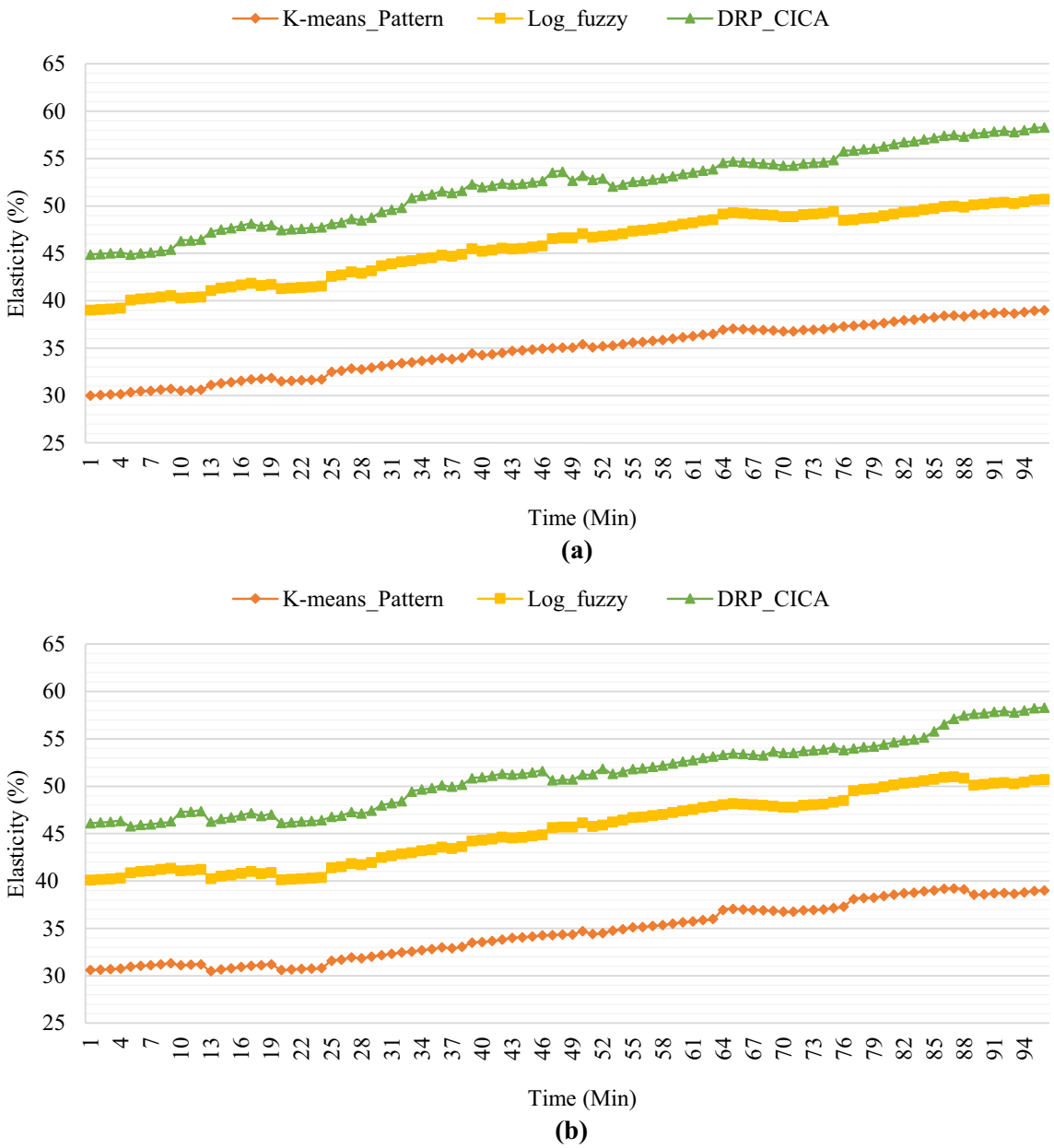


Fig. 9 The average elasticity value at different intervals for **a** FIFA and **b** NASA

Table 11 The average elasticity for FIFA and NASA workloads

	DRP_CICA (%)	K-means_Pattern (%)	Log_fuzzy (%)
FIFA	51.9	34.8	45.7
NASA	51.1	34.5	45.3

respectively. And under the NASA workload, the proposed method reduces the cost by up to 9.8 and 4.4% Nasa compared with K-means_Pattern and Log_fuzzy, respectively.

