Engineering and Technology Journal e-ISSN: 2456-3358

Volume 10 Issue 05 May-2025, Page No.- 4662-4671

DOI: 10.47191/etj/v10i05.02, I.F. – 8.482

© 2025, ETJ

Design a Hierarchical Resource Allocating Approach by Using Dual Q-Learning in Deep Reinforcement Algorithm

Seyed Danial Alizadeh Javaheri[1], Reza Ghaemi[2], Hossein Monshizadeh Naeen[3]

[1,3]Department of Computer Engineering, Ne.C., Islamic Azad University, Neyshabur, Iran.

[2]Department of Computer Engineering, Qu.C., Islamic Azad University, Quchan, Iran.

ABSTRACT: With the advent of the Internet of Things (IoT), the number of devices that work in these environments has greatly increased, which has caused major changes in the structure of data processing and information storage. In this regard, cloud computing is an Internet-based computing platform that provides the necessary processing resources for users. However, due to the ever-increasing volume, speed and communication technologies, the current model of cloud processing can hardly provide satisfactory performance quality. So, proper management of the incoming requests is very critical for the continuous operation. In this project, in order to increase the speed of resource allocation operation, virtual machines are ranked by game theory, which significantly reduces the time and computational complexity of the whole process. The combination of these three approaches and their integration in sequence increases the speed of allocation performance and reduces the costs associated with it. In this article, two characteristics of activity completion time and service quality have been evaluated. According to the obtained results, applying the proposed structure on the Borg dataset provided by Google has yielded results in the range of more than 35% for reduction in CPU usage cost compared to other existing methods.

KEYWORDS: Hierarchical Approach, Dual Q-Learning, Deep Reinforcement Learning, Game Theory

INTRODUCTION

In order to achieve an optimal level for quality of service (QoS) in the IoT structure, the cloud computing model has been changed into “Fog processing” which is defined as follows [1]:

"Fog processing is a virtual platform that is able to provide processing services, storage and network functions between equipment’s located on the edge and traditional cloud centers. »

In fog space, facilities that can provide resources at the edge of the network are called fog nodes (FN). Indeed, tasks are distributed on the fog nodes at the edge of the network for increasing the efficiency. Therefore, it is necessary to use a series of buffers in the fog environment for this purpose. In this case, the buffers have a number of physical machines (PM) which themselves consist of a number of virtual machines (VM) and in this inter-layer structure they can respond to the corresponding requests in the IoT environment. By using the fog space, although the response span is improved, but the limitations in the processing resources is becoming serious challenge [2-4].

Fog Computing (FC) has emerged as a novel paradigm to address the aforementioned challenges. Its infrastructure consists of a large collection of heterogeneous and distributed devices such as routers, access points, cameras, vehicles, and smartphones which located in close proximity to end users. These devices are capable of communicating and cooperating with one another to perform storage and computational tasks requested by end users. The proximity of FC devices to end users enables time-sensitive applications to execute their operations with reduced communication traffic over core network links. In this architecture, devices that provide computational, storage, or networking capabilities are referred to as FNs. FNs can be deployed in dedicated environments where all of their resources are exclusively reserved for FC purposes, or in shared environments, where they perform their primary functions while also offering their remaining capacity to other platforms.

In real-time scheduling environments, researchers often need to handle a set of jobs comprising a variable number of parallel tasks. When these tasks are executed in parallel, they must share computational resources and collectively determine the final outcome of the entire job. For example, in Apache Hadoop systems based on cloud infrastructure, a program is decomposed into multiple mapping sub-tasks and dispatched to the cloud, which ultimately requires coordinated scheduling as a complete set. Similarly, distributed relational database queries, Monte Carlo simulations, and Blast searches are just a few examples of such workload patterns. In [5], a parallel task scheduling method based on deep reinforcement learning (DRL) is proposed to enhance resource utilization in computing platforms and to ensure QoS. This method is optimized using practical data. However, when DRL algorithms are directly

applied to multi-task allocation problems, the DRL agent tends to uniformly learn a single resource allocation strategy for all tasks, which may lead to suboptimal performance, especially when task diversity and scalability are not taken into account. Moreover, the size of the action space grows exponentially with the number of processing nodes and tasks, introducing significant complexity and posing additional challenges to efficient resource allocation.

In this project, game theory is utilized to enable the system to rapidly adopt an effective strategy for optimal resource allocation by learning from its past performance. In this context, DRL is employed for allocating heterogeneous tasks among users, leveraging its real-time adaptability and high flexibility. On the other hand, for tasks that require low latency, awareness of user location, and mobility support, it is essential to integrate fog computing with cloud infrastructure. This combination allows for efficient utilization of edge-layer network resources. The results demonstrate that the proposed approach leads to more than a 35% reduction in CPU usage cost compared to other existing methods.

