Ghobaei-Arani, M, Souri, A, Baker, T and Hussien, A

ControCity: An Autonomous Approach for Controlling Elasticity Using Buffer Management in Cloud Computing Environment

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

ControCity: An Autonomous Approach for Controlling Elasticity Using Buffer Management in Cloud Computing Environment

Mostafa Ghobaei-Arani[1,2], Alireza Souri[3], Thar Baker[4*] and Aseel Hussien[5]

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

[2]Young Researchers and Elite Club, Qom Branch, Islamic Azad University, Qom, Iran

E-mail: m.ghobaei@qom-iau.ac.ir

[3]Young Researchers and Elite Club, Islamshahr Branch, Islamic Azad University, Islamshahr, Iran

E-mail: a.souri@srbiau.ac.ir

[4]Department of Computer Science, Liverpool John Moores University, Liverpool, UK

*Corresponding Author E-mail: t.baker@ljmu.ac.uk

[5]Department of the Built Environment, Liverpool John Moores University, Liverpool, UK

E-mail: a.hussien@ljmu.ac.uk

ABSTRACT Cloud computing has been one of the most popular distributed computing paradigms. Elasticity is a crucial feature that distinguishes cloud computing from other distributed computing models. It considers the resource provisioning and allocation processes can be implemented automatically and dynamically. Elasticity feature allows cloud platforms to handle different loads efficiently without disrupting the normal behavior of the application. Therefore, providing a resource elasticity analytical model can play a significant role in cloud resource management. This paper presents Controlling Elasticity (ControCity) framework for controlling resources elasticity through using "*buffer management*" and "*elasticity management*". In the proposed framework, there are two essential components called buffer manager and elasticity manager in the application layer and middleware layer, respectively. The buffer management controls the input queue of the user's request and the elasticity management controls the elasticity of the cloud platform using learning automata technique. In the application layer, applications are received by cloud applications and, then, placed in the control of the buffer. Buffer manager controls the queue of requests, and elasticity manager of the middleware layer using the learning automata provides a solution for controlling the elasticity of the cloud platform. The experimental results indicate that ControCity reduces the response time by up to 3.7%, and increases the resource utilization and elasticity by up to 8.4% and 5.4%, respectively, compared with the other approaches.

INDEX TERMS Cloud Computing, Elasticity, Buffer Management, Learning Automata

I. INTRODUCTION

In recent years, the computing trend moved toward the cloud computing paradigm, particularly when large computing resources are required to serve a cloud application, using the ideas of computing power as a utility to deliver a unified service to the end-users [1, 2]. In cloud computing, the IT infrastructures such as storage, servers, and network can be dynamically provisioned according to the user requirements using on-demand self-service delivery model [3, 4]. One of the considerable properties that differentiate cloud computing from other computing paradigm is elasticity [5]. Elasticity property allows the cloud platforms to efficiently add or remove the cloud infrastructures (e.g., VMs) automatically according to the number of users for supporting the rapid fluctuation of loads to serve better.

A. Research motivation and challenges

Since the end-users may have irregular access to cloud applications over time, it is difficult to handle load fluctuations with the traditional infrastructure [6, 7]. Load fluctuations are the points where the workload of the system changes continuously. This is one of the important issues that should be considered for managing cloud infrastructure as the backbone of the cloud platform. If load fluctuations of cloud applications using elasticity property is not correctly managed, the whole cloud platform can fail and Quality of Service (QoS) would be adversely affected and it may face to the over-provisioning or the under-provisioning issues [8-10]. In the over-provisioning issue, the cloud infrastructures allocated are greater than the user needs, and this leads to useless cost to lease the cloud infrastructures while QoS requirements can be satisfied. In the under-provisioning issue, the allocated cloud infrastructures are smaller than the user needs, and this causes violation of Service

Level Objectives (SLOs) agreed between end-users and cloud platform. Therefore, managing the cloud infrastructure to guarantee elasticity property of cloud platform can play an important role in cloud resource management to deal with the under or over-provisioning issues.

B. Our approach

In this paper, we designed an autonomous framework for controlling elasticity in a cloud platform that includes two major components named buffer manager and the elasticity manager. The buffer manager component is responsible for controlling the input queue of the requests, and it follows a reference autonomous computing model proposed by IBM [11-13], which is called the MAPE (monitoring-analysis-planning-execution) control loop. The monitoring phase observes the number of incoming requests and the buffer remaining space as inputs analysis phase to predict the number of future requests. The planning phase using learning automata [14] determines the size of buffer memory to be low or high, or kept in the same state [15]. Besides, the elasticity manager component is responsible for controlling the elasticity of the cloud platform using the learning automata technique based on QoS analysis.

C. Contributions

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

- Designing an autonomous framework for managing of elasticity feature in a cloud platform.
- Utilizing a learning automata technique as a decision-maker into the elasticity manager component of the proposed framework to control the elasticity of the cloud platform.
- Evaluating the performance of the proposed solution under three real workloads by performing a series of experiments for improving elasticity and resource utilization.

D. Organization of the paper

The rest of this paper is organized as follows: In Section 2, we focus on a literature review of related works. Section 3 describes the proposed solution in more details. Section 4 presents an evaluation and discuss the experimental results. In Section 5, we conclude the paper and present future works.

II. RELATED WORKS

In this section, we review research studies about the elasticity management mechanisms in cloud environments.

Ullah et al. [16] have studied the cloud elasticity property by focusing on control theoretical mechanisms and provide a comprehensive taxonomy from the point of view of control theory as an implementation mechanism. Besides, they investigate some research challenges such as heterogeneity, interoperability, computational overhead analysis, uncertainty, scalability, oscillation, and resource usage analysis that needs to be further addressed. Albonico et al. [17] have proposed a mechanism that manages the elasticity feature of web applications according to their QoS requirements. Their mechanism controls automatically the workload generation to manage web applications using elasticity states including scaling out, ready, and scaling in states. Finally, they evaluate their solution on Amazon EC2 and indicated that their solution can manage web applications in minimal time.

Salah et al. [18] design an analytical model using Markova chains to ensure proper elasticity for cloud-hosted applications and services. Their model utilizes the offered workload and the number of VM instances as an input to estimate the minimal number of VMs required to satisfy a given Service-Level Objective (SLO) criterion. Besides, their proposed model can estimate the number of load balancers needed to achieve proper elasticity. They evaluated their proposed model using practical scenarios of cloud elastic services that include web service, Netflix video streaming, and the Amazon Web Services (AWS) cloud platform. Their numerical results indicated the effectiveness of their proposed analytical model outperforms in capacity engineering and estimation of the cloud computation and network resources for different real-world scenarios compared with other algorithms.

Zhang et al. [19] have designed a lightweight container-based framework named auto-scaler for controlling the elasticity feature to deal with load fluctuations in the small devices. Their proposed framework consists of four components namely, the monitoring mechanism, history recorder, decision mechanism, and execution mechanism. Further, they describe the elasticity feature mathematically for quantifying cloud elasticity using container-based auto-scaling mechanisms. They validate their framework on the Mesosphere Data Center Operating System cloud infrastructure using the stress workload and illustrated that their framework can manage the tradeoff between stability and elasticity. Nouri et al. [20] have presented an autonomic decentralized elasticity controller for managing resources on web applications in cloud environments. Their proposed controller utilized a reinforcement learning-based technique to handle workload arrival patterns using a set of states and actions. Their simulation results under real-world workloads demonstrated that their proposed controller reduces SLA violations percentage and cost of provisioning cloud infrastructure.

In [21] an elasticity control algorithm for containerized-based cloud infrastructure for augmenting the load balancing is introduced. Their proposed algorithm utilized two agents, namely the master agent for coordinating between hosts and the host agents for monitoring and predicting resource utilization using Autoregressive Moving Average (ARMA) prediction model. Their numerical results indicated that their proposed algorithm outperforms in terms of elasticity and power consumption compared with other algorithms. Jrad et al. [22] have introduced a framework for evaluating elasticity mechanisms for service-based business processes in cloud environments. Their proposed framework includes a set of

domain-specific languages to facilitate the description and the evaluation of elasticity mechanisms. Al-Dhuraibi et al. [23] have proposed a model-driven elasticity management system according to the Open Cloud Computing Interface (OCCI) standard. Their solution considers both VM and container virtualization technologies, both vertical and horizontal scaling, and multiple cloud providers, simultaneously. Also, their proposed elasticity system handles the heterogeneity of elasticity mechanisms on three popular cloud providers namely, Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform.

