Auto-Scaling Approach for Cloud based Mobile Learning Applications

Amani Nasser Almutlaq[1], Dr. Yassine Daadaa[2]
College of Computer Sciences and Information
Al Imam Mohammad Ibn Saud Islamic University (IMSIU)
Riyadh, Saudi Arabia

***Abstract*—In the last decade, mobile learning applications have attracted a significant amount of attention. Huge investments have been made to develop educational applications that can be implemented on mobile devices. However, mobile learning applications have some limitations, such as storage space and battery life. Cloud computing provides a new idea to solve some limitations of mobile learning applications. However, there are other limitations, like scalability, that must be solved before mobile cloud learning can become completely operational. There are two main problems with scalability. The first occurs when the application server's performance declines due to an increase in the number of requests, which affects usability. The second is that a decrease in the number of requests makes most application servers idle and therefore wastes money. These two problems can be avoided or minimized by provisioning auto-scaling techniques that permit the acquisition and release of resources dynamically to accommodate demand. In this paper, we propose an intelligent neuro-fuzzy reinforcement learning approach to solve the scalability problem in mobile cloud learning applications, and evaluate the proposed approach against some of the existing approaches via MATLAB. The large state space and long training time required to find the optimal policy are the main problems of reinforcement learning. We use fuzzy Q-learning to solve the large state space problem by grouping similar variables in the same state; there is then no need to use large look-up tables. The use of parallel learning agents reduces the training time needed to determine optimal policies. The experimental results prove that the proposed approach is able to increase learning speed and reduce the training time needed to determine optimal policies.**

Keywords*—Auto-scaling; reinforcement learning; fuzzy Q-learning

I. Introduction

Cloud computing is a computing business paradigm where services such as servers, storage, and applications are delivered to end users through the internet. There are three categories of cloud computing [1] [2] [3] [4]: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS includes storage, servers, and networking components. Amazon EC2 [5] is a suite that is built on an IaaS service model. PaaS provides the platforms (e.g. operating systems) needed to develop and run applications, such as the Google App Engine [6]. Software as a Service (SaaS) offers access to web-based software and its functions, including services such as Salesforce.com [7]. There are three deployment methods for cloud computing [3] [8]: private, public, and hybrid. Private clouds are provisioned for use by a single organization while public clouds are provisioned for open use. Hybrid is a combination of both private and public clouds.

Over the past decade, many universities, schools and other educational institutions have moved their e-Learning applications to mobile learning applications. Mobile learning applications [9] are the most important e-Learning model, using handheld devices such as smart phones and tablets. Mobile learning applications have many limitations, however, such as storage space, battery life, and potential data loss. To solve some of these limitations, mobile cloud learning (MCL) applications have been proposed.

MCL integrates the advantages of mobile learning and cloud computing. The main advantages of MCL are solving the data storage limitation in mobile learning by storing data in the cloud rather than in the device, increasing the ease of sharing knowledge, easing accessibility as access is through a browser rather than a mobile operating system, and low costs for set-up and maintenance.

There are some limitations, like scalability, that must be solved before MCL can become completely operational. Scalability refers to resource allocation that can be acquired or released depending on demand. Cloud scalability has two dimensions:

- Horizontal cloud scalability (scaling out): adding more servers that perform the same work, and
- Vertical cloud scalability (scaling up): increasing capacity by adding more resources, such as adding processing power to a server to make it faster.

Most cloud providers use horizontal scalability because vertical scaling requires rebooting. Auto scaling automatically scales up or down the capacity; this allows the system to maintain performance while also saving money. The auto scaling system needs two elements: a monitor and the scaling unit. There are different performance metrics for scaling purposes, such as CPU utilization, the size of the request queue, and memory usage.

There are two approaches for automatically matching computing requirements with computing resources: schedule-based and rule-based. In schedule-based scaling, the scale adjusts by days and times, so it cannot respond to unexpected changes. Rule-based scaling is dependent upon creating two rules to determine when to scale, such as reinforcement learning (RL).

The premise of RL is learning through trial-and-error from the learner's performance and feedback from the environment.

It captures the performance model of a target application and its policy without any a priori knowledge [10] [11] [12] [13] [14]. There are four fundamental components in RL: agent, state, action, and reward. The agent is the decision-maker that learns from experience. A state s can be defined as w, u, or p, where w is the total number of user requests observed in a given time period, u is the number of virtual machines (VMs) allocated to the application, and p is the performance in terms of the average response time to requests. The action is what the agent can do (e.g. add or remove application resources). Each action is associated with a reward. The objective is for the agent to choose actions so as to maximize the expected reward over a given period of time.

Neuro-fuzzy systems [15] [16] [17] is field of artificial intelligence based on neural networks and fuzzy logic, in which truth values may range from 0 to 1.

The rest of the paper is organized as follows. Section 2 provides an overview of related work and we provide an explanation of our proposed approach in Section 3. Experimental results and their analysis are presented in Section 4. Finally, we conclude the paper and discuss future work in Section 5.