The main contributions of this paper are as follows:

- Designing an innovative three-level hierarchical model in order to optimize the allocation of resources
- Applying a dual Q-learning approach in order to reduce time-spatial complexity and improve allocating rate
- Minimizing energy consumption based on using DRL and dual Q-learning algorithm

LITERATURE REVIEW

- **Optimal resource allocation**

In [6], an optimal task allocation problem is presented in which the objective is to maximize the number of user's requested tasks while considering deadlines. The authors have also proven that the optimal task allocation problem is NP-hard, and therefore, linear programming techniques and greedy algorithms must be employed to obtain a near-optimal solution. In [7], a Deep Q-Network (DQN) approach is used for resource allocation in mobile edge computing (MEC). The optimization system is designed to minimize energy cost, computational cost, and latency. However, in this study, task allocation is performed offline, and due to the high latency of the proposed method, it is not well-suited for real-time or online task allocation scenarios. In [8], fog computing is employed for task allocation. The key evaluation criteria include quality of service, bandwidth utilization, and latency reduction. A deep learning model is used to define the reward function for decision-making. The main challenge identified in this study is the assumption of task homogeneity. In [9], a deep reinforcement learning approach is utilized to find the optimal solution for resource allocation. The proposed method employs a double deep Q-network (DDQN) to model the problem as a path planning challenge. This method is location-dependent and sensitive to the timing of events, while also considering resource constraints in the objective of maximizing travel distance. Overall, one of the major challenges encountered during agent training using policy gradient algorithms is the risk of losing the optimization objective due to unstable agent performance. Recovering from such scenarios is particularly difficult, as the agent may begin to follow suboptimal trajectories and use them to update its policy, thereby reinforcing ineffective behaviors. In such situations, policy-based algorithms are no longer able to effectively utilize the available data, rendering these samples inefficient for training. To address this issue, various types of planning models have been proposed to simulate real-world conditions. Among them, task queue models have been introduced to describe the state of task generation and the temporary storage (buffer) in fog and cloud nodes [10–13].

Q-learning applications

In general, the RL algorithm provides an advanced solution for complex decision-making processes. The quality of the actions are usually evaluated by the amount of reward expected for future time steps. In this situation, value functions determine actions and allow them to select strategy according to their surrounding environment. There are many factors such as the certainty of actions or results, the ability to estimate the state after performing each action, the training of the agent by observer or only based on performing actions which can influence the choice of strategy. In other words, the main idea in the reinforcement learning method is to train the agent based on the changes that occur in the environment. One of the most common techniques used to determine the optimal policy in reinforcement learning methods is the use of Q-learning algorithm. This algorithm does not need a model and uses the value of Q (s_t,a_t) stored in the Q table to select the action. This algorithm has been used in various fields including advanced industrial processes, network control, game theory, robotics, operational search, control theory and image recognition. For example, in stochastic cooperative game theory, Q learning algorithm is used to maximize the profit of the whole system. By combining game theory and Q-learning, effective resource allocating can be implemented to improve the overall performance of the system. In addition, Q learning has been used to improve performance in games with a huge scale of information. In the following, a brief overview of the various functions of the Q learning algorithm is presented in Table 1. In fact, game theories are able to find a balance point between players who have opposite interests. On the other hand, due to the importance of fast processing and prioritization with proper delay, the two-player game theory is used, which has a high-performance speed and a suitable convergence rate. In [14], DQN approach has been used to allocate resources in edge computing. The optimized system is designed based on minimizing of energy, computational and delay cost. However, the allocation of activities is done offline and due

to the long time delay in the presented approach, it cannot be used well in online problems. In [15], the writers used fog space to allocate requested tasks. The investigated criteria included service quality, bandwidth and time delay reduction. In this regard, the deep learning model has been used to define the decision reward. One of the main drawbacks of this research is considering only the homogeneous tasks.

Table 1: Comparison of Q-learning algorithms applications and limitations

Ref	Applications	Limitations
[16]	A comprehensive review of reinforcement learning algorithms Using the Bellman equation to describe RL - offline/online methods to determine a policy	Without comparison of their performance in the stated problems
[17]	A comprehensive review of Q-learning algorithms - Classification of methods based on the behavior of agents	Without providing technical details of Q-learning algorithms - Lack of investigation multi-agents functions
[18]	Classification of multiple reinforcement learning with focus on deep Q-learning Evaluation of features, challenges and deep reinforcement learning algorithms	not providing a comprehensive mathematical background Lack of evaluation of introduced functions
[19]	Using the Q-learning algorithm in order to converge users' participation	lack of modeling of VM's movement in their dynamic conditions over time.