Finally, Jamshidi et al. [24] propose a dynamic self-adaptive method based on the fuzzy controller for organized elasticity management in cloud computing. The proposed method has applied to Azure and OpenStack based on self-learning fuzzy controller that confirms and improves fuzzy rules at real-time execution.

Overall, since elasticity property is trying to adapt the load changes to available resources at any time by allocating /reallocating/deallocating resources in an autonomous manner. First, the current studies focused on controlling elasticity property on behalf of resources and infrastructure and did not pay attention to controlling the elasticity from the perspective user requests using a buffering technique. Secondly, most of the previous studies have only utilized the prediction models or machine learning techniques for controlling elasticity, while our proposed solution combines machine learning techniques (i.e., learning automata) and prediction models (i.e., Moving Average prediction model) to ensure elasticity feature.

III. PROPOSED CONTROCITY APPROACH

In this section, we explain our ControCity framework in more details, as shown in Figure 1. The proposed framework consists of three main layers: the application layer, the middleware layer, and the infrastructure layer. In this framework, there are two important components called buffer manager in the application layer and the elasticity manager in the middleware layer. In the application layer, applications are received by cloud applications and then placed in the control of the buffer. Buffer manager controls the queue of requests, and the elasticity manager of the middleware layer using the learning automata provides a solution for controlling the elasticity of the cloud platform.

In the proposed approach, elasticity management has a part called QoS mapper unit, which is responsible for receiving cloud information including the amount of Machine Instruction Per Second (MIPS) allocated and the MIPS consumed. The next step of the workload and the degree of service provision violation occurred and the learning automata will be used to provide the right amount of resources. The infrastructure layer is responsible for underlying cloud resource management. In this layer, each VM is placed in one of these layers. The VM may migrate from one layer to another by reconfiguration [2]. If the input requests to each layer exceed the service limit, the load regulator on each layer creates a queue request, by which, after being released, each virtual machine is offered a request.

In the following, we describe the main components of the ControCity framework in more details. Also, the existing applied notations of the proposed solution are shown in Table 1.

A. Buffer manager

Figure 2 shows the overall structure of the buffer in the proposed method. Specifically, in the proposed solution, the buffer is composed of two parts of an administrator and decision-maker and a part of management. The decision-maker part is responsible for increasing or decreasing buffer memory based on the input request traffic, and the management part is responsible for sorting and scheduling the input requests to send. The amount of buffer remaining space and the input request traffic are effective in decision making of the monitoring and decision unit. Specifically, the decisions of the monitoring and decision unit are applied to buffer memory by management.

Table 1. Notations of the proposed solution

Notation	Description
Pri_i	Request priority *i-th*
TTL	Time To Live of request
q_{min}	Low Threshold for number of requests
$Drop$	Removed request
$Send\_Req$	Ready request to send
$SLA\_Violation_{Response}$	Response Time Violation Rate
$SLA\_Violation_{Cost}$	The cost of the violation
$Cost_{Res}$	Final cost of answering requests
$Cost_{Req}$	Initial request cost
L_Thr	Low Threshold for queue capacity
H_Thr	High Threshold for queue capacity
$RequestedBytes\_Queue_i$	The data volume of the input request in the *i-th* queue
$AvailableBytes\_Queue_i$	The total capacity of the *i-th* queue
$WL_{Current}$	Current workload
$WL_{Predict}$	Predicted workload
R	Response time of the request
R_{max}	Maximum Response Time of Request
R_{min}	Minimum response time of request
C	Request cost
C_{max}	The maximum cost of request
C_{min}	The minimum cost of request
$Scaling\_Type$	Scaling type
$Scaling\_Amount$	Scaling rate

According to the structure in Figure 3, each request contains the user ID, the cost, and the deadline for the response time. If the cost and response time exceed this limit, the violation of the terms of service has occurred. In the proposed method, the buffer management comprises three queues; in which each queue specifies the priority of the requests. In order to specify the priority of requests, Eq. (1) is used.

$$Pri_i = \alpha \times \frac{1}{Current\_time_i} + (1-\alpha) \times \frac{1}{DeadLineTime_i} \quad (1)$$

In Eq. (1), the $Current\_Time_i$, is the arrival time of the request and the $DeadlineTime_i$ is the time limit for the response. α is the amount of weight for each of the parameters. Therefore, the

input requests are placed in one of the queues based on the request time and the priority. The three types of priorities of top, normal and low are considered.

Figure 4 shows the structure of the input request segmentation. If the length of each queue is considered in buffer B, the buffer length is equal to *B** 3 and generally, if the queues are B1, B2, and B3 respectively, then, the following relationship can be used about the buffer memory.

$$B = \{B_1 \cup B_2 \cup B_3\} \quad (2)$$

According to Figure 4, the input request is initially prioritized by the classifier and then it is placed in its queue. Due to the attempts made in the previous step, the size of buffer memory varies with traffic, but the overflow may happen due to any reason; so, there is no more space in buffer memory to add a new request; in this case, the request must be deleted from the buffer.

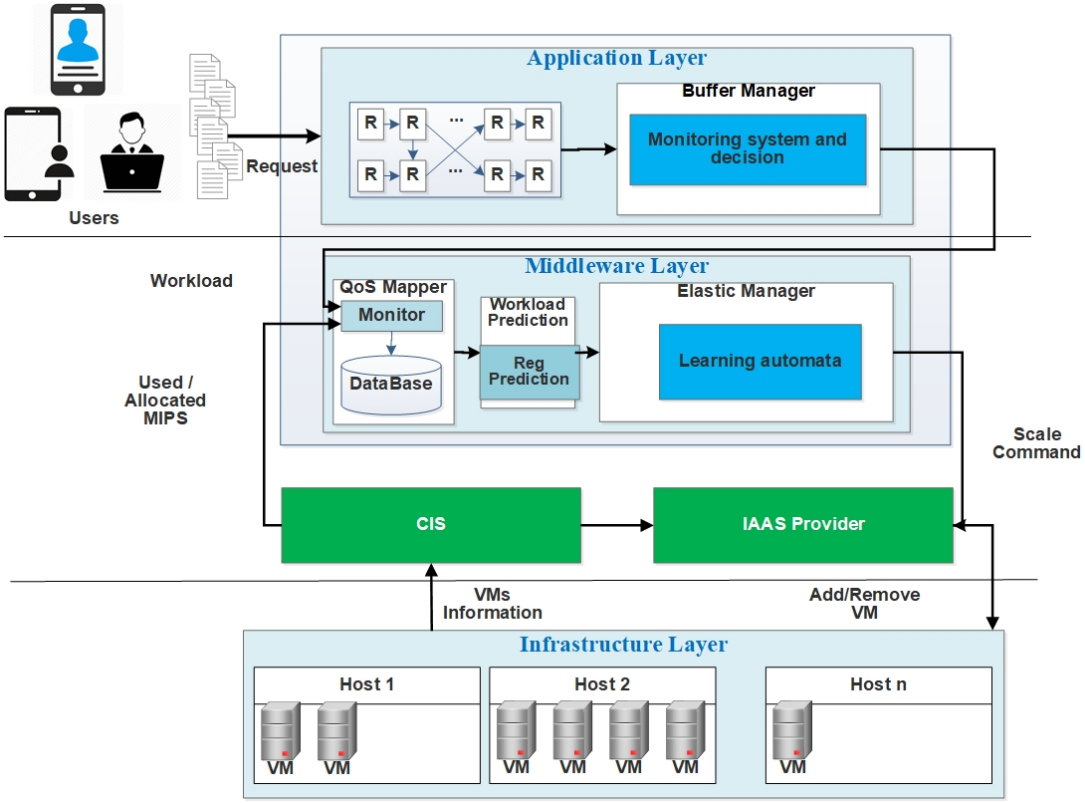


Fig 1. A high-level overview of ControCity framework

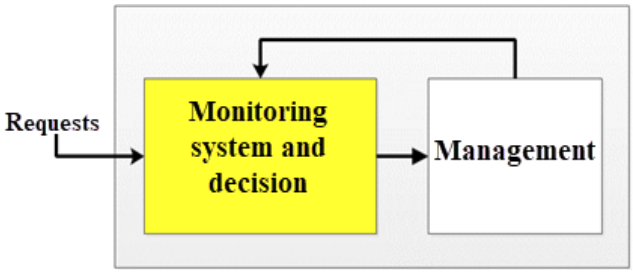


Fig .2. The overall buffer structure in the proposed method

In the proposed solution, the user's request is made under SLA rules. Each SLA consists of several objectives or SLOs. In this research, the user's request structure includes two objectives of cost and response time. The user can simultaneously submit multiple requests to the cloud provider. Figure 3 shows the structure of each user's request.