II. Related Work

S. Chen et al. [1] proposed a model for an MCL system consisting of four layers: infrastructure, platform, business application, and service access. The infrastructure layer includes system resources (i.e. CPU, network, and storage) which are represented by a virtual resource that provides scalable and flexible services. The platform layer provides software development, application services, database services, data storage, and recovery services. The business application layer supports different application software modules, such as a learning module which could provide self-learning for students and allow teachers to review students' results. Such a teaching module would allow teachers to manage courses, while a communication module would provide a communication method for teachers and students, such as SMS or a blog. A system administration module would provide system management and access control. The service access layer would then work as an interface for students and teachers.

In mobile cloud computing (MCC), data processing and storage are performed outside the mobile device and inside the cloud, offering many applications. In [18], Arun and Prabu discuss some of these applications, including vehicle monitoring, mobile learning, biometry, and digital forensic analysis.

Veerabhadram and Conradie [19] proposed an architecture for MCC, consisting of three main parts: the mobile client, middleware, and cloud services. The mobile client is the mean by which the user can access the system (e.g. a smartphone) and the middleware pushes service updates to mobile clients. The main goal of the architecture is to provide a proxy for mobile clients to connect to cloud services. The authors used a questionnaire to gather the views of educators and students on mobile learning. The results indicate that MCC will be an important technology for education in the near future. Accordingly, a model for mobile cloud learning systems and their applications has been proposed in [20]. The structure of this model also has three layers: user, system, and application. The user layer authenticates users, the system layer contains system resources (CPU, network, and storage), and the application layer contains learning system processes and a test.

Kitanov and Davcev [2] proposed a new model for high performance computing using a high performance computing cluster infrastructure. Cisco's WebEx mobile cloud applications have been used to test remote learning in both fixed and mobile environments and for a variety of educational scenarios; WebEx Whiteboard as a tool for teachers in remote learning environments and Telemedicine to share and highlight medical images. The test relied on the Quality Of Experience (QOE), which measures users' satisfaction. The QOE was evaluated via questionnaire to the participants after the completion of the remote learning course. The result implied that remote learning in a mobile environment is easier than in a fixed environment.

P. Hazarika et al. [21] classified the MCC challenges into three categories: technical, security, and miscellaneous. The goal of MCC is to have seamless user interaction reach its full potential. However, this presents some critical technical challenges like data latency, service unavailability, and heterogeneous wireless networks interfaces (WCDMA, GPRS, WiMAX, WLAN). Security challenges are classified into three categories: cloud services, communication channels, and mobile applications. Network accessibility and cloud compliance are examples of miscellaneous challenges. To illustrate, using MCC without network access is useless. Likewise, compliance problems like regulation may affect the MCC user; due to the nature of the cloud, data may span different regions, with each region having different regulations for the stored data.

Chao and Yue [22] presented different methods of access modes for mobile learning based on cloud computing. The first method is mobile learning based on SMS. In this method, the user sends a message from a mobile device through the internet to the teaching server. The teaching server analyzes and processes the data, then sends the requested data back to the user's mobile phone. The second method is mobile learning based on webpages. In this method, the user accesses the internet and visits the mobile website that contains learning resources, including text, images, sound, animation, video, and other media forms. The third method is mobile learning based on a micro-blog. This method is similar to a blog but each message is restricted to only 140 words; the user can send ideas in the form of messages to mobile phone users and a personalized website group. The final method is multimedia interactive learning based on a Wireless Application Protocol (WAP) browser. A WAP browser is a web browser for mobile devices. WAP browsing is similar to computer browser applications but improves content performance.

A proposed algorithm for parallel learning agents was presented in [23]. The authors aimed to accelerate the exploration procedure and reduce the training time to determine optimal policies by using parallel learning agents (swarm behaviors). They proposed a neuro-fuzzy system with an actor-critic method, a kind of RL methodology. The actor is used to select an action and the critic is used to evaluate the action chosen. The proposed algorithm focuses on two stages for each individual agent. First, it classifies the input state via fuzzy net. Then, the actor-critic method is applied. Each agent is independent from one other and the adaptive swarm behavior

is acquired only as a reward from the environment. Simulation results from this algorithm show that the swarm behavior is a quicker exploration procedure than individual learning. This algorithm does not balance exploration and exploitation because it uses a fixed value for the learning rate.

In [24], a solution was proposed to solve the problem of managing the balance between exploration and exploitation that was present in [23]. The authors proposed an adaptive learning rate, which uses larger learning rates for less visited states and smaller learning rates for more visited states. The authors showed how the adaptive learning rate affected a neuro-fuzzy system with SARSA learning; simulation results from this algorithm showed the effectiveness of the adaptive learning rate.

In [25], an algorithm was proposed to balance exploration and exploitation in a multi-agent environment, using the ξ-greedy method. Random action (exploration) is selected by the ξ parameter and is updated in each time step. Three fuzzy control parameters are used to update ξ: the weighted difference between maximum and minimum move values in the current state, the difference value of the current rate, and the previous state and exploration rate. One of the drawbacks of this method is the long time it requires for the learning process.