METHODOLOGY

Q-learning algorithm

As a model-independent algorithm, Q-learning algorithm is known as an efficient method for solving reinforcement learning problems. For the mathematical modeling of this algorithm, let π represent the policy. The function Q acting on the state-action pair to determine the cumulative reward of the agent through the Equ. (1). The complexity of implementing Equ. (1) grows exponentially with the increase in the size of the state space and operational space. In practical situations, due to the high number of action-state pairs, it is not possible to solve this challenge directly. So, one of the solutions to overcome this problem is to approximate Q values using a number of variables. In this situation, the reward function is defined according to the Equ. (2). The γ coefficient is a discount coefficient and helps to improve the convergence [20]. In this structure, approximation plays an important role [21-22].

$$Q^{\pi}(s_t, a) = (1-\alpha)Q(s_t, a) + \alpha.\left(r(s_t, a) + \gamma \max \acute{Q}\,(s_{t+1}, \acute{a}; \acute{\pi})\right) \quad (1)$$

$r(s_t, a)$: Reward of selecting action a in state s_t
γ : Discount factor
α : Learning rate

For this purpose, a special structure in the form of replay memory has been created for estimation through training in DRL. The purpose of using replay memory is to store a number of records, each record containing (s_t, a_t, r_t, s_{t+1}). In this case, a part of the memory is used as a estimator part. The mentioned system stores the record (s_t, a_t, r_t, s_{t+1}) for each step. During network learning, the same part of memory is extracted from the total available memory so that Q network learns from its past experiences. Due to the use of replay memory, enough training data is available to extract the Q value for the state-action pair (Equ. (3)). According to [21], parameter w is defined as a vector that is used for regression in deep neural network. In particular, the target network output is updated by rewarding and is used as an index for the performance evaluation (Equ. (4)). In this regard, a gradient descent algorithm is used to minimize the difference between the output of the target and estimation network (Equ. (5)) with updating the w vector.

$$R_t = r_{(t+1)} + \gamma r_{(t+2)} + \gamma^2 r_{(t+3)} + \cdots + \gamma^T r_{(t+T+1)} = \sum_{k=0}^{T} \gamma^k r_{(t+k+1)} \quad (2)$$

$$Q^{\pi}(s_t, a, w) \approx Q^{*}(s_t, a) \quad (3)$$

$$Q^{lab}(s_t, a) = r(s_t, a) + \gamma.\max Q^{tar}(s_{t+1}, \acute{a}, \acute{w}) \quad (4)$$

$$Loss = \left(Q^{lab}(s_t, a) - Q^{pre}(s_t, a, w)\right)^2 \quad (5)$$

Dual Q-learning algorithm

In this section, the Double Q-learning algorithm is extended to handle non-deterministic states. In the non-deterministic case, it is first necessary to acknowledge that the outputs are no longer deterministic. Under these conditions, Equ. (6) is redefined for policy π as an expected discounted cumulative reward formulation. The optimal policy π* is the one for which the value function $V^{\pi}(s)$ is maximized for all states s.

Subsequently, the Q-function is modified according to Equ. (7), where $P(\acute{s}|s,a)$ denotes the probability of transitioning to state ś after taking action a in state s. Under these conditions, the Q-function is further refined as shown in Equ. (8). In summary, the revised definition of Q(s,a) is the mathematical expectation of the previously defined value under uncertainty. At this point, it becomes necessary to rewrite the learning rule. Since the previous learning rule does not converge in this context, the new rule must be adjusted accordingly. In the proposed approach, rather than discarding the previous estimate, a weighted average between the previous and new estimates is calculated.

According to Equ. (9), the necessary conditions for the convergence of the Q-function are established. In this equation, s and a represent the state and action occurring at the n-th iteration, and $visits_n(s, a)$ denotes the total number of state–action pairs that have been visited up to iteration n. A key aspect of this update rule is that it results in slower changes compared to the previous rule. By reducing the value of α, the $\hat{Q}(s, a)$ average term is updated more gradually. Consequently, the rate of learning progression decreases over time, and with an appropriate learning rate, convergence of the Q-function can be achieved. In non-deterministic Markov processes, it is assumed that the Q-learning agent operates within a non-deterministic Markov Decision Process (MDP) environment. For the reward values, an upper bound is defined according to Equ. (11).

The Q-learning agent attempts to estimate the Q-function using an arbitrary but bounded initial value. If n denotes an iteration index, it indicates that action a has been executed for the i^{th} episode in state s, under conditions defined by Equa. (12) and Equ. (13). Although it has been theoretically proven that Q-learning and other reinforcement learning algorithms converge under certain conditions, in practice, several thousand iterations of the main loop are often required to reach a satisfactory level of convergence. For instance, in the TD-Gammon game, there are approximately 1.5 million distinct games, each involving dozens of unique state–action pairs.

$$V^{\pi}(s_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right] \tag{6}$$

$$Q(s,a) = E\left[r(s,a) + \gamma V^*(\delta(s,a))\right] = E\left[r(s,a)\right] + \gamma E[V^*(\delta(s,a))] \tag{7}$$

$$= E\left[r(s,a)] + \gamma \sum_{\acute{s}} P(\acute{s}|s,a)\, V^*(\acute{s})\right]$$

$$Q(s,a) = E\left[r(s,a)] + \gamma \sum_{\acute{s}} P(\acute{s}|s,a)\, \max Q(\acute{s},\acute{a})\right] \tag{8}$$