4.3.3 Utilization

Figure 8a shows the average CPU utilization of the four approaches under the Google cluster traces at each interval. From the results, we observe that the DRP_CICA approach has outperformed in term of CPU utilization, and it can utilize resources more fully. In the proposed approach, workload clustering is performed using combined K-mean clustering and the imperialist competitive algorithm and Decision Tree algorithm is applied for resource provisioning where workload detection and analysis are conducted more accurately compared to other approaches.

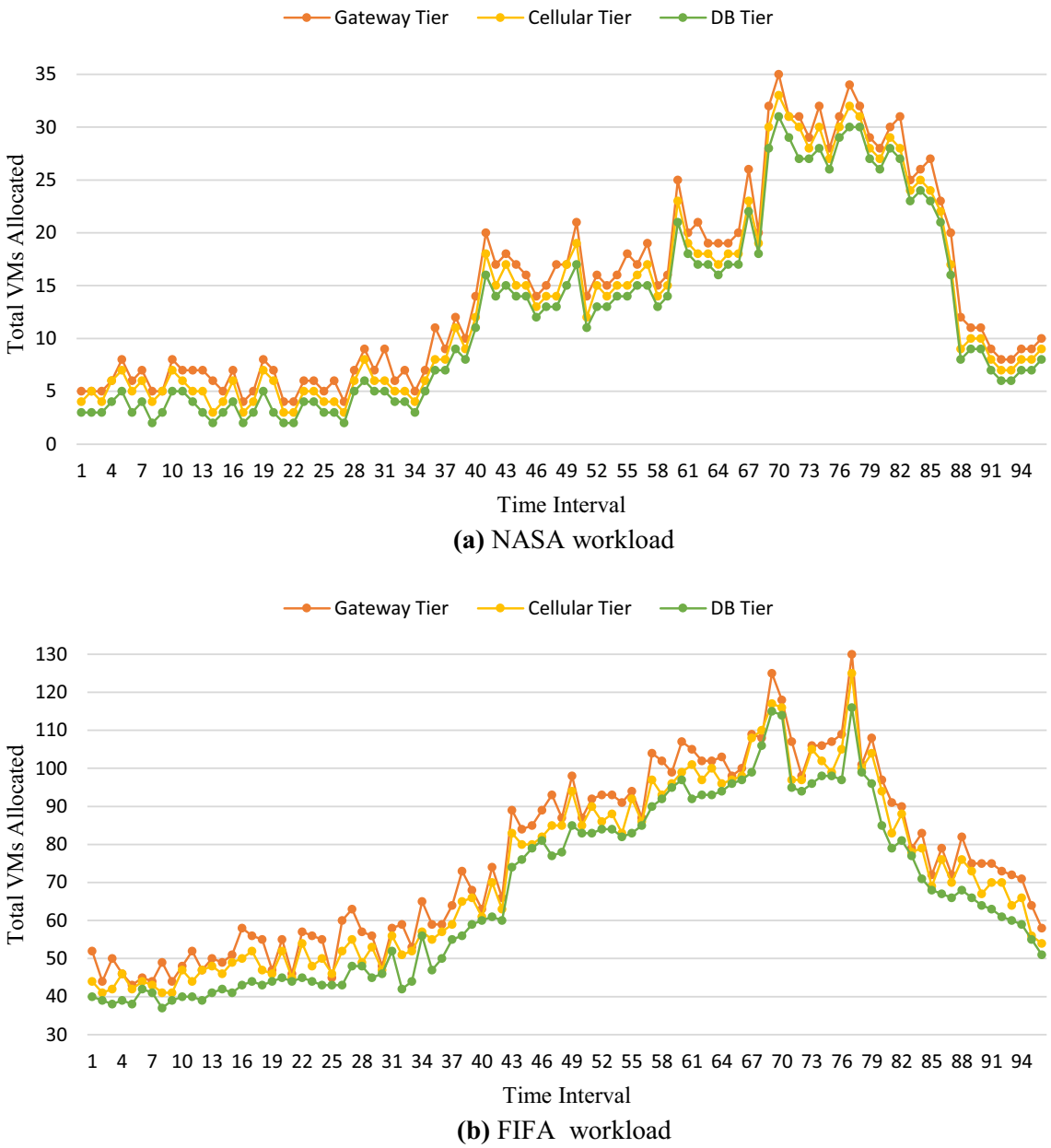


Fig. 10 Comparison between the numbers of allocated VMs in each tier under various workloads in the proposed solution