User id	Time	User SLA	
		SLO1	SLO2
		Cost	Deadline Time

Fig. 3. Structure of each user request

The deletion of requests from each queue is made separately. The key point in eliminating the request from the queue is that a request must be removed from a queue that has the maximum amount of survival time. This means that the oldest request with the longest queuing time in the queue will be excluded; but the most important thing is that low- priority queries must be

removed first, and then in the absence of space, the medium-priority queries, and at last in the absence of space, high priority queries will be removed. So, if the buffer memory space is full and a new request arrives, one of these three modes will occur:

A) First mode (receiving a high- priority request)
In this mode, equation (4) is performed. First, if the number of requests in the low- priority queue is more than the considered threshold, it will be removed. Otherwise, if the number of requests in the average-priority queue is more than the considered threshold, it will be removed and, otherwise, the queue will be executed from the high- priority queue.

$$Drop = \begin{cases} \text{Req From } Q_3 \text{ With Least TTL} & \text{if } Count(Q_3) > q_{min} \\ \text{Req From } Q_2 \text{ With Least TTL} & \text{if } Count(Q_2) > q_{min} \\ \text{Req From } Q_1 \text{ With Least TTL} & \text{Otherwise} \end{cases} \quad (4)$$

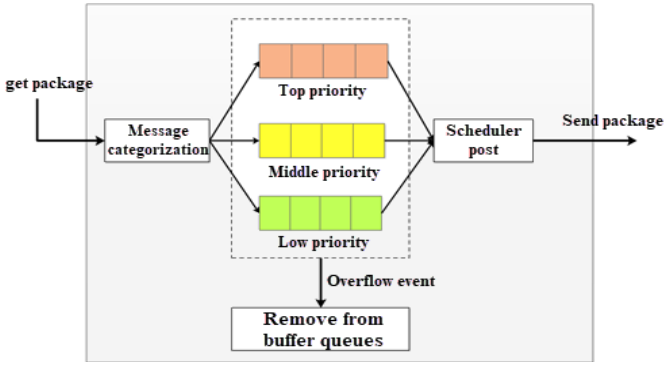


Fig .4. Buffer Management structure.

B) Second mode (receiving an average priority request)
In this mode, equation (5) is performed. First, if the number of requests in the low- priority queue is more than the considered threshold, it will be removed. Otherwise, the elimination is done from the average-priority queue.

$$Drop = \begin{cases} Req\ From\ Q_3\ With\ Least\ TTL & if\ \ Count(Q_3) > q_{min} \\ Req\ From\ Q_2\ With\ Least\ TTL & Otherwise \end{cases} \quad (5)$$

C) Third mode (receiving a low priority request)
In this mode, equation (6) is applied, in which the request is only deleted from the lower priority queue.

$$Drop = \text{Req From } Q_3 \text{ With Least TTL} \quad (6)$$

When there are multiple requests in queues, and each one is ready to be sent, first, the buffer scheduler sends the top queue requests and after completing, it will go to the average and low priority queue, respectively. Equation (7) shows the structure of sending requests according to their priority.

$$Send\_Req = \begin{cases} \text{Req} \in Q_3 & \text{if } Q_3 \text{ is not empty} \\ \text{Req} \in Q_2 & \text{if } Q_2 \text{ is not empty} \\ \text{Req} \in Q_1 & \text{Otherwise} \end{cases} \quad (7)$$

The requests are submitted to the Elasticity Management Unit, respectively, so that the unit decides on resource allocation based on the information received from the service quality mapping unit.

B. Monitoring and Decision System

The monitoring and decision system in the proposed approach acts on the MAPE structure. Due to the fact that this structure consists of 4 phases of monitoring, analysis, planning, and execution, all phases have a specific function in the proposed method. The proposed method is based on the loop and acts on the buffer decision-maker. Figure 5 shows the MAPE structure. According to Figure 5, the monitoring phase observes the number of incoming requests and the buffer remaining space as inputs analysis phase. Data collected by the monitoring system is provided to the analysis phase. The analysis phase is responsible for predicting the future status of input requests based on the current status. According to the prediction of the analysis phase and the amount of buffer space remaining, the planning phase using learning automata determines the size of buffer memory to be low or high, or kept in the same state. The decisions taken by the planning phase is sent to the execution phase to make this decision operational.

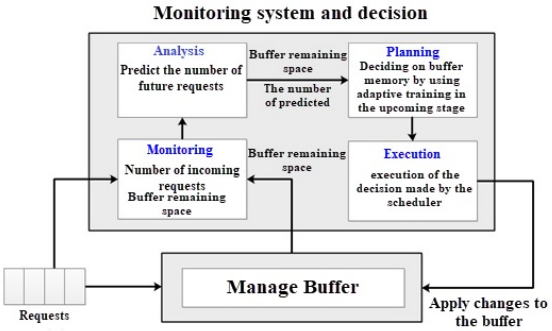


Fig .5. Monitoring decision system in buffer management.

The monitoring phase of the MAPE structure monitors the received traffic requests and the amount of buffer space available. The monitoring phase has a knowledge base. It checks the buffer status at certain intervals and extracts its required variables and records them in its knowledge base. The knowledge base consists of two parts: the storage sections of input traffic and remaining buffer space. The structure of the knowledge base records part I is shown in Table 2 and the structure of the knowledge base records part II is shown in Table 3.

Table 2. Structure of the Knowledge Base Records Part I

Request ID	Required layer data	data

Table 3. Structure of the Knowledge Base Records Part II

The amount of Residual buffer amount from queue 1	The amount of Residual buffer from queue 2	The amount of Residual buffer from queue 3

C. QoS Mapper Unit

This unit is responsible for saving responsive information and service delivery to the request. After recording the data, the unit calculates the amount of violation of service terms and informs the elasticity management about it. The data storage structure of this unit is presented in Table 4.

Table 4. Structure of Service Quality Information for a Request

The user ID	The request ID	Response Time Layer I	Cost Layer I	Response Time Layer II	Cost Layer II	Response Time Layer III	Cost Layer III

The amount of violations of service conditions is calculated by the knowledge of service quality. If the response time exceeds the allowed response time to the request, the Equation (8) shows how to calculate the amount of service violation for the response time. Similarly, if the cost of responding to a request exceeds the intended cost, Equation (9) shows how to calculate the amount of service violation for the cost.

$$SLA\_Violation_{\mathrm{Re}sponse} = \mathrm{Re}sponse_{Time} - Deadline_{time}$$
$$SLAV\_Count_{\mathrm{Re}sponse} = Count\left(SLA\_Violation_{\mathrm{Re}sponse} > 0\right) \quad (8)$$

$$SLA\_Violation_{Cost} = Cost_{Res} - Cost_{Req}$$
$$SLA\_Count_{Cost} = Count\,(SLA\_Violation_{Cost} > 0) \quad (9)$$

In Equation (8), *Response Time* is the response time to the request, and *Deadline Time* is the allowed response time for a request. In Equation (9), $Cost_{Req}$ is the initial cost considered for the request and $Cost_{Res}$ is the final cost of responding to the request.

D. Workload Prediction

One of the most important parts of the MAPE structure is the analysis of results. Future requests will be considered in the traffic prediction analysis phase. The more accurate the prediction is in this part; the planning design phase will have a higher quality. In the proposed method, the average mobility prediction structure of Moving Average (MA) has been used. In this model, the prediction is obtained based on the average number of requests in the preceding steps, and the number of requests in the following steps. Equation (10) displays how to calculate the number of requests in the following step by using MA,

$$W_{t+1} = \mu + \in_t + \theta_1 \in_{t-1} + \theta_2 \in_{t-2} + .. + \theta_q \in_{t-q}$$
$$\mu = \frac{1}{n}\sum_{i=1}^{n} W_{t-i} \quad (10)$$

where μ is the average number of requests in the previous steps. $\theta_1 \ldots \theta_q$ are the parameters of the MA prediction model that are determined based on the number of requests. $\epsilon_t \ldots \epsilon_{t-q}$ are random values with the normal distribution and zero mean. Generally, these values are referred to as white noise. In the planning phase, each queue of the buffer consists of three modes of busy, idle and normal. In order to determine the different states of a queue, the upper and lower thresholds need to be defined. The method is that if the buffer is not in normal mode, it should be modified. If each buffer queue is placed in a state for two successive times and the predicted workload is proportional to the buffer queue state, the changes in the buffer queues will be adapted and strengthened. The amount of buffer overflow (Eq. (11)) is obtained by dividing the two variables of Available Bytes (the available residual space) and Requested Bytes (bytes required for current requests). These are two input variables. Comparing the amount of overflow determines the upper and lower thresholds. Buffer increasing, decreasing or being idle is done based on these actions.