The authors in [26] compared two classic RL algorithms, fuzzy SARSA learning (on-policy) and fuzzy Q-learning (off-policy). SARSA compares the current state with the actual next state. Q-learning compares the current state with the best possible next states.

In [27], an algorithm was proposed to combine a fuzzy logic controller and fuzzy Q-learning to increase performance and minimize costs. It is assumed that there is no prior knowledge of policies and the fuzzy rules are automatically updated to learn optimal policies during the runtime to improve its performance. This algorithm is good for dynamic workloads because of its capabilities for self-adapting and self-learning.

M. Sharafi et al. [28] combine an RL algorithm (SARSA learning) with fuzzified actions. They test their proposed method by simulation using MATLAB and show that this algorithm is efficient for a dynamic workload.

In [29], Kao-Shing Hwang and Wei-Cheng Jiang proposed shaped-Q learning for multi-agent systems. In the architecture, each agent maintains a cooperative tendency table. The action with the maximal shaped Q-value in this state will be selected. This method can make agents complete the task together more efficiently and speed up the learning process.

III. The Proposed Approach

Our proposed method combines fuzzy Q-learning [30] with a proposed parallel agents technique in order to solve the two main problems of RL: large state space and long training time.

The main components of the architecture are fuzzy Q-learning and the proposed parallel agents technique. Fuzzy Q-learning is used to solve the large state space problem, in which a similar group of variables belongs to the same state rather than using large look up tables. Parallel agents are used to reduce the training time needed to determine optimal policies.

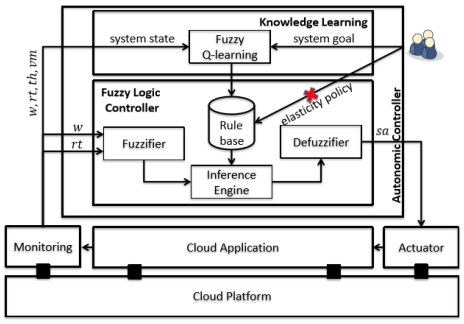

Fig. 1. Fuzzy Q-Learning Architecture [27]

The distinct components of the architecture are elaborated below.

A. Fuzzy Q-learning

The architecture of each individual agent consists of two parts - the fuzzy logic controller and fuzzy Q-learning, as shown in Fig. 1. The fuzzy logic controller takes the observed data and generates scaling actions through fuzzy rules (rules are generated by fuzzy Q-learning). The inputs to the fuzzy logic controller are workload (w) and response time (RT). The output is (sa) in terms of adding or removing of the number of virtual machines (VMs).

The first step for the fuzzy logic controller is partitioning the input to many fuzzy sets by membership functions $\mu y(x)$, the degree of membership of an input signal x to the fuzzy set y. Membership function is a curve that defines how each input is mapped to a membership value between 0 and 1. In this thesis, we use triangular and trapezoidal membership functions. The fuzzy sets of w are divided into linguistic values *Low, Medium* and *High*. The fuzzy sets of RT are divided into linguistic values *Bad, Okay* and *Good*. The output is an integer constant from the interval $\{-2, -1, 0, +1, +2\}$.

The next step is defining fuzzy if-then rules for the form if X is A, then Y is B, where A and B are linguistic values defined by the fuzzy set. For example, if workload is high and response time is bad, then add VMs.

The three steps that the fuzzy logic controller performs are:

1) Fuzzification of the inputs: the first step is partitioning the state space of each input variable into various fuzzy sets through membership functions. The fuzzification process is a transfer from crisp value to linguistic value by membership functions.
2) Fuzzy reasoning: this step performs the operation in the rule and founds the scaling action.
3) Defuzzification of the output: the process of transferring the linguistic value to a crisp value. To calculate the output action, use equation 1. N is the number of rules, $\mu_i(x)$ is the degree of truth of the rule, i for the input signal, and x and a_i is the consequent function for the same rule.

$$y(x) = \sum_{i=1}^{N} \mu_i \times a_i \quad (1)$$

TABLE I. INITIALIZED Q-TABLE VALUES ($q[i, j]$) TO 0

State (W, RT) / Action	+2	+1	0	-1	-2
High, Bad	0	0	0	0	0
High, Okay	0	0	0	0	0
High, Good	0	0	0	0	0
Medium, Bad	0	0	0	0	0
Medium, Okay	0	0	0	0	0
Medium, Good	0	0	0	0	0
Low, Bad	0	0	0	0	0
Low, Okay	0	0	0	0	0
Low, Good	0	0	0	0	0

The fuzzy logic controller starts working with the rules provided by users. There are limitations for the fuzzy logic controller because it uses fixed fuzzy rules. The rules are defined by the user and may not be the optimal policies. To solve this problem, fuzzy Q-learning is needed.