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max \hat{Q}_{n-1}(\acute{s},\acute{a})] \tag{9}$$

$$\alpha_n = \frac{1}{1 + visits_n(s,a)} \tag{10}$$

$$|r(s,a)| < c \tag{11}$$

$$\sum_{i=1}^{\infty} \alpha_{n(i,s,t)} = \infty \tag{12}$$

$$\sum_{i=1}^{\infty} \left[\alpha_{n(i,s,t)}\right]^2 = \infty \tag{13}$$

Deep reinforcement learning

Deep learning, as a subset of machine learning, is a suitable choice for online learning due to its capability to represent complex intrinsic relationships between system inputs and outputs. Therefore, it can be concluded that deep reinforcement learning is a promising approach for autonomous and real-time decision-making in the task scheduling problem of IoT applications within a fog computing environment. Reinforcement Learning (RL) problems are typically defined with a single objective, which is to maximize the cumulative rewards received by the agent. The agent aims to optimize this objective function by selecting appropriate actions, using reward signals to reinforce its beneficial behaviors. In general, the RL algorithm offers an enhanced solution for managing complex decision-making processes. The core idea of RL is to train a specific agent to adapt to dynamic changes occurring within its environment. Under these conditions, the Q-learning algorithm is employed to learn an optimal planned policy by taking future decisions into account and evaluating the feedback received from the cloud environment. Let us assume that $E = \{e_1, e_2, \dots, e_n\}$ represents the set of tasks requested by a number of users, and $V = \{v_1, v_2, \dots, v_m\}$ denotes the set of available VMs. Under this setting, the probabilistic relationships described by Equ. (14) to Equ. (15) can be defined accordingly.

$$P(\acute{s}|s,a) = P[s_{t+1} = \acute{s}|s_t = s, a_t = a] \tag{14}$$

$$\sum_{\acute{s}\epsilon S} P(\acute{s}|s,a) = 1 \quad (15)$$

s_t : State of cloud-programmer at time step t in state S
a_t : Selected action of action space A at time step t

The policy planner $\pi(a|s)$, which is responsible for mapping states to actions, assigns each task to a VM. The immediate reward of such an action is calculated as r_t. The goal of the cloud scheduler is to identify an optimal policy that minimizes the cumulative reward. In the proposed optimization-based scheduling model, the Q-learning method is utilized to evaluate the feedback obtained from the cloud environment in order to optimize future decision-making. Under this framework, after collecting all rewards, the average Q-value for each reward in every state is computed, and the optimal value is derived using Equation (4-2).

$$\xi_{ij} = (\psi_{ij} + \varphi_{ij}) \times P_j \quad (16)$$

$$Q^*(s,a) = \min \; Q_\pi(s,a) \quad (17)$$

ξ_{ij} : Operation cost
ψ_{ij} : Operation time
φ_{ij} : Waiting time for assigning
P_j : Cost per unit of j^{th} VM

The optimal value function is typically computed using the Bellman optimization equation, as described in Equ. (18). In this context, the resources allocated to each task are predefined, and for each task assignment, the conditions specified in Equ. (19) to Equ. (22) must be satisfied. Subsequently, the cloud scheduler evaluates the Q-value corresponding to the current policy. The policy is then updated according to Equ. (23). Ultimately, the primary objective of this model is to determine an optimal policy that minimizes the reward for each state, as defined by Equ. (24).

$$Q^*(s,a) = \sum_{\acute{s}} \Upsilon(\acute{s}|s,a)[r + \gamma_{min}.Q^*(\acute{s},\acute{a})] \quad (18)$$

$$k_i^{CPU} \le CPU_j^t \quad (19)$$

$$k_i^{RAM} \le RAM_j^t \quad (20)$$

$$k_i^{BW} \le BW_j^t \quad (21)$$

$$k_i^{DS} \le DS_j^t \quad (22)$$

$$\acute{\pi} = \arg\min Q_\pi(s,a) \quad (23)$$

$$\text{Min } E[Q_{\pi^*}(s,a)] \qquad \forall s \in S \quad (24)$$

k_i^{CPU} : Required CPU
k_i^{RAM} : Required RAM
k_i^{BW} : Required Bandwidth
k_i^{DS} : Required Memory

Objective Function

In the Deep Q-Learning method, a neural network is used to represent the Q-function, denoted as Q(s,a;θ), where θ represents the weights of the neural network. The Q-network is trained during each episode by updating these parameters to approximate the Q-values. Although neural networks offer considerable flexibility, it is essential to ensure the stability of Q-learning. To address this, a deep neural network can be employed in DQNs instead of a simple Q-function approximation, thereby enhancing both efficiency and stability. Furthermore, the gradient descent algorithm can be used to minimize the loss between the target network's output and the predicted Q-values (Equ. 25), which in turn updates the weights w. The main objective is to achieve maximum coverage with minimal monitoring cost. In fact, despite data redundancy in monitoring, higher accuracy in data acquisition can be achieved. However, in applications with lower coverage requirements, energy consumption can be reduced by decreasing the activation rate of monitoring nodes. In this way, a trade-off between data quality and energy consumption is implemented. On the one hand, node participation must be managed in a way that minimizes energy consumption. Following a pricing mechanism based on the degree of participation, the final solution is determined using reinforcement learning, aiming to maximize coverage while minimizing users' energy consumption.