$$Buffer\_Queue_j^i = \frac{RequestedBytes\_Queue_j^i}{AvailableBytes\_Queue_i} \quad (11)$$

$RequestedBytes\_Queue_j^i$ determines the data volume of the input request in the *i-th* queue at time *t* and $AvailableBytes\_Queue_i$ determines the total capacity of the *i-th* queue. In the following, three modes of busy, idle and normal are introduced. Suppose that the amount of input traffic to the provider is greater than the queue length in the buffer. In this case, the provider confronts a lack of memory to hold input requests. Then, the queue is fully engaged, and the algorithm increases the buffer length queue. Equation (12) shows the busy mode of the *i-th* queue in the buffer.

$$Buffer\_Queue_t^i \geq H\_Thr$$
$$Buffer\_State\_Queue_t^i = UP \quad (12)$$

The queue memory is generally idle when the amount of input traffic to the queue is lower than the least capacity considered for the queue in the buffer. Then the buffer queue will be in the idle mode and memory space reduction must be done. Equation (13) shows the idle mode of *i-th* queue in the buffer.

$$\begin{aligned} &Buffer\_Quete_t^i \le L\_Thr \\ &Buffer\_State\_Quete_t^i = Down \end{aligned} \quad (13)$$

When the buffer queue is sufficient for input traffic, the buffer is in a normal state and it does not do anything. Equation (14) shows the normal mode in the queue.

$$\begin{aligned} &L\_Thr \le Buffer\_Queue_t^i \le H\_Thr \\ &Buffer\_State\_Quete_t^i = Normal \end{aligned} \quad (14)$$

Let $Buffer\_State\_Quete_t^i$ denote the buffer state of the *i-th* queue at time *t* and L_Thr, H_Thr be the lower threshold and the higher threshold to determine the buffer state according to the capacity of the *i-th* queue, respectively, where $H\_Thr > L\_Thr$. Generally, the buffer state is expressed by Equation (15):

$$Buffer\_State\_Quete_t^i = \begin{cases} UP & if \quad Buffer\_Queue_t^i > H\_Thr \\ Normarl & if \quad L\_Thr \le Buffer\_Queue_t^i \le H\_Thr \\ DOWN & if \quad Buffer\_Quete_t^i < L\_Thr \end{cases} \quad (15)$$

L_Thr, H_Thr depends on the amount of input traffic to the queue (i.e., workload changes) and determined according to the queue capacity. Let $D = H\_Thr - L\_Thr$ denote the difference between the lower threshold and the higher threshold for the capacity of the *i-th* queue. The higher value D leads to waste of the provided queue capacity (i.e., under-load state) and extra and unnecessary cost while QoS requirements can be satisfied. Whereas, the lower value D results in performance degradation due to inadequate the provided queue capacity (i.e., over-load state) for serving user requests. Therefore, we will need to adjust an appropriate value D to determine the buffer state, accurately.

According to the *i-th* queue state at time *t* ($Buffer\_Queue_t^i$), for comparative training, the values of *U* and *P* are calculated as adding and decreasing queues.

$$A = \begin{cases} 1 & \text{if Buffer\_State\_Queue}_t^i = Buffer\_State - Queue_{t-1}^i \\ 0 & \text{Otherwise} \end{cases} \quad (16)$$

$$B = \begin{cases} 1 & \text{if Buffer\_State\_Queue}_t^i = UP \\ 0 & \text{if Buffer\_State\_Queue}_t^i = \text{DOWN} \end{cases} \quad (17)$$

Then, based on *A* and *B*, the final calculation is done.

$$\begin{cases} U_{t-1} + \left| RequestedBytes\_Queue_j^i - Predicted_i \right| & if\ A = 1\ and\ B = 1\ and\ U_{t-1} > 0 \\ \left| RequestedBytes\_Queue_j^i - Predicted_i \right| & if\ A = 0\ and\ B = 1\ and\ U_{t-1} = 0 \\ 0 & Otherwise \end{cases} \quad (18)$$

$$L_t = \begin{cases} L_{t-1} + \left| RequestedBytes\_Queue_j^i - Predicted_i \right| & if\ A = 1\ and\ B = 0\ and\ L_{t-1} > 0 \\ \left| RequestedBytes\_Queue_j^i - Predicted_i \right| & if\ A = 0\ and\ B = 0\ and\ L_{t-1} = 0 \\ 0 & Otherwise \end{cases} \quad (19)$$

According to the values, U_t is considered as increasing the queue and L_t is considered as decreasing the queue. Increasing and decreasing the length of each queue are performed, respectively, by Equations (19) and (20).

$$AvailableBytes\_Queue_i = \begin{cases} U_t + AvailableBytes\_Queue_i & if \quad B = 1 \\ 0 & Otherwise \end{cases} \quad (20)$$

$$AvailableBytes\_Queue_i = \begin{cases} AvailableBytes - Queue_i - Lt & if \quad B = 0 \\ 0 & Otherwise \end{cases} \quad (21)$$

After calculating the amount of increasing or decreasing, the value and its decision are announced to the implementation unit. After decision making in the planning phase, the decision is to implement in the implementation section. According to the type and amount of change, the implementation section announces the changes to the buffer manager to adjust the structure based on the decision-maker. According to the type of a decision, the volume of the buffer increases or decreases in one of the queues 1, 2, or 3.

E. Elasticity Manager

Specifically, no decision will be made for the first workload, because the output of the service quality mapping unit does not exist; but in the later stages, based on workload prediction, scaling is performed at the buffer management stage. So, according to the need for each layer request, the scaling is done

proportionally for that layer. For scaling each layer, the decision-making structure is based on the current and the predicted workload of each layer. Equation (22) specifies how to calculate the type and the extent of scaling.

$$Scaling = \begin{cases} UP & \mathrm{WL}_{Current} < WL_{predict} \\ NO-UP & \mathrm{WL}_{Current} = WL_{predict} \\ DOWN & \mathrm{WL}_{Current} > WL_{predict} \end{cases} \qquad (22)$$

$$Scaling\_Amount = ABS(\mathrm{WL}_{Current} - WL_{predict})$$

In the proposed structure, a learning automaton is placed on each layer. If the structure of each layer is considered as an automaton performance environment, this environment can be represented by the triplet $E \equiv \{\alpha, \beta, c\}$, in which the set of environmental inputs or actions selected by the automaton $\alpha = \{ScaleDown, ScaleUp, NoOp\}$, and the outputs of β and C are likely to be penalized. The key point about the environment inputs is that these actions have been taken by the previous decision-maker and will be reinforced or weakened by the automaton. In this environment, $\beta = 1$ is considered as an undesirable response or failure, and $\beta = 0$ is considered as a desirable or successful response. C specifies the probabilities of the penalty (i.e., failure) for the environment responses and is defined as follows.

$$c = \Pr ob\{\beta = 1\} \qquad (23)$$

Figure 6 shows the structure of the automata connection with each application layer. According to the mentioned parameters, by choosing any action by automata, the probability of doing that will change. The equations (24) and (25) are used for the reward or penalty of the selected operation, where a is a reward coefficient and b is a penalty coefficient. $P_i(n)$ is the probability of the occurrence of the action i in the step n, and specifically $P_i(n+1)$ is the probability of the occurrence of the future event.