Fuzzy Q-learning can start working with no prior knowledge base and obtains knowledge at runtime through the knowledge evolution mechanism. It learns the policies and tries to choose the action that returns a good reward. The objective of the agent is to maximize the received reward, as described in equation 2:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{i=1}^{\infty} \gamma^k r_{t+k+1} \qquad (2)$$

It does not always choose the action with a high reward because a different action may lead to better rewards in the future. Therefore, there is a trade-off between exploitation and exploration; exploitation utilizes known information to maximize rewards while exploration discovers more information about the environment. Fuzzy Q-learning continuously updates the rules.

The algorithm for the fuzzy logic controller is summarized in Algorithm 1. First, Q-table values ($q[i, j]$) are initialized to 0 as shown in Table I. Then, an action is selected for each fired rule. The control action is calculated by the fuzzy controller, as described in equation 1. After that, the Q function is approximated from the current Q-values and the firing level of the rules. $Q(s, a)$ denotes this Q function and it is defined in RL to determine the benefit of taking action a in state s. Then, once the action is taken, the system goes to the next state $s(t + 1)$. The reward $r(t + 1)$ is observed and the value for the new state is computed. Finally, error signal and Q-values are calculated and updated respectively. The space complexity is $O(N * J)$,where N is the number of states and J is the number of actions. For example, if the number of states is 9 and the number of actions is 5, then the space complexity is $O(9 * 5)$ which equals 45 q-values, as clarified in Table I.

The reward function is defined based on SLO violations criteria. To illustrate, the action is appropriate if the response time is less than or equal to SLO, and the reward takes the value 1. The action is not effective and the reward is 0 if the response time is greater than SLO and less than the previous response time. In the other cases, the action is not appropriate and the reward takes a negative value.

Algorithm 1 Fuzzy Q-learning algorithm

1: **Initialize** q-values in the look-up table to 0:
q[i, j] = 0, 1<i<N , 1<j<J , N is the number of states and J is the number of actions.
2: Select an action for each activated rule (ϵ-greedy policy):
$a_i = argmax_k\ q[i, k]$ with probability 1-ϵ,
$a_i = random\{ak, k = 1, 2, \cdots, J\}$ with probability ϵ
3: Calculate the control action by the fuzzy logic controller:
$a = \sum_{i=1}^{N} \mu_i(x) \times a_i$
4: Approximate the Q function from the current q-values and the degree of truth of the rules:
$Q(s(t), a) = \sum_{i=1}^{N} \mu_i(S) \times q[i, a_i]$ where $Q(s(t), a)$ is the value of the Q function for the current state $s(t)$ in iteration t and the action a
5: Take action a and leave the system to evolve to the next state, $s(t + 1)$.
6: Observe the reward signal, r(t + 1), and compute the value for the new state denoted by $V(s(t + 1))$:
$V(s(t + 1) = \sum_{i=1}^{N} \mu_i(s(t + 1) \cdot max_k\ (q[i, a_k])$
7: Calculate the error signal:
$\triangle Q = r(t+1)\ + \gamma\ \times\ V_t(s(t+1))\ -\ Q(s(t), a)$ where γ is a discount factor
8: Update q-values:
$q[i, ai]\ =\ q[i, ai]\ +\ \eta \cdot \triangle Q \cdot \mu_i\ (s(t))$ where η is a learning rate
9: Repeat the process starting from **step 2** for the new state until it converges.

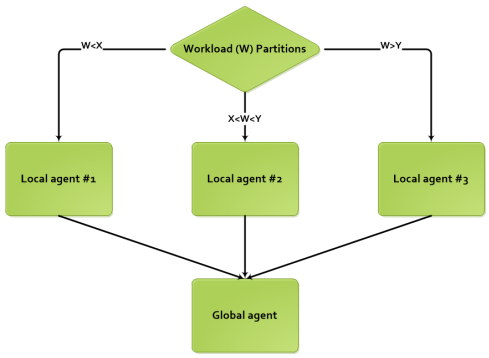

Fig. 2. The proposed parallel agent with state partition technique.

B. Parallel Agent

In this section we propose a new approach of Parallel Reinforcement Learning with State Space Partitioning. We divide the state space into multiple partitions, and PRL agents are assigned to explore each specific region, with the goal of increasing the exploration and improving the learning speed. There are two types of agents in our PRL implementation- one global agent and many local agents as shown in Fig. 2 Both are based on fuzzy Q-learning and each agent independently maintains a fuzzy Q-learning. Fuzzy Q-learning for local and global agent value estimates are initialized to 0. At each time step, the knowledge learned by all local agents is synchronized with the global agent.