Table 12 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

From the results, we observe that the DRP_CICA approach in terms of CPU utilization metric is higher than K-means_Pattern and Log_fuzzy approaches under two real workload traces, as shown in Table 10. The elasticity values of the DRP_CICA approach compared with K-means_Pattern and Log_fuzzy approaches are enhanced by 18.7% and 8.1% under the FIFA traces, 19.8% and 8.5% under the NASA traces.

4.3.4 Elasticity

One of the important metrics in comparing the performance of resource provisioning approaches is the elasticity metric. To experiment, 96-time intervals were selected from each

Table 13 Values of parameters for each 4 algorithms

Proposed		ICA		PSO		GA	
Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
N_{Pop}	100	N_{Pop}	100	$C_1 = C_2$	2	Population	100
N_{Imp}	3	N_{Imp}	3	ω_{min}	0.5	Mutation rate	0.001
β	5	β	5	ω_{max}	1	Crossover	0.8
μ	0.05	μ	0.05				

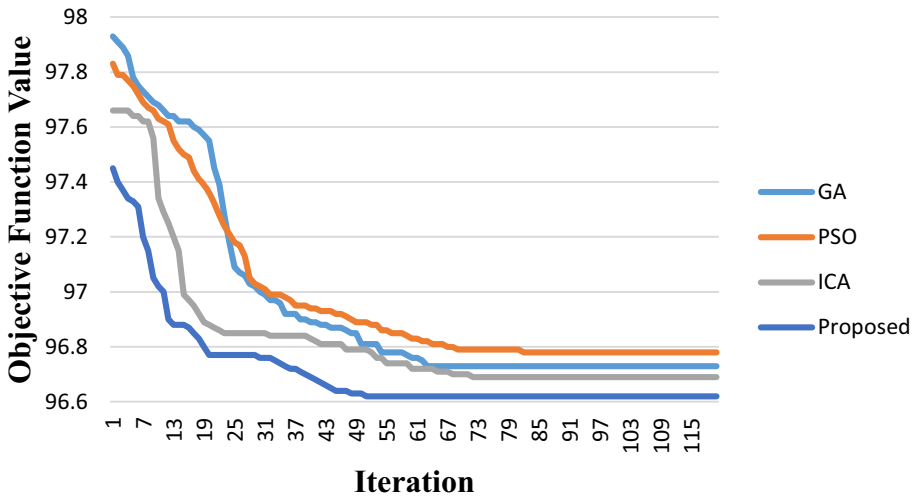


Fig. 11 Convergence characteristic of the proposed method, ICA, PSO, and GA for the best solutions

workload so that the number of requests was ascending from the first to 96th interval. As we mentioned earlier, the elasticity is defined as the degree to which a cloud computing platform adapted at the presence of workload fluctuations. Elasticity is measured by the percentage of time when the platform is in normal-provisioning states. When the elasticity tends to 100 percent, it means that the platform is always in normal provisioning state. Figure 9 shows that the elasticity increases when the number of requests is increasing. The reason is that increasing the number of requests results in a significant reduction in the probability of over-provisioning. Figure 9 shows the average elasticity value of the three approaches under the FIFA and NASA traces at each interval. The results show that the DRP_CICA approach in terms of elasticity metric is better than K-means_Pattern and Log_fuzzy approaches under two real workload traces, as shown in Table 11. The elasticity values of the DRP_CICA approach compared with K-means_Pattern and Log_fuzzy approaches are enhanced by 49.1% and 13.5% under the FIFA traces, 48.1% and 12.8% under the NASA traces.

4.3.5 VMs allocated

One of the most important performance metrics is the number of VMs allocated to the multi-tier cloud application. Figure 10 illustrates the average number of VMs allocated under NASA and FIFA workload traces at each tier of multi-tier cloud application by the proposed solution, separately. From the simulation results, we indicate that the web tier required a number of VMs due to it has more interaction with end-users. Also, since the database tier utilized virtual memory or storage, it will require the minimum number of VMs compared with other tiers. Note that, the disk is the most important resource in the database tier while main memory (i.e., RAM) and processing power (i.e., CPU) are the most important resource in the web and application tiers. It is clear that resource allocation in web tier is important, because of in the multi-tier cloud application, the web tier is as input tier of user requests. For instance, consider an online store multi-tier application where customer requests are usually often to browse and often, they don't need access to the lower tiers of the multi-tier application (i.e., database tier).