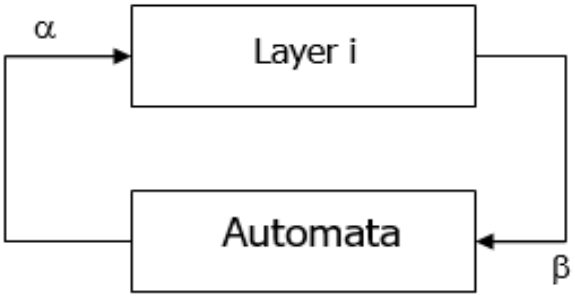


Fig .6. Automaton connection with the layer

$$\begin{aligned} P_i(n+1) &= P_i(n) + \alpha[1 - P_i(n)] \\ P_j(n+1) &= 1 - P_i(n+1) \end{aligned} \qquad (24)$$

$$\begin{aligned} P_i(n+1) &= (1-b)P_i(n) \\ P_j(n+1) &= 1 - P_i(n-1) \end{aligned} \qquad (25)$$

One of the most important points is how to fine and encourage selective action. In each cycle, after action selection, if the *Scale Up*, *Scale Down or No Op* action is selected, respectively the equations (26), (27) or (28) of the β - reinforcement signal will be calculated. If $\beta = 1$, then, the selected operation is fined by equation (26); otherwise, if $\beta = 0$, then it is rewarded according to equation (27). If R_Ci is the mean of the response time of the *i-th* layer and C_Ci is the average cost of the *i-th* layer, then it is rewarded according to equation (28).

$$\beta = \begin{cases} 1 & \mathrm{R\_C}_i < \dfrac{C_i R\max + C_i R\min}{2} \text{ and } \mathrm{C\_C}_i < \dfrac{C_i C\max + C_i C\min}{2} \\ 0 & \mathrm{R\text{-}C}_i \geq C_i R\max \text{ or } \mathrm{C\text{-}C}_i \geq C_i C\max \end{cases} \qquad (26)$$

$$\beta = \begin{cases} 1 & \mathrm{R\_C}_i \geq \dfrac{C_i R\max + C_i R\min}{2} \text{ and } \mathrm{C\_C}_i \geq \dfrac{C_i C\max + C_i C\min}{2} \\ 0 & \mathrm{R\_C}_i < \dfrac{C_i R\max + C_i R\min}{2} \text{ and } \mathrm{C\_C}_i < \dfrac{C_i C\max + C_i C\min}{2} \end{cases} \qquad (27)$$

$$\beta = \begin{cases} 1 & \mathrm{R\text{-}C}_i > C_i R\max \text{ or } \mathrm{C\text{-}C}_i > C_i C\max \\ 0 & Otherwise \end{cases} \qquad (28)$$

The dependent structure is used to determine the amount of penalty and reward. Equation (29) specifies how to calculate the reward coefficient.

$$a = \left(\frac{1}{\left|R\_C_i - C_i R\max\right| + \left|C\_C_i - C_i C\max\right|} \right) \qquad (29)$$

Equation (30) specifies how to calculate the penalty coefficient.

$$b = 1 - \left(\frac{1}{\left|\mathrm{R\_C}_i - C_i R\max\right| + \left|\mathrm{C\_C}_i - C_i C\max\right|} \right) \qquad (30)$$

IV. PERFORMANCE EVALUATION

In this section, we evaluate the effectiveness of the proposed approach for improving the elasticity feature in the cloud environment. We first explain the experimental setup and

performance metrics, and, then, the experimental results are discussed.

A. Experimental setup

In this section, we explain the simulation setup in more details. The CloudSim toolkit [25] as a simulation framework is utilized for modeling and developing the cloud computing infrastructures. Besides, there are five physical hosts at each cloud data center, so they all have the existing specification according to Tables 5 and 6.

Table 5. Data centers specification

Architecture	Operating System	Virtual Machine Manager
X64	Cloud Linux	XEN

Table 6. Host specification

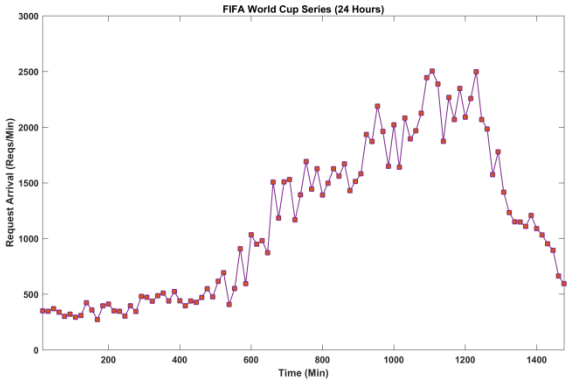
Name	Processor Type	Number of cores	Frequency (MIPS)	main memory (GB)	Bandwidth
Host1	Intel Xeon 5370	16	4096	16	1 Gbit/s

Also, we can consider three types of VMs that are offered by any cloud provider: small, medium, and extra-large. The configuration details of different types of VMs into the three categories with the different capabilities are shown in Table 7.

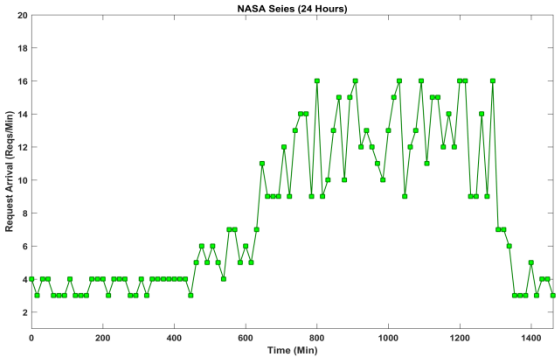
Table 7. Virtual machines specification

Machin Name	CPU(MIPS)	RAM (GB)	Storage (GB)	BW (Gbps)	Price ($ per Hour)
t2.small	10200	2	1 GB - 16 TB	100Mbps	0.023
m3.Medium	12000	3.75	1×4 GB	1Gbps	0.070
m4.4Xlarge	15000	64	1 GB - 16 TB	1Gbps	0.862
r3.4Xlarge	80000	122	1×320 GB	10Gbps	1.330
m4.10Xlarge	97000	160	1 GB - 16 TB	10Gbps	2.155
d2.4Xlarge	105000	122	12×2000 GB(24 TB)	10Gbps	2.76
m4.16Xlarge	280000	256	1 GB - 16 TB	100Gbps	3.447
r4.16Xlarge	350000	488	1 GB - 16 TB	100Gbps	4.256
d2.8Xlarge	500000	244	24×2000 GB(48 TB)	100Gbps	5.52

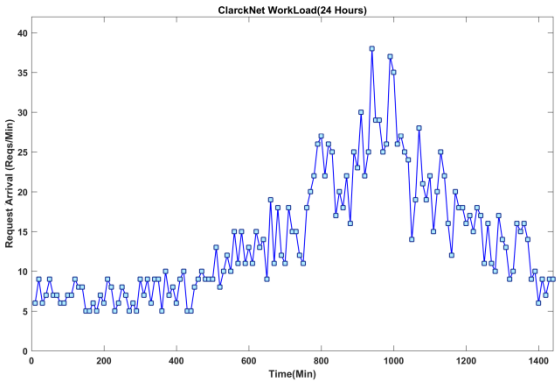
To evaluate our approach, we used three types of real workloads, including three data sets of FIFA World Cup, ClarkNet and NASA. NASA data set includes 2 months of workload and 3461612 requests. FIFA World Cup data set includes 88 days of workload and 1352804107 requests. ClarkNet data set includes two weeks of workload and 338587 requests. These workload traces extracted from well-known websites and indicates realistic load variations which make the results more realistic and reliable to be used in a real cloud platform. In this paper, the time intervals are considered in 15-min intervals. Thus, each day includes 96-time intervals. These three real workloads traces are shown in Figure 7.



(a)



(b)



(c)

Fig 7. Workload patterns: (a) FIFA world cup (b) NASA (c) ClarkNet

B. Performance Metrics

We applied the following metrics for a comparison of our approach with other strategies:

Elasticity: This metric is defined as the degree to which a cloud computing platform adapted upon the fluctuation of workloads and can be measured by the percentage of time when the cloud platform is in normal-provisioning states and is calculated by

Equation (31) [26]:

$$E = \frac{T_n}{T} = 1 - \frac{T_o + T_u}{T} \qquad (31)$$

where T denotes the total time that a system is operating for a sufficiently long time period, T_o be the total time period that the system is in the over-provisioning state, T_u be the total time period that the system is in the under-provisioning state, and T_n be the total time period that the system is in the normal-provisioning state. Therefore, T includes all the time periods in the normal, over-provisioning, and under-provisioning states; that is, $T = T_o + T_u + T_n$

Utilization: The CPU utilization of the cloud platform is defined as the ratio of the average amount of the *allocated* Machine Instruction Per Second *(MIPS)* of VMs for serving user requests to the average amount of the *total MIPS* that is potentially offered by VMs into the cloud platform, and is expressed by Equation (32):

$$U = \frac{Allocated\ MIPS\ (VMs)}{Total\ MIPS\ (VMs)} \qquad (32)$$

Response time: The actual response time is the time difference between the user request start time and the first response time received from the user by the cloud platform.