Each local agent selects actions using the ϵ-greedy strategy, where a random action is chosen with probability ϵ, or the action with the best expected reward is chosen with the

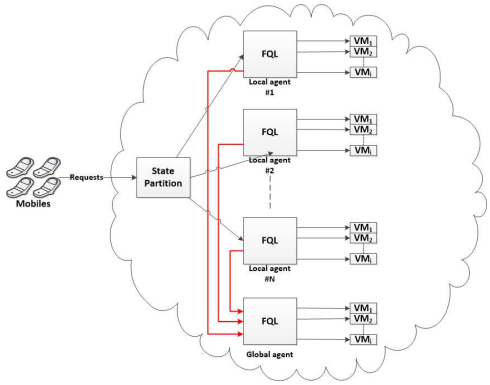

Fig. 3. The proposed parallel agent with state partition technique.

remaining probability $1-\epsilon$. The global agent always chooses the action with the maximum Q-value for a given state.

The steps that summarized the proposed method are shown below:
step 1: Divide the state space into multiple partitions.
step 2: Assign each partition to an agent.
step 3: All local and global agents are initialized to 0.
step 4: Local agents are swapped between exploitation and exploration while the global agent takes only exploitation.
step 5: At each time step, the knowledge learned by all local agents is synchronized with the global agent.

C. Combine Fuzzy Q-learning with the Proposed Parallel Agent Technique

The architecture for combining fuzzy Q-learning and parallel agents is shown in Fig. 3. Users send requests using a mobile learning application. The state space is divided into multiple partitions and each state partition directs the incoming requests to its local agent. The agent (local or global) schedules the requests that arrive from the users. These requests are distributed evenly based on a certain load balancing method, such as least connection or round robin. Also, each agent is responsible for auto-scaling and monitoring its region. The local agent receives all the incoming requests and forwards them to one of the servers in the pool. At each time step, the knowledge learned by all local agents is synchronized with the global agent.

The procedure of combining fuzzy Q-learning and the proposed parallel agent technique is described in Algorithm 2.

IV. Evaluation

In this section, we illustrate the dataset and the experimental setup of the proposed technique. Also, we present and discuss the experimental results of the proposed parallel agent with the state space partitioning technique.

A. Dataset

We have evaluated the performance of our proposed technique by using a dataset from ClarkNet, a full-access internet provider for the Baltimore-Washington DC metropolitan area, that contains two week's worth of all HTTP requests to the ClarkNet WWW server.

Algorithm 2 The proposed algorithm for combining fuzzy Q-learning and parallel agents

1: Divide the state space into multiple partitions.
2: Assign each partition to a local agent.
3: Initialize q-values of local and global agents in the look-up tables to 0.
4: Send each state to its local agent depending on the state-partition.
5: All agents work in parallel and follow steps 6 through 13:
6: Select an action for each activated rule (ϵ-greedy policy):
$a_i = argmax_k\ q[i,k]$ with probability 1-ϵ,
$a_i = random\{ak, k = 1, 2, \cdots, J\}$ with probability ϵ.
7: Calculate the control action by the fuzzy logic controller:
$a = \sum_{i=1}^{N} \mu_i(x) \times a_i$.
8: Approximate the Q function from the current q-values and the degree of truth of the rules:
$Q(s(t), a) = \sum_{i=1}^{N} \mu_i(S) \times q[i, a_i]$ where $Q(s(t), a)$ is the value of the Q function for the current state $s(t)$ in iteration t and action a.
9: Take action a and leave the system to evolve to the next state, $s(t+1)$.
10: Observe the reward signal, $r(t+1)$, and compute the value for the new state denoted by $V(s(t+1))$:
$V(s(t+1) = \sum_{i=1}^{N} \mu_i(s(t+1) \cdot max_k\ (q[i, a_k])$.
11: Calculate the error signal:
$\triangle Q = r(t+1)\ + \gamma\ \times\ V_t(s(t+1))\ -\ Q(s(t), a)$ where γ is a discount factor.
12: Update q-values:
$q[i, ai]\ = q[i, ai]\ +\ \eta \cdot \triangle Q \cdot \mu_i\ (s(t))$ where η is the learning rate.
13: The knowledge learned by all local agents is synchronized with the global agent.
14: Repeat the process starting from **step 4** for the new state until it converges.

TABLE II. Experiment Parameters

Parameter	Value
Discount factor	0.8
Fixed learning rate	0.1
Adaptive learning rate (min, max)	0.001, 0.3
Epsilon	0.1
Min VM instances	1
Max VM instances	Default value is infinity

B. Experiment Setup

Experiments were conducted to evaluate whether the proposed parallel agents with the state space partitioning technique reduces the training time needed to determine optimal policies. The fixed learning rate in the experiments were set to a constant value $\eta = 0.1$ and the adaptive learning rate minimum and maximum were set to 0.001 and 0.3 respectively. The discount factor was set to $\gamma = 0.8$. The minimum and maximum number of VM instances were set to 1 and infinity respectively. The trade-off between exploitation and exploration to determine more information about the environment was set with an Epsilon value of 0.1. Table II shows the parameters that have been used in the experiments.