$$Loss = \left(Q^{lab}(s,a) - Q^{pre}(s,a,w)\right)^2 \quad (25)$$

Dataset

In this paper, the data set obtained from the tracking of requested tasks on Google's clusters is used, which was collected in a period of 5 hr and 15 min [23]. Each request that is defined as a job in the database consists of a number of tasks, and the following characteristics are specified for each task:

- Time: Max. execution time (sec)
- JobID: A unique code for each specific job
- TaskID: unique code for each specific task
- JobType: unique code to define the type of requested job
- Normalized Task Core: the average no. of used CPUs
- Normalized Task Memory: avg. memory used for each individual task

In this model, the dataset is anonymized through various methods. Job and task names are replaced with numeric identifiers, and all timestamps are adjusted relative to an anonymous reference point. Temporal information is reported in 5-minute intervals. Memory usage and CPU core utilization are normalized using an unknown linear transformation. Moreover, no semantic information is provided regarding the types of specified jobs. The trace covers 75 time intervals (5-minute), comprising a total of 3,535,029 observations, 9,218 unique jobs, and 176,580 distinct tasks defined in the system. Table (2) shows the specifications of 4 different types of work. It is worth noting that the categories labeled as CPU exhibit a higher memory-to-core ratio compared to those labeled as Mem. Table (3) shows the correlation coefficient between types of work for this collection. The value of coefficient changes varies from 1 (strong correlation) and 0 (no correlation) to -1 (strong anti-correlation). Identifying such correlations is important, as they reflect the type of Job, which can vary depending on user requirements and operational conditions. In this situation, high correlation coefficients show the number of related tasks.

Table 2: Characteristics of 4 different types of requested Jobs [24]

Cluster description	Active Long CPU	Active Long CPU Few tasks	Active Long Mem	Active Short CPU	Active Very Short Mem	Inactive Short	Inactive Long
Type 0	37	93	6	72	1260	318	293
Type 1	36	104	0	12	2280	531	203
Type 2	90	237	28	692	1276	519	398
Type 3	21	142	10	1	2	7	196

Table 3: Correlation coefficient between different types of jobs [24]

Cluster description	Active Long CPU	Active Long CPU Few tasks	Active Long Mem	Active Short CPU	Active Very Short Mem	Inactive Short	Inactive Long
Type 0	-0.02	-0.06	-0.03	0.26	0.01	-0.04	0.04
Type 1	-0.05	-0.11	-0.06	-0.09	0.33	0.22	-0.22
Type 2	0.04	0.05	0.05	-0.14	-0.27	0.01	-0.01
Type 3	0.10	0.40	0.11	-0.04	-0.24	-0.32	0.32

Results evaluation

In the implementation of the game theory, it is necessary to determine the strategies of the players. In this regard, DRL has been used in order to determine the appropriate type of strategy using the history of activities. To implement this approach, the agents are well trained so that they can adopt the right policy for any situation. The training stage of choosing the optimal policy in the proposed method is done by episodes. In each episode, a variable number of virtual machines (VM) in the range between [50-350] and a fixed number of task requesters (TR) in the range between [400-1600] are considered. The resource allocation is done based on the selected policy and continue until all the considered tasks are allocated on the proper VMs. During training, the exploration has been done through a fixed number of episodes, which improves the strategy generation process in the next time steps. In other words, a fixed number of episodes (100) were simulated for each request in order to explore the action space under the current policy and utilize the resulting data to improve the overall activity policy. In other words, the state, action, and reward information for each episode was stored and used to calculate the cumulative reward per episode. A total of 1000 iterations were simulated, after which the average reward was computed. The minimum reward, corresponding to the lowest cost, was considered equivalent to full accuracy, and other accuracy values were

subsequently derived. To evaluate the effectiveness of the proposed method, 20% of the dataset was used, during which the agent followed its learned policy by selecting the action with the lowest reward.

It is worth noting that in some cases, values for resource utilization and accuracy metrics were also obtained during the simulations. Additionally, for all simulations (except for the sensitivity analysis) λ was set to 10^{-5}. The value of the λ parameter influences the estimation behavior: the closer λ is to 1, the algorithm tends to approximate the average of all models; conversely, as λ approaches 0, a single model can dominate the others, producing a final result that closely aligns with the predictions of that specific model. It should be noted that In practice, the optimal value can be determined based on historical data, and subsequently, the value of λ can be adjusted by analyzing the model error observed in the system's past records. Figure 1 illustrates the minimum, average, and maximum values of the effective error over a specified episode.

Figure 1. Min., Mean and Max. values of the effective error (for 1 episode)

The red curve represents the performance of the standard Q-learning algorithm. The green curve corresponds to the results obtained using the Deep Reinforcement Learning (DRL) algorithm, while the blue curve shows the results derived from the application of the Double Q-learning algorithm integrated with DRL. Figure 2 presents the results over the entire observation period (6 hours and 15 minutes). For comparison, the minimum, average, and maximum effective error values predicted by the DRL method for CPU consumption are 1448.4 W, 195.3 W, and 318.9 W, respectively. In contrast, the corresponding values for the standard Q-learning method are estimated at 173.1 W, 259.4 W, and 357.5 W, respectively.