C. Experimental Results

To evaluate the performance of the proposed approach, we design three scenarios based on three real workload traces and performance metrics that were discussed in the previous subsections, as shown in Table 8. We compare our approach with two baseline approaches. The first on is called Automatic Elasticity- Fuzzy Based System "AE_FBS" algorithm [24, 27], which is a fuzzy rule-based controller linked with a reinforcement learning algorithm that learns and modifies elasticity policies at runtime for auto-configuration of VMs in a cloud environment. The second approach is called "CTMC" [18], which uses an analytical model based on Continuous-Time Markov Chain (CTMC) to estimate the number of virtual machines needed to adjust the resource elasticity value of a cloud platform. Finally, our ControCity approach is called Automatic Elastic-Buffer Management Automata Learning "AE_BMLA" algorithm. The reason for choosing these approaches for comparison with ours is (i) these approaches are proactive, i.e., try to predict the number of resources at any given time to deal with workload fluctuations, and, (ii) these approaches follow horizontal scaling (i.e., replication) for adding/removing VM instances from a cloud platform to provide elasticity.

Table 8. Different scenarios for evaluating the proposed approach

Scenario	Workloads type	Objective
Scenario 1	FIFA, NASA, ClarckNet	Studying the simulation results AE_FBS , CTMC, and AE_BMLA approaches in terms of the elasticity
Scenario 2	FIFA, NASA, ClarckNet	Studying the simulation results AE_FBS , CTMC, and AE_BMLA approaches in terms of the CPU utilization
Scenario 3	FIFA, NASA, ClarckNet	Studying the simulation results AE_FBS , CTMC, and AE_BMLA approaches in terms of the response time

C.1 First scenario: Impact of elasticity metric

In this scenario, the elasticity of the AE_BMLA algorithm is examined against CTMC and AE_FBS algorithms. One of the important parameters in comparing the performance of the resource allocation algorithm is elasticity. To conduct this experiment, ten different time intervals were selected from each workload in a way that the number of requests from the first to the tenth interval is ascending. Figure 8 displays the elasticity of the AE_BMLA algorithm in comparison with the CTMC and AE_FBS algorithms in the FIFA workload. According to Figure 8, due to the queuing and use of the controller structure to select the appropriate host and accurate increasing and decreasing the virtual machine, the proposed method shows a better performance about elasticity.

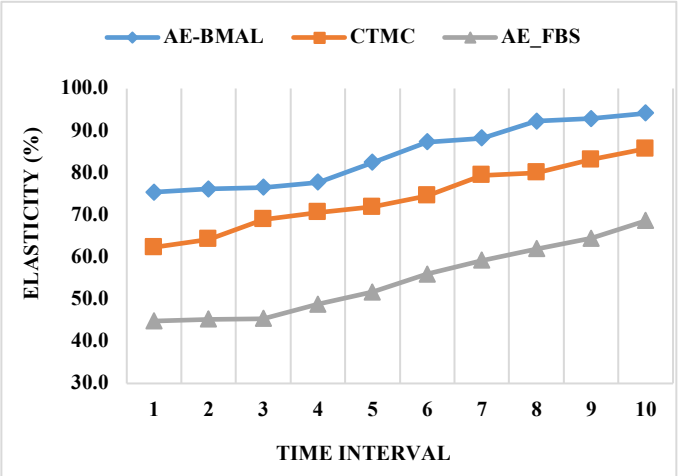


Fig .8. Comparison of elasticity in FIFA workloads

Figures 9 and 10 illustrate the elasticity of the AE_BMLA

algorithm in comparison with the CTMC and AE - FBS algorithms in the NASA and ClarkNet workloads. The elasticity of the proposed method is higher than the CTMC and AE_FBS algorithms at all three workloads.

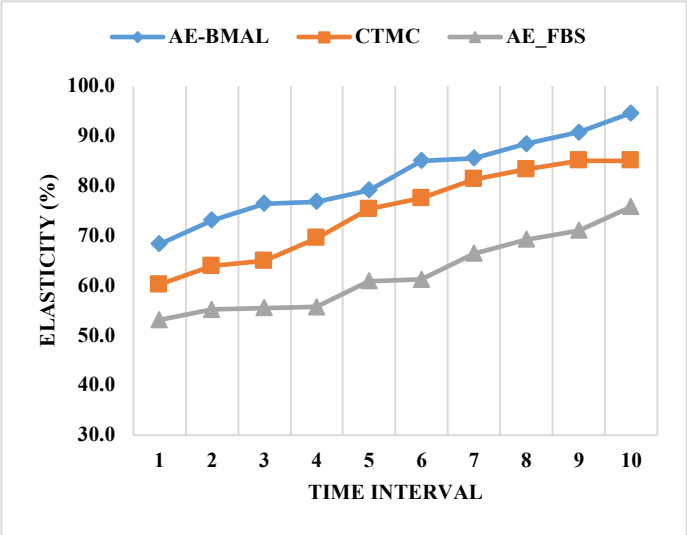


Fig .9. Comparison of elasticity in NASA workload

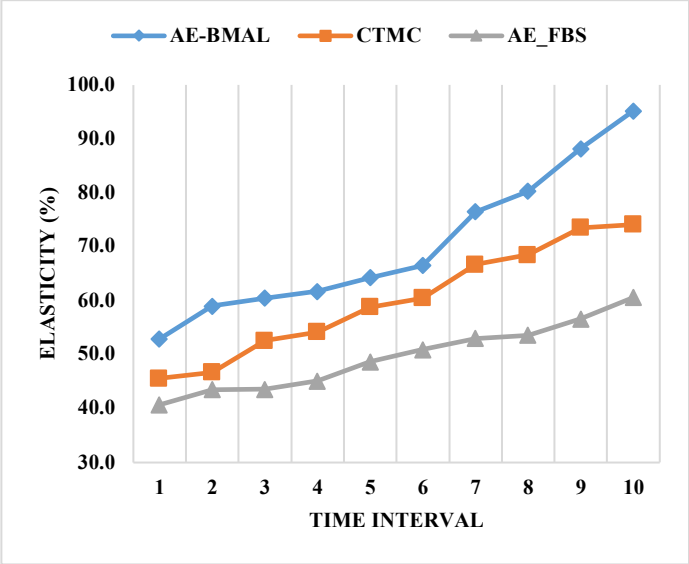


Fig. 10. Comparison of elasticity in ClarkNet workload

The average improvement of elasticity in the proposed method for all three workloads compared to the CTMC and AE_FBS algorithms is 13.3% and 42%, respectively.

C.2. Second scenario: Impact of response time metric

Response time as one of the most effective objectives of SLA plays an important role in choosing a virtual machine. If the requested response time identified by the allowed response time, does not meet, then the correct scaling must be done.

Figure 11 shows the average response time of the AE_BMLA algorithm in comparison with the two CTMC and AE_FBS algorithms in the FIFA workload. Due to the desirable performance of learning automata as the decision-maker in the increasing section of virtual machine system and the two-stage queuing structure and finally, the correct control of the controller section as one of the important decision- making factors, resource allocation is done correctly. More accurate resource allocation will increase the response rate. According to the results, the proposed method has a better performance about the response time. Figure 12 shows the average response time in the AE_BMLA algorithm compared to the two CTMC and AE_FBS algorithms in the NASA's work load.

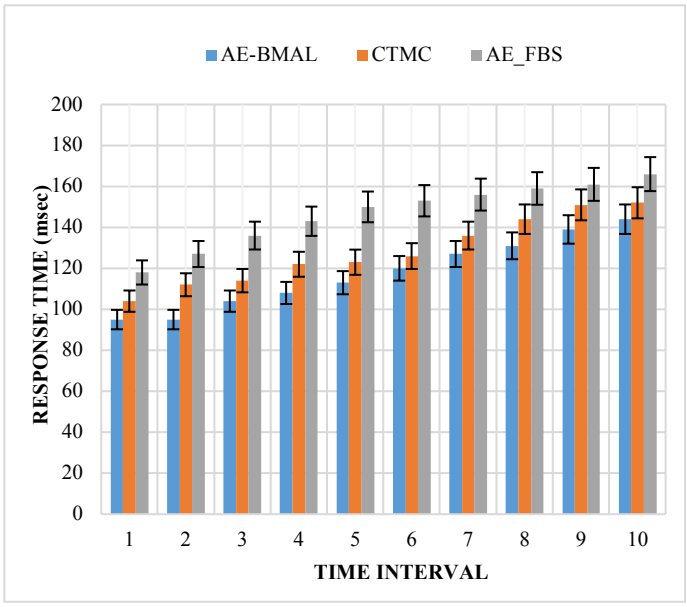


Fig. 11. Comparison of the average response time in FIFA workloads

According to Figure 13, the average response time of the ClarkNet workload in the proposed method is lower than the CTMC and AE_FBS algorithms. This is due to the proper performance of the learning automata and the appropriate buffer management. In all three workloads, the average response time in the proposed method is lower than the two CTMC and AE_FBS algorithms, which indicates the improved quality of service delivery in this structure. Compared to the CTMC and AE_FBS algorithms, the average response time in the proposed method for all three workloads has decreased by 11.4% and 18.8%, respectively.