In our approach the inputs are: workload w and response time RT and output is scaling action sa in terms of incre-

TABLE III. NUMBER OF STATES

State #	W	RT
1	High	Bad
2	High	Okay
3	High	Good
4	Medium	Bad
5	Medium	Okay
6	Medium	Good
7	Low	Bad
8	Low	Okay
9	Low	Good

TABLE IV. NUMBER OF ACTIONS

Action #	SA
1	2
2	1
3	0
4	-1
4	-2

ment or decrement in the number of virtual machines VMs. Workload represents all HTTP requests to the ClarkNet WWW server. Workload w Range is $[0-100]$ and the fuzzy sets of workload are Low $[0-20]$, Medium $[10-60]$, and High $[40-100]$. The response time for a workload is computed as:

$$RT = PT + QT \tag{3}$$

The execution time (PT) would be computed as:

$$PT = workload \times \frac{CPI}{CPU\_SPEED} \tag{4}$$

where, CPU_SPEED is CPU speed in Hz, CPI is the average cycle per instruction (request). The analysis of the average queuing time is complicated and depends on several factors. It can be estimated by modeling the environment as M/M/N queuing system (M = distribution of the inter-arrival times (negative exponential distribution), N = number of servers (VMs))

However, for this environment, we might assume the queuing time (QT) is inversely proportional to the number of active VMs:

$$QT = \frac{C\_VM}{VM} \tag{5}$$

where, VM is the number of active VMs (initially = 1), C_VM is the Coefficient of proportionality of the queuing time and the number of active VMs. RT range within the interval $[0-100]$, and the fuzzy sets of response time are *Good* $[0-30]$, *Okay* $[20-80]$, and *Bad* $[70-100]$. Output function is a constant value, which can be an integer in $-2, -1, 0, +1, +2$ which is associated to the change in the number of VM.

There are nine states, as shown in Table III, and five actions, as shown in Table IV.

First, we divide the state space into 3 partitions - local agents #1, #2, and #3.

TABLE V. INITIALIZED LOCAL AGENT #1 Q-TABLE VALUES TO 0

State(W,RT) / Action	+2	+1	0	-1	-2
High, Bad	0	0	0	0	0
High, Okay	0	0	0	0	0
High, Good	0	0	0	0	0

TABLE VI. INITIALIZED LOCAL AGENT #2 Q-TABLE VALUES TO 0

State (W,RT) / Action	+2	+1	0	-1	-2
Medium, Bad	0	0	0	0	0
Medium, Okay	0	0	0	0	0
Medium, Good	0	0	0	0	0

TABLE VII. INITIALIZED LOCAL AGENT #3 Q-TABLE VALUES TO 0

State(W,RT) / Action	+2	+1	0	-1	-2
Low, Bad	0	0	0	0	0
Low, Okay	0	0	0	0	0
Low, Good	0	0	0	0	0

TABLE VIII. INITIALIZED GLOBAL AGENT Q-TABLE VALUES TO 0

State (W, RT) / Action	+2	+1	0	-1	-2
High, Bad	0	0	0	0	0
High, Okay	0	0	0	0	0
High, Good	0	0	0	0	0
Medium, Bad	0	0	0	0	0
Medium, Okay	0	0	0	0	0
Medium, Good	0	0	0	0	0
Low, Bad	0	0	0	0	0
Low, Okay	0	0	0	0	0
Low, Good	0	0	0	0	0

Table V shows local agent #1 with fuzzy set workload (w) and Range High [40-100].

Table VI demonstrates local agent #2 with fuzzy set workload (w) and Range Medium [10-60].

Table VII shows local agent #3 with fuzzy set workload (w) and Range Low [0-20].

We then initialized global agent values to 0 as shown in Table VIII.

C. Experimental Results

The initial design-time surface is not shown as it is a constant plane at point zero. Fig. 4, 6, and 8 show the temporal evolution of the control surface of the fuzzy controller for agents #1, #2, and #3 respectively; the surface evolves until the learning converges. The second surface is presented in Fig. 5, 7, and 9, where the learning has converged for agents #1, #2, and #3, respectively.

1) Global Agent: The initial design-time surface is not shown as it is a constant plane at point zero. Fig. 10 shows the temporal evolution of the control surface of the fuzzy controller; the surface evolves until the learning converges. The second surface is presented in Fig. 11, where the learning has converged.

Table IX demonstrates that parallel agents can reduce the training time needed to determine optimal policies, as compared to some of the existing approaches.