4.3.6 Convergence speed results

In this section, we present the convergence speed of different algorithms over iterations. The value range of each request, including response time, cost, availability and reliability are shown in Table 12. Parameter settings for the

proposed, ICA, PSO and GA algorithms are illustrated in Table 13.

Figure 11 shows the convergence characteristics of the proposed method, ICA, PSO, and GA for the best solutions on 100 requests of NASA data set. In this figure, the x-axis represents the objective function value defined by Eq. (4), and the y-axis represents iterations.

Simulation results show that the proposed algorithm converges to the global optimum after 48 iterations while ICA, PSO and GA algorithms converge to the global optimum after 72, 69, and 63 iterations, respectively. Moreover, the proposed algorithm shows better results on the best solution. The best global solutions for the proposed method, ICA, PSO, and GA are 96.63, 96.68, 96.78, and 96.73.

5 Conclusion

The usage and popularity of cloud computing as one of the most popular internet-based technologies for offering the services of the computing power or infrastructure to IT organizations for serving cloud workloads are increasing every day in the industrial community and is anticipated to grow further. The heterogeneous cloud workloads submitted by end-users to cloud providers in the form of web services, financial services, online transaction processing services, mobile computing services, and graphics-based services with different QoS requirements in the form of SLA. Identification and analysis of cloud workloads to find their QoS requirements can play an important role in efficient resource provisioning in a cloud environment. In this work, we studied the resource provisioning problem using workload clustering in a cloud environment. Firstly, we present a hybrid solution for clustering the cloud workload according to QoS requirements using ICA and K-means algorithms. Then, we propose a decision tree technique based on workload volume and CPU utilization metrics to determine scaling decisions for resource provisioning to serve heterogeneous cloud workloads. We carried out a series of experiments under NASA and FIFA real workload traces to validate the proposed solution in terms of the total cost, response time, CPU utilization, and elasticity metrics. Our simulation results illustrate a significant improvement in related performance metrics compared with other methods. In the future, we plan to validate our solution under heterogeneous real cloud workload traces with their QoS requirements on the OpenStack and Azure cloud platforms. We project to develop an autonomic computing paradigm to manage large-scale database workload in a data warehouse and extension of workload clustering phase using the support vector regression (SVR) model as a supervised learning technique for classifying various cloud workloads.