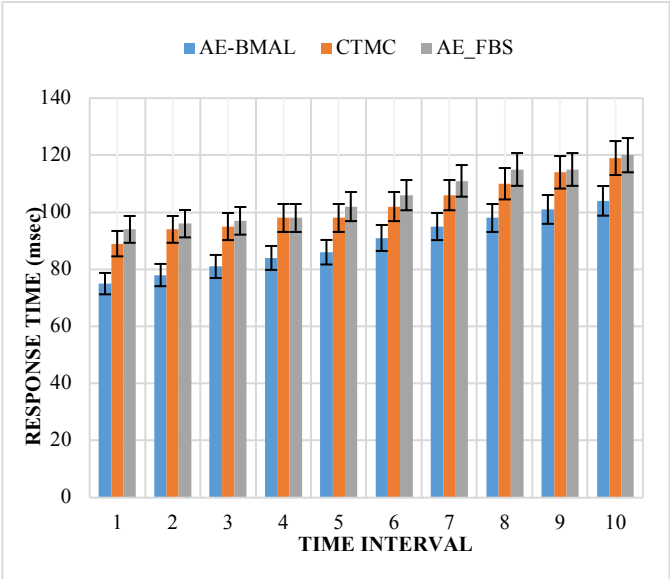


Fig. 12. Comparison of the average response time in NASA's workload

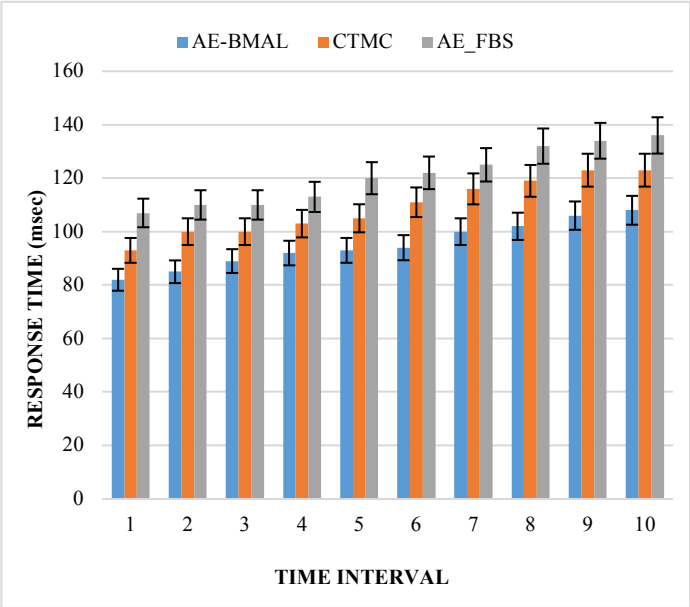


Fig. 13. Comparison of Average Response Time in ClarkNet Workload

C.3. Third scenario: Impact of utilization metric

Figure 14 shows the average utilization in the FIFA workload. The precise monitoring of learning automata over request logs is a very effective way to control the status of resources. Due to the optimal performance of the controller program and the use of learning automata, resource allocation is done in a desirable manner. Correct resource allocation will provide accurate service delivery to the requests. Specifically, due to the correct supply of resources, the processor is best suited for requests. Of course, when the number of requests is high, the utilization level may reach to threshold one, and this is due to the high number of requests that need to be received by the processor.

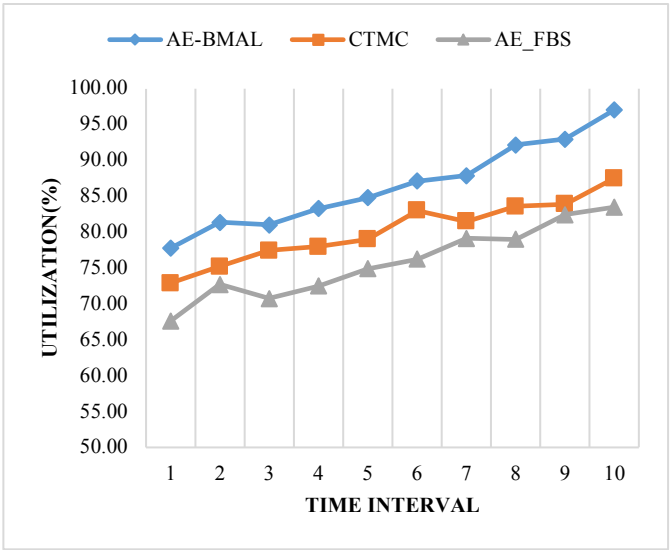


Fig. 14. Comparison of the utilization in the FIFA workload

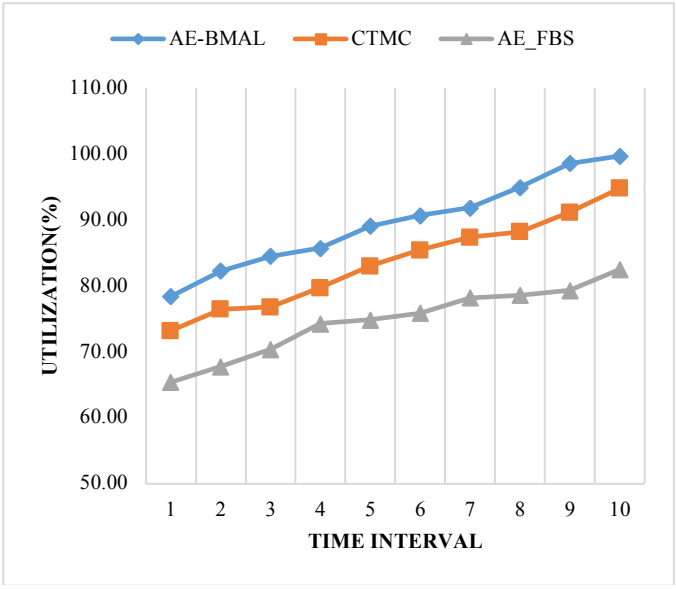


Fig. 15. Comparison of the utilization in the NASA workload

The use of buffer management has greatly influenced the proper control of requests and the decision making about the elasticity of the cloud system. It means that the learning automata can easily decide whether to increases/decreases the virtual machine for the occurrence or response to an incoming service. Figures 15 and 16 show the utilization of the AE_BMLA algorithm in comparison to the two CTMC and AE_FBS algorithms in NASA and ClarkNet workloads. In all three workloads, the utilization of the proposed method is higher than the CTMC and AE_FBS algorithms, which indicates an improvement in the quality of service delivery in this structure. Compared to the CTMC and AE_FBS algorithms, the average utilization in the proposed method for all three workloads increased by 8% and 17.2%, respectively.

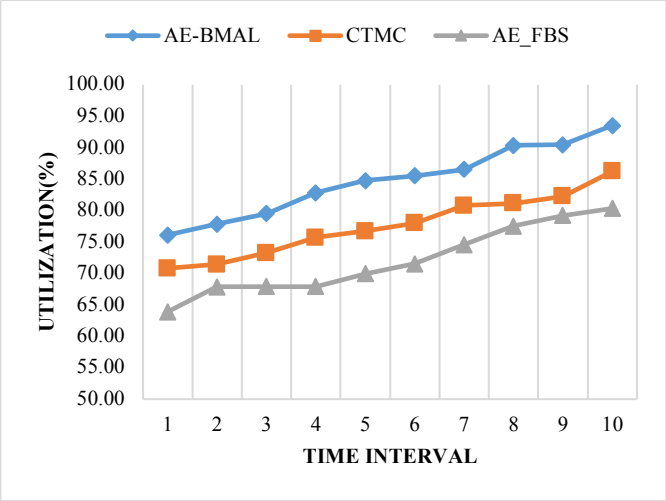


Fig. 16. Comparison of the utilization in the ClarkNet workload

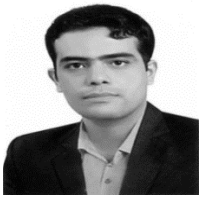
V. CONCLUSION

Cloud computing is the on-demand delivery of resources through a cloud platform via the internet to the end-users. The application providers use cloud infrastructures for hosting their applications due to its elasticity feature. The cloud elasticity feature allows application providers to grow or shrink computing resources on-demand, which enables automatic scaling of cloud resources according to workload changes. Therefore, any elasticity mechanism must have the capacity to estimate the desired resources to deal with workload fluctuations and satisfying the SLO requirements for avoiding over-provisioning or under-provisioning problems. In this paper, an approach was developed to improve elasticity using buffer management and centralized elastic management. The buffer management is responsible for controlling the input queue of the request, and the elastic management is responsible for controlling the elasticity of the system using the learning automata technique. We evaluated the proposed solution under real workloads traces, including three data sets of FIFA World Cup, ClarkNet and NASA, and their experimental results indicated that it significantly outperforms in terms of the elasticity value, response time, and CPU utilization compared with the other approaches. In future work, we will arrange to investigate: integration of the proposed solution with auto-scaling mechanisms, evaluation the proposed solution in a real cloud infrastructure such as OpenStack and extension of proposed solution using colored Petri Net models. Also, we will utilize the deep Q-learning instead of learning automata to gain higher accuracy.