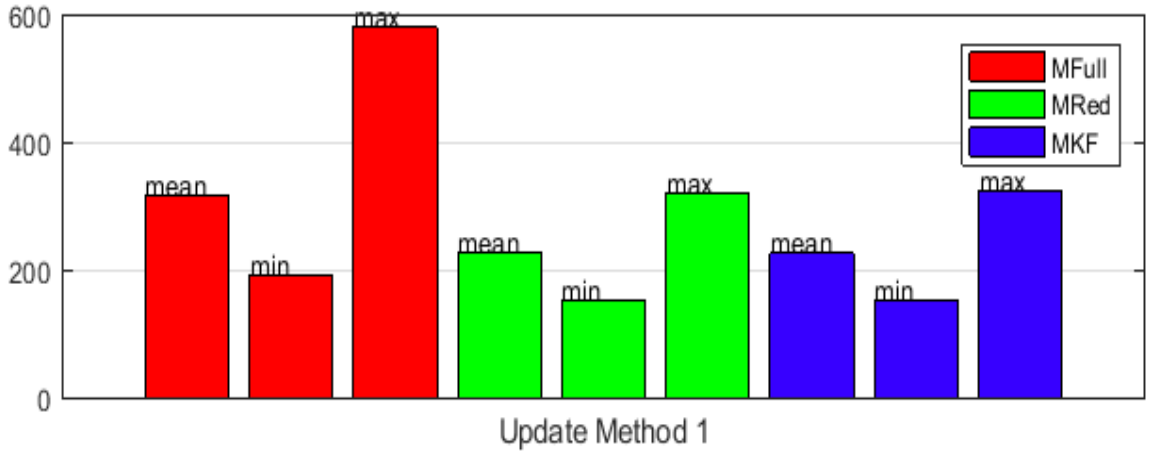

Figure 2. Min., Mean and Max. values of the effective error (for total episodes)

The sensitivity of the proposed model to various parameters is evaluated in the following section. To this end, the sensitivity to γ is first analyzed, followed by an evaluation of sensitivity to the λ parameter. The proposed method is applied over the entire observation period, and the values of the two parameters under investigation are systematically varied to assess their impact on the results. Specifically, γ is varied from 0 to 0.9 in increments of 0.1, while λ is varied from 10^{-7} to 10^{-3}, with each step obtained by multiplying the previous value by 10. The DRL method is executed for each

combination of these two parameters using the double Q-learning algorithm. When γ values are close to zero, the effective error tends to be high, as the DRL method in this case has limited influence in improving the results when high prediction errors are present. In other words, a γ value close to zero prevents the DRL method from adequately adjusting its behavior in subsequent time steps to reduce future errors. When γ is close to zero, the effective error gradually decreases with increasing λ, as this change facilitates faster adaptation of the weight coefficients within the algorithm. However, it should be noted that at high λ values (e.g., 10^{-3}), the model weights tend to become oscillatory. Additionally, when γ is high (e.g., 0.9) and λ is low, the effective error significantly increases. Because under such conditions, the DRL method performs predictions over very short time steps, and the weight coefficients adjust very slowly. Conversely, increasing λ in the high γ leads to a reduction in the effective error. Nonetheless, the weight coefficients may still undergo substantial fluctuations due to discrepancies between the predicted and actual system behavior. The γ and λ values used in the previous simulation section do not necessarily yield the absolute minimum effective error. However, they offer a good balance by maintaining a low error while ensuring that the rate of change in weight coefficients remains moderate. Subsequently, the results obtained using the DRL method are compared with those from the standard Q-learning algorithm. In Figure 3, the performance of standard Q-learning over the entire observation period is presented, followed by Figure 4, which illustrates the outcomes of applying the combined DRL and double Q-learning approach.