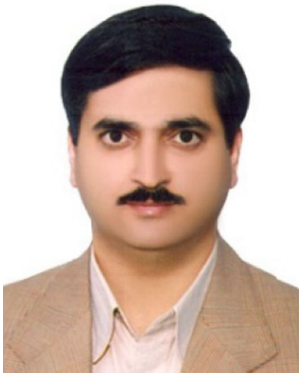
References

1. Chandrasekaran, K.: Essentials of cloud computing. Chapman and Hall/CRC, Boca Raton (2014)
2. Ghobaei-Arani, M., Souri, A.: LP-WSC: a linear programming approach for web service composition in geographically distributed cloud environments. J. Supercomput. **75**(5), 2603–2628 (2019)
3. Chaisiri, S., Lee, B.-S., Niyato, D.: Optimization of resource provisioning cost in cloud computing. IEEE Trans. Serv. Comput. **5**(2), 164–177 (2011)
4. Zhang, L., Zhang, Y., Jamshidi, P., Xu, L., Pahl, C.: Service workload patterns for Qos-driven cloud resource management. J. Cloud Comput. **4**(1), 23 (2015)
5. Mian, R., Martin, P., Vazquez-Poletti, J.L.: Provisioning data analytic workloads in a cloud. Fut. Gener. Comput. Syst. **29**(6), 1452–1458 (2013)
6. Singh, S., Chana, I.: Q-aware: Quality of service based cloud resource provisioning. Comput. Electr. Eng. **47**, 138–160 (2015)
7. Silva Filho, T.M., Pimentel, B.A., Souza, R.M., Oliveira, A.L.: Hybrid methods for fuzzy clustering based on fuzzy c-means and improved particle swarm optimization. Exp. Syst. Appl. **42**(17), 6315–6328 (2015)
8. Niknam, T., Fard, E.T., Pourjafarian, N., Rousta, A.: An efficient hybrid algorithm based on modified imperialist competitive algorithm and K-means for data clustering. Eng. Appl. Artif. Intell. **24**(2), 306–317 (2011)
9. Singh, S., Chana, I., Singh, M.: The journey of QoS-aware autonomic cloud computing. IT Professional **19**(2), 42–49 (2017)
10. Kaur, P., Mehta, S.: Resource provisioning and work flow scheduling in clouds using augmented Shuffled Frog Leaping Algorithm. J. Parallel Distrib. Comput. **101**, 41–50 (2017)
11. Haghighi, M.A., Maeen, M., Haghparast, M.: An energy-efficient dynamic resource management approach based on clustering and meta-heuristic algorithms in cloud computing IaaS platforms. Wireless Pers. Commun. **104**(4), 1367–1391 (2019)
12. Singh, S., Chana, I., Buyya, R.: STAR: SLA-aware autonomic management of cloud resources. IEEE Trans. Cloud Comput. (2017). https://doi.org/10.1109/TCC.2017.2648788
13. Chen, J., Zhu, X., Bao, W., Wu, G., Yan, H., Zhang, X.: TRIERS: traffic burst oriented adaptive resource provisioning in cloud. J. Phys. **1168**(3), 032061 (2019)
14. Gill, S.S., Buyya, R., Chana, I., Singh, M., Abraham, A.: BULLET: particle swarm optimization based scheduling technique for provisioned cloud resources. J. Netw. Syst. Manag. **26**(2), 361–400 (2018)
15. Suresh, A., Varatharajan, R.: Competent resource provisioning and distribution techniques for cloud computing environment. Clust. Comput. (2017). https://doi.org/10.1007/s10586-017-1293-6
16. Cheng, M., Li, J., Nazarian, S.: DRL-cloud: deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In: Proceedings of the 23rd Asia and South Pacific Design Automation Conference, pp. 129–134. IEEE Press (2018). https://doi.org/10.1109/ASPDAC.2018.8297294
17. Gong, S., Yin, B., Zheng, Z., Cai, K.-Y.: An adaptive control method for resource provisioning with resource utilization constraints in cloud computing. Int. J. Comput. Intell. Syst. **12**(2), 485–497 (2019)