REFERENCES

[1] K. Chandrasekaran, *Essentials of cloud computing*: Chapman and Hall/CRC, 2014.
[2] M. Aloqaily, I. Al Ridhawi, H. B. Salameh, and Y. Jararweh, "Data and service management in densely crowded environments: Challenges, opportunities, and recent developments," *IEEE Communications Magazine,* vol. 57, pp. 81-87, 2019.
[3] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering cloud computing: foundations and applications programming*: Newnes, 2013.
[4] Y. Kotb, I. Al Ridhawi, M. Aloqaily, T. Baker, Y. Jararweh, and H. Tawfik, "Cloud-Based Multi-Agent Cooperation for IoT Devices Using Workflow-Nets," *Journal of Grid Computing,* pp. 1-26, 2019.
[5] M. Ghobaei-Arani, R. Khorsand, and M. Ramezanpour, "An autonomous resource provisioning framework for massively multiplayer online games in cloud environment," *Journal of Network and Computer Applications,* 2019.
[6] S. S. Manvi and G. K. Shyam, "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey," *Journal of network and computer applications,* vol. 41, pp. 424-440, 2014.
[7] M. Aloqaily, S. Otoum, I. Al Ridhawi, and Y. Jararweh, "An intrusion detection system for connected vehicles in smart cities," *Ad Hoc Networks,* vol. 90, p. 101842, 2019.
[8] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic approach for resource provisioning of cloud services," *Cluster Computing,* vol. 19, pp. 1017-1036, 2016.
[9] S. Otoum, B. Kantarci, and H. T. Mouftah, "On the feasibility of deep learning in sensor network intrusion detection," *IEEE Networking Letters,* vol. 1, pp. 68-71, 2019.
[10] Q. Yaseen, F. AlBalas, Y. Jararweh, and M. Al-Ayyoub, "A fog computing based system for selective forwarding detection in mobile wireless sensor networks," in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, 2016, pp. 256-262.
[11] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer,* pp. 41-50, 2003.
[12] M. Al-khafajiy, T. Baker, H. Al-Libawy, Z. Maamar, M. Aloqaily, and Y. Jararweh, "Improving fog computing performance via fog-2-fog collaboration," *Future Generation Computer Systems,* vol. 100, pp. 266-280, 2019.
[13] I. Al Ridhawi, M. Aloqaily, B. Kantarci, Y. Jararweh, and H. T. Mouftah, "A continuous diversified vehicular cloud service availability framework for smart cities," *Computer Networks,* vol. 145, pp. 207-218, 2018.
[14] A. Rezvanian, A. M. Saghiri, S. M. Vahidipour, M. Esnaashari, and M. R. Meybodi, *Recent advances in learning automata* vol. 754: Springer, 2018.
[15] M. Ghobaei-Arani, A. A. Rahmanian, M. Shamsi, and A. Rasouli-Kenari, "A learning-based approach for virtual machine placement in cloud data centers," *International Journal of Communication Systems,* vol. 31, p. e3537, 2018.
[16] A. Ullah, J. Li, Y. Shen, and A. Hussain, "A control theoretical view of cloud elasticity: taxonomy, survey and challenges," *Cluster Computing,* vol. 21, pp. 1735-1764,

2018.
[17] M. Albonico, J.-M. Mottu, and G. Sunyé, "Controlling the elasticity of web applications on cloud computing," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 816-819.
[18] K. Salah, K. Elbadawi, and R. Boutaba, "An analytical model for estimating cloud resources of elastic services," *Journal of Network and Systems Management,* vol. 24, pp. 285-308, 2016.
[19] F. Zhang, X. Tang, X. Li, S. U. Khan, and Z. Li, "Quantifying cloud elasticity with container-based autoscaling," *Future Generation Computer Systems,* vol. 98, pp. 672-681, 2019.
[20] S. M. R. Nouri, H. Li, S. Venugopal, W. Guo, M. He, and W. Tian, "Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications," *Future Generation Computer Systems,* vol. 94, pp. 765-780, 2019.
[21] W. A. Hanafy, A. E. Mohamed, and S. A. Salem, "A New Infrastructure Elasticity Control Algorithm for Containerized Cloud," *IEEE Access,* vol. 7, pp. 39731-39741, 2019.
[22] A. B. Jrad, S. Bhiri, and S. Tata, "STRATFram: A framework for describing and evaluating elasticity strategies for service-based business processes in the cloud," *Future Generation Computer Systems,* vol. 97, pp. 69-89, 2019.
[23] Y. Al-Dhuraibi, F. Zalila, N. Djarallah, and P. Merle, "Model-Driven Elasticity Management with OCCI," *IEEE Transactions on Cloud Computing,* 2019.
[24] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada, "Fuzzy Self-Learning Controllers for Elasticity Management in Dynamic Cloud Architectures," in *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, 2016, pp. 70-79.
[25] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience,* vol. 41, pp. 23-50, 2011.
[26] K. Li, "Quantitative modeling and analytical calculation of elasticity in cloud computing," *IEEE Transactions on Cloud Computing,* 2017.
[27] D. Milios, G. Sanguinetti, and D. Schnoerr, "Probabilistic model checking for continuous-time Markov chains via sequential Bayesian inference," in *International Conference on Quantitative Evaluation of Systems*, 2018, pp. 289-305.



Mostafa Ghobaei-Arani received his Ph.D. degree in Software Engineering from Islamic Azad University, Science and Research Branch, Tehran, Iran. He has published more than 50 journal and conference papers in the area of distributed computing. 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 conferences international conference (https://publons.com/researcher/1267819/mostafa-ghobaei-arani/). His research interests include Distributed Computing, Cloud Computing, Autonomic Computing, Edge/Fog Computing, Exascale Computing, Soft Computing, and IoT.

Alireza Souri received his B.S. degree in Software Engineering from University College of Nabi Akram, Iran, and his M.Sc. and PhD degrees in Software Engineering from Science and Research Branch, Islamic Azad University, Iran. Up to now, he has authored/co-authored more than 40 academic articles. He currently is Associate Editor of Human-Centric Computing and Information Sciences (Springer), Cluster Computing (Springer) and IET Communications (IEEE) journals. His research interests include Formal Specification & Verification, Model checking, Fog & Cloud computing, IoT, Data mining and social networks.

Thar Baker is Senior Lecturer in Distributed Systems Engineering and Head of Applied Computing Research Group (ACRG) in the Faculty of Engineering and Technology at Liverpool John Moores University (LJMU, UK). He received his PhD in Autonomic Cloud Applications from LJMU in 2010, and became a Senior Fellow of Higher Education Academy (SFHEA) in 2018. Dr Baker has published numerous refereed research papers in multidisciplinary research areas including: Big Data, Algorithm Design, Green and Sustainable Computing, and Energy Routing Protocols. Dr Baker has been actively involved as member of editorial board and review committee for a number of peer-reviewed international journals, and is on program committee for a number of international conferences. For example, he is Associate Editor of Future Generation Computer System. Dr. Baker is Expert Evaluator of EU H2020, ICTFund, and British Council.

Aseel Hussien is a Program Leader in Building Surveying and Facilities Management at Liverpool John Moores University. She holds a (BSc) Degree in Architecture Engineering, and a Master of Science (MSc) in Computing Information Systems. Her Doctor of Philosophy Degree (PhD) was obtained from Liverpool John Moores University titled ARGILE: A Conceptual Framework Combining Augmented Reality with Agile Philosophy for the UK Construction Industry. Aseel's research interest related to the virtual reality, smart city, augmented reality, agile project management, and Building information model BIM.