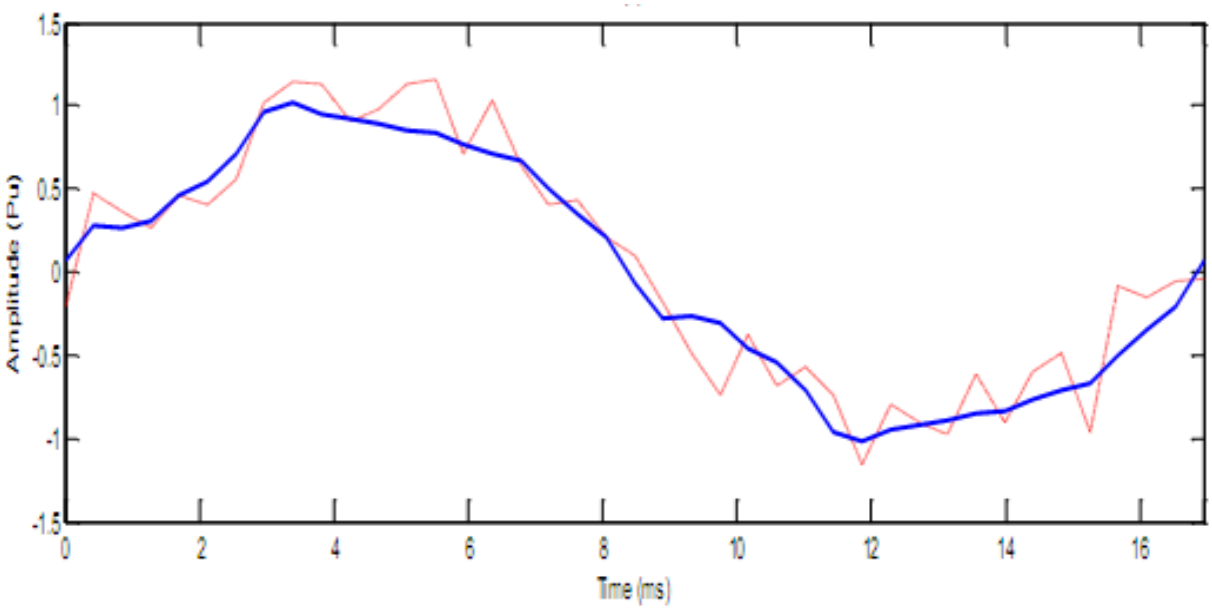

Figure 3. Tracking the reference weighting signal (simple Q-learning)

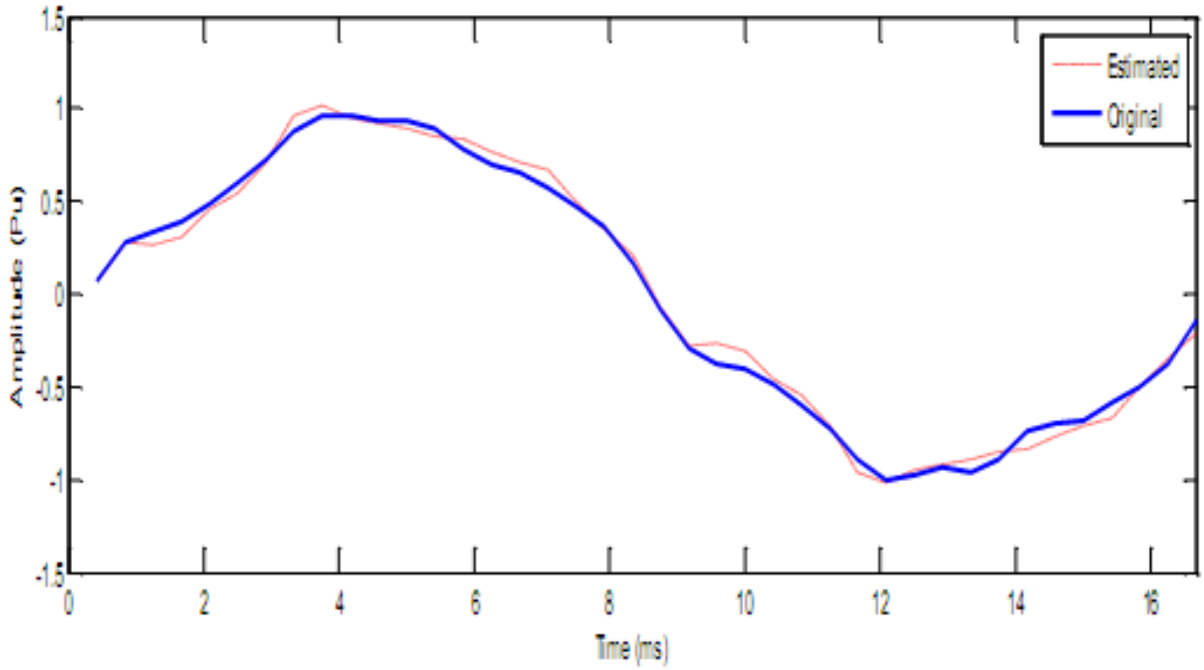

Figure 4. Tracking the reference weighting signal (double Q-learning + DRL)

CONCLUSION

One of the essential requirements of all computing systems, which also contributes to improved network performance, is efficient resource management and task scheduling. The primary objective of this study is to reduce the average service delay for IoT applications in a cloud-fog computing environment. For this target, the study proposes a novel task allocation method by integrating deep reinforcement learning (DRL) with the double Q-learning algorithm within a fog-based IoT framework. Due to the dynamic nature of modern networks and the complexity of accurately modeling them, solving the scheduling problem requires an online and adaptive approach. The proposed method is designed to autonomously develop an effective scheduling strategy over time, based on prior experience. In the proposed approach, deep learning is utilized to estimate the level of user participation in the task allocation process. Based on these estimations, a pricing mechanism is applied, and the final solution is derived using reinforcement learning techniques that aim to maximize coverage and minimize the energy consumption of participating users. According to the obtained results, applying the proposed structure on the Borg dataset provided by Google has yielded results in the range of more than 90% for the optimal allocation of tasks to the considered virtual machines, which can be considered as an indicator to prove the effectiveness of the proposed approach.