18. Gill, S.S., Buyya, R.: 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 (2019). https://doi.org/10.1007/s10723-017-9424-0
19. Moreno-Vozmediano, R., Montero, R.S., Huedo, E., Llorente, I.M.: Efficient resource provisioning for elastic Cloud services based on machine learning techniques. J. Cloud Comput. **8**(1), 5 (2019)
20. Feng, D., Wu, Z., Zuo, D., Zhang, Z.: ERP: an elastic resource provisioning approach for cloud applications. PLoS ONE **14**(4), e0216067 (2019)
21. Erradi, A., Iqbal, W., Mahmood, A., Bouguettaya, A.: Web application resource requirements estimation based on the workload latent features. IEEE Trans. Serv. Comput. (2019). https://doi.org/10.1109/TSC.2019.2918776
22. Ramesh, K., Pandey, A.: An improved normalization technique for white light photoelasticity. Opt. Lasers Eng. **109**, 7–16 (2018)
23. Aslanpour, M.S., Dashti, S.E., Ghobaei-Arani, M., Rahmanian, A.A.: Resource provisioning for cloud applications: a 3-D, provident and flexible approach. J. Supercomput. **74**(12), 6470–6501 (2018)
24. Ghobaei-Arani, M., Shamsi, M., Rahmanian, A.A.: An efficient approach for improving virtual machine placement in cloud computing environment. J. Exp. Theor. Artifi. Intell. **29**(6), 1149–1171 (2017)
25. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Fut. Gener. Comput. Syst. **27**(6), 871–879 (2011)
26. Chuprikov, P., Nikolenko, S., Kogan, K.: On demand elastic capacity planning for service auto-scaling. In: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9. IEEE (2016). https://doi.org/10.1109/INFOCOM.2016.7524616
27. Qavami, H.R., Jamali, S., Akbari, M.K., Javadi, B.: Dynamic resource provisioning in cloud computing: a heuristic markovian approach. In: International conference on cloud computing, pp. 102–111. Springer (2013). https://doi.org/10.1007/978-3-319-05506-0_10
28. Koperek, P., Funika, W.: Dynamic business metrics-driven resource provisioning in cloud environments. In: International Conference on Parallel Processing and Applied Mathematics, pp. 171–180. Springer (2011). https://doi.org/10.1007/978-3-642-31500-8_18
29. Hasan, M.Z., Magana, E., Clemm, A., Tucker, L., Gudreddi, S.L.D.: Integrated and autonomic cloud resource scaling. In: 2012 IEEE network operations and management symposium, pp. 1327–1334. IEEE (2012). https://doi.org/10.1109/NOMS.2012.6212070
30. https://support.rightscale.com/03-Tutorials/02-AWS/index.html.
31. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software **41**(1), 23–50 (2011)
32. Ghobaei-Arani, M., Souri, A., Baker, T., Hussien, A.: ControCity: an autonomous approach for controlling elasticity using buffer Management in Cloud Computing Environment. IEEE Access **7**, 106912–106924 (2019). https://doi.org/10.1109/ACCESS.2019.2932462
33. *OW2 Consortium, RUBiS: An auction site prototype, 1999,* https://rubis.ow2.org/.
34. "FIFA. 2014. 1998 World Cup Web Site Access Logs—The Internet Traffic Archive. Retrieved March 27, 2018 from https://ita.ee.lbl.gov/html/contrib/WorldCup.html
35. *Nasa-http- two months of http logs from the kscnasa www server.* https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html
36. Ghobaei-Arani, M., Khorsand, R., Ramezanpour, M.: An autonomous resource provisioning framework for massively multiplayer online games in cloud environment. J. Netw. Comput. Appl. **142**, 76–97 (2019). https://doi.org/10.1016/j.jnca.2019.06.002
37. Shahidinejad, A., Ghobaei-Arani, M. and Esmaeili, L.: An elastic controller using Colored Petri Nets in cloud computing environment. Cluster Computing, pp.1–27 (2019)
38. Li, K.: Quantitative modeling and analytical calculation of elasticity in cloud computing. IEEE Trans. Cloud Comput. (2017). https://doi.org/10.1109/TCC.2017.2665549

Ali Shahidinejad received the B.S. degree in computer hardware engineering from Islamic Azad University of Kashan, Iran in 2008, the M.S. degree in computer architecture from Islamic Azad University of Arak, Iran, in 2010 and Ph.D. degree in Computer Networks at the Universiti Teknologi Malaysia/RWTH Aachen University, Malaysia/Germany, in 2015. He joined the Department of Computer Engineering, Islamic Azad University of Qom, as an Assistant Professor. He is currently the head of department of higher educations in computer engineering at Islamic Azad University of Qom. His research interests include Quantum-dot Cellular Automata, Optical Wireless Communications, Micro Ring resonators, Network on Chip, Cloud Computing, Internet of Things and Network Security.

Mostafa Ghobaei-Arani received the Ph.D. degree in software engineering from Islamic Azad University, Science and Research Branch, Tehran, Iran. He is assistant professor of Computer Engineering Department, Qom Branch, Islamic Azad University, Qom, Iran. He has published more than 50 journal and conference papers in the area of distributed computing. His research interests include distributed computing, cloud computing, autonomic computing, edge/fog computing, exascale computing, soft computing, and the IoT. He has served as a member of editorial board and review committee for a number of peer-reviewed international journals and PC member of various international conferences (https://publons.com/researcher/1267819/mostafa-ghobaei-arani/).



Mohammad Masdari received his B.Tech. Degree in Computer Software Engineering from Islamic Azad University, Qazvin Branch, Iran, in 2001, and M.Tech degree in Computer Software Engineering from Islamic Azad University, South Tehran Branch, Tehran, Iran, in 2003. He received his Ph.D. degree in Computer Software Engineering from Islamic Azad University, Science and Research Branch, Tehran, Iran, in 2014. Since 2003, he worked a faculty member of Islamic Azad University, Urmia Branch, Iran. Presently he is an Assistant Professor in the Department of Computer Engineering of Islamic Azad University, Urmia Branch, Iran. His research interests include Distributed Systems and Network Security.