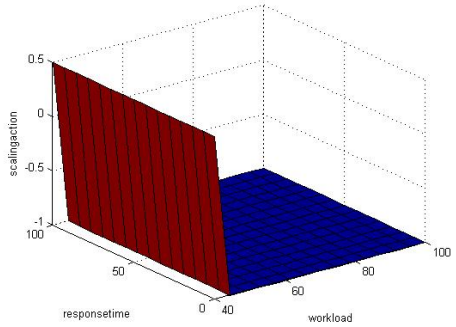

Fig. 4. Agent #1 temporal evolution of the control surface

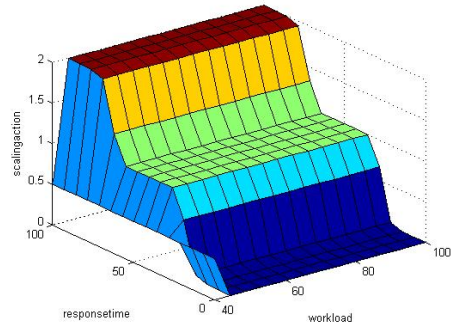

Fig. 5. Agent #1 where learning has converged

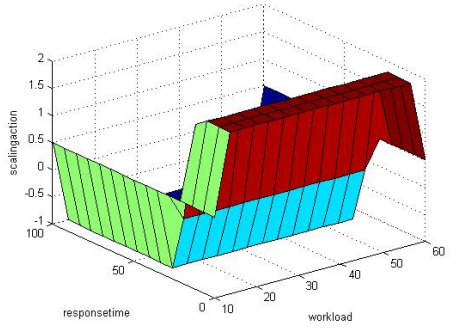

Fig. 6. Agent #2 temporal evolution of the control surface

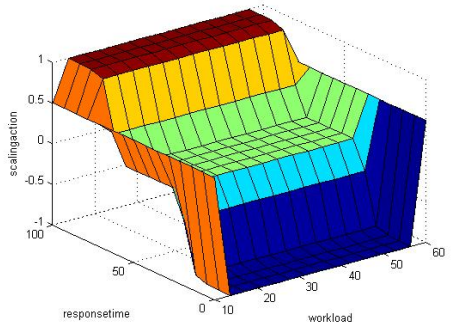

Fig. 7. Agent #2 where learning has converged

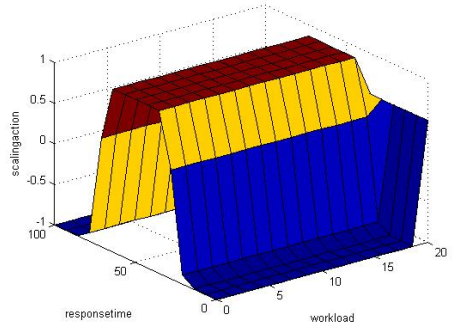

Fig. 8. Agent #3 temporal evolution of the control surface

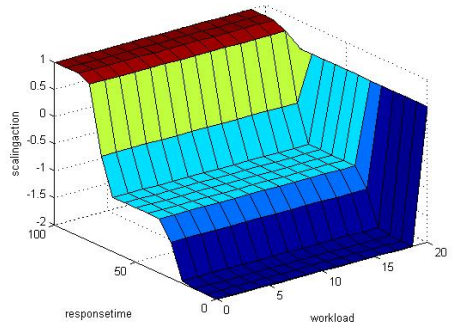

Fig. 9. Agent #3 where learning has converged

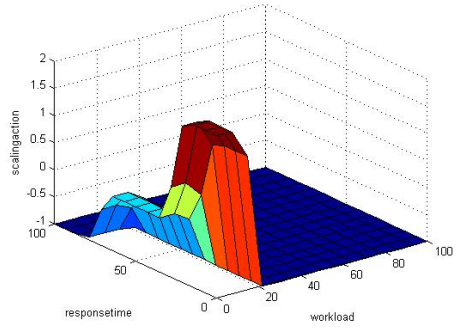

Fig. 10. Global agent temporal evolution of the control surface

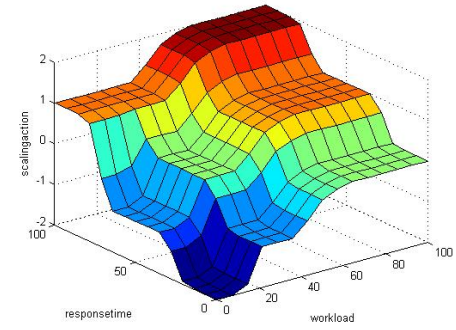

Fig. 11. Global agent when learning has been converged

TABLE IX. COMPARING THE TOTAL TRAINING TIME NEEDED FOR THE PROPOSED APPROACH AND SOME EXISTING APPROACHES TO DETERMINE OPTIMAL POLICIES

Authors & References	Method	Total Time
P. Jamshidi et al. [26]	Fuzzy Q-Learning (Fixed Learning Rate)	82.563 s
M. Sharafi et al. [28]	Fuzzy SARSA Learning (Fixed Learning Rate)	86.959 s
T. Kuremoto et al. [23]	Fuzzy SARSA Learning (Adaptive Learning Rate)	95.049 s
Proposed Method	Fuzzy Q-Learning (Fixed Learning Rate) with parallel learning agent	19.215 s

V. CONCLUSION

In this paper, a new parallel reinforcement learning technique with a fuzzy Q-learning algorithm has been proposed. In our solution we divide the state space into multiple partitions, and PRL agents are assigned to explore each specific region. There are two types of agents in our PRL implementation- one global agent and many local agents. We have evaluated our approach experimentally and proven that parallel agents can increase learning speed and reduce the training time needed to determine optimal policies as compared to some existing approaches. As part of our future work, we plan to evaluate this technique as a solution to other problems, such as smart grids. We plan to use automated partitioning strategies, instead of manual partitioning using human knowledge, because a good partitioning strategy is one of the challenges for new applications.