REFERENCES

1. Bonomi, Flavio, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things." In Proceedings of the first edition of the MCC workshop on Mobile cloud computing, pp. 13-16. 2012.
2. Osanaiye, Opeyemi, Shuo Chen, Zheng Yan, Rongxing Lu, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. "From cloud to fog computing: A review and a conceptual live VM migration framework." IEEE Access 5 (2017): 8284-8300.
3. Yi, Shanhe, Zijiang Hao, Zhengrui Qin, and Qun Li. "Fog computing: Platform and applications." In 2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb), pp. 73-78. IEEE, 2015.
4. Chiang, Mung, and Tao Zhang. "Fog and IoT: An overview of research opportunities." IEEE Internet of things journal 3, no. 6 (2016): 854-864.
5. Zhang, Lingxin, Qi Qi, Jingyu Wang, Haifeng Sun, and Jianxin Liao. "Multi-task deep reinforcement learning for scalable parallel task scheduling." In 2019 IEEE International Conference on Big Data (Big Data), pp. 2992-3001. IEEE, 2019.
6. Cheng, Mingxi, Ji Li, and Shahin Nazarian. "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers." In 2018 23rd Asia and South pacific design automation conference (ASP-DAC), pp. 129-134. IEEE, 2018.
7. Ali, Tariq, Umar Draz, Sana Yasin, Javeria Noureen, Ahmad Shaf, and Munwar Ali. "An Efficient Participant's Selection Algorithm for Crowdsensing." Int. J. Adv. Comput. Sci. Appl 9 (2018): 399-404.
8. Graesser, Laura, and Wah Loon Keng. Foundations of deep reinforcement learning: theory and practice in Python. Addison-Wesley Professional, 2019.
9. Kumar, Neetesh, Syed Shameerur Rahman, and Navin Dhakad. "Fuzzy inference enabled deep reinforcement learning-based traffic light control for intelligent transportation system." IEEE Transactions on Intelligent Transportation Systems (2020).
10. Nan, Yucen, Wei Li, Wei Bao, Flavia C. Delicato, Paulo F. Pires, and Albert Y. Zomaya. "A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems." Journal of Parallel and Distributed Computing 112 (2018): 53-66.
11. Li, He, Kaoru Ota, and Mianxiong Dong. "Deep reinforcement scheduling for mobile crowdsensing in fog computing." ACM Transactions on Internet Technology (TOIT) 19, no. 2 (2019): 1-18.
12. Huang, Liang, Xu Feng, Cheng Zhang, Liping Qian, and Yuan Wu. "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing." Digital Communications and Networks 5, no. 1 (2019): 10-17.
13. Zhai, Junyong. "Dynamic output-feedback control for nonlinear time-delay systems and applications to chemical reactor systems." IEEE Transactions on Circuits and Systems II: Express Briefs 66, no. 11 (2019): 1845-1849.
14. Huang, Liang, Xu Feng, Cheng Zhang, Liping Qian, and Yuan Wu. "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing." Digital Communications and Networks 5, no. 1 (2019): 10-17.
15. Graesser, Laura, and Wah Loon Keng. Foundations of deep reinforcement learning: theory and practice in Python. Addison-Wesley Professional, 2019.
16. Liu, Yan, Bin Guo, Yang Wang, Wenle Wu, Zhiwen Yu, and Daqing Zhang. "TaskMe: Multi-task allocation in mobile crowd sensing." In Proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing, pp. 403-414. 2016.
17. Chen, Jianwei, Huadong Ma, Dong Zhao, and David SL Wei. "Participant density-independent location privacy protection for data aggregation in mobile

crowd-sensing." Wireless Personal Communications 98 (2018): 699-723.

18. Buhussain, A. A., R. E. D. Grande, and A. Boukerche. "Performance analysis of Bio-Inspired scheduling algorithms for cloud." In IEEE International parallel and distributed processing symposium workshops, pp. 776-785. 2016.
19. Zhang, Jialin, Xianxian Li, Zhenkui Shi, and Cong Zhu. "A reputation-based and privacy-preserving incentive scheme for mobile crowd sensing: a deep reinforcement learning approach." Wireless Networks (2022): 1-14.
20. Yu, Yan, Qian Shi, and Hak-Ke\ung Lam. "Fuzzy sliding mode control of a continuum manipulator." In 2018 IEEE International Conference on Robotics and Biomimetics (ROBIO), pp. 2057-2062. IEEE, 2018.
21. Nair, Arun, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam et al. "Massively parallel methods for deep reinforcement learning." arXiv preprint arXiv:1507.04296(2015).
22. Liu, Ruishan, and James Zou. "The effects of memory replay in reinforcement learning." In 2018 56th annual allerton conference on communication, control, and computing (Allerton), pp. 478-485. IEEE, 2018.
23. Chen, Yanpei, Archana Sulochana Ganapathi, Rean Griffith, and Randy H