REFERENCES

[1] S. Chen, M. Lin, and H. Zhang, "Research of mobile learning system based on cloud computing," in *Proceeding of the International Conference on e-Education, Entertainment and e-Management*, 2011.

[2] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling techniques for elastic applications in cloud environments," *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09-12*, 2012.

[3] J. Srinivas, K. V. S. Reddy, and A. M. Qyser, "Cloud computing basics," *International journal of advanced research in computer and communication engineering*, vol. 1, no. 5, 2012.

[4] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[5] Amazon, "Amazon elastic compute cloud (amazon ec2)," http://aws.amazon.com/ec2/.

[6] G. A. Engine, "Google app engine," https://developers.google.com/appengine/, 2009.

[7] Salesforce, "Salesforce.com," http://www.salesforce.com/, 2012.

[8] M. Kriushanth, L. Arockiam, and G. J. Mirobi, "Auto scaling in cloud computing: An overview," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 7, 2013.

[9] S. Kitanov and D. Davcev, "Mobile cloud computing environment as a support for mobile learning," in *Cloud Computing 2012, The Third International Conference on cloud computing, GRIDs, and Virtualization*. Citeseer, 2012, pp. 99–105.

[10] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *Mechatronic Science, Electric Engineering and Computer (MEC), 2011 International Conference on*. IEEE, 2011, pp. 1143–1146.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[12] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[13] A. Ghanbari, Y. Vaghei, S. Noorani, and S. M. Reza, "Reinforcement learning in neural networks: a survey," *International Journal of Advanced Biological and Biomedical Research*, vol. 2, no. 5, pp. 1398–1416, 2014.

[14] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[15] R. Ramya, R. Anandanatarajan, R. Priya, and G. A. Selvan, "Applications of fuzzy logic and artificial neural network for solving real world problem," in *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*. IEEE, 2012, pp. 443–448.

[16] D. Nauck, "Neuro-fuzzy methods," *Fuzzy logic control: Advances in applications*, vol. 23, p. 65, 1999.

[17] N. K. Kasabov, *Foundations of neural networks, fuzzy systems, and knowledge engineering*. Marcel Alencar, 1996.

[18] C. Arun and K. Prabu, "Applications of mobile cloud computing: A survey," in *Intelligent Computing and Control Systems (ICICCS), 2017 International Conference on*. IEEE, 2017, pp. 1037–1041.

[19] P. Veerabhadram and P. Conradie, "Mobile cloud framework architecture for education institutions," in *Science and Information Conference (SAI), 2013*. IEEE, 2013, pp. 924–927.

[20] J. D. Lee and J.-H. Park, "Application for mobile cloud learning," in *2013 16th International Conference on Network-Based Information Systems*, 2013.

[21] P. Hazarika, V. Baliga, and S. Tolety, "The mobile-cloud computing (mcc) roadblocks," in *2014 Eleventh International Conference on Wireless and Optical Communications Networks (WOCN)*. IEEE, 2014, pp. 1–5.

[22] C. Chao and Z. Yue, "Research on the m-learning model based on the cloud computing," in *Chinese Automation Congress (CAC), 2013*. IEEE, 2013, pp. 806–811.

[23] S. Hettiarachchi, W. M. Spears, T. Kuremoto, M. Obayashi, and K. Kobayashi, "Adaptive swarm behavior acquisition by a neuro-fuzzy system and reinforcement learning algorithm," *International Journal of Intelligent Computing and Cybernetics*, vol. 2, no. 4, pp. 724–744, 2009.

[24] T. Kuremoto, M. Obayashi, K. Kobayashi, and S. Mabu, "How an adaptive learning rate benefits neuro-fuzzy reinforcement learning systems," in *International Conference in Swarm Intelligence*. Springer, 2014, pp. 324–331.

[25] S. M. H. Nabavi and S. Hajforoosh, "Exploration and exploitation tradeoff using fuzzy reinforcement learning," *International Journal of Computer Applications*, vol. 59, no. 5, 2012.

[26] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 2017, pp. 64–73.

[27] 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.

[28] M. Sharafi, M. R. Aalami, and S. A. Fakoorian, "An intelligent system for parking trailer using reinforcement learning and type 2 fuzzy logic," *Journal of Electrical and Control Engineering*, vol. 3, no. 5, 2013.

[29] K.-S. Hwang and W.-C. Jiang, "A shaped-q learning for multi-agents systems," in *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2024–2027.

[30] P. Jamshidi, A. M. Sharifloo, C. Pahl, A. Metzger, and G. Estrada, "Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution," in *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*. IEEE, 2015, pp. 208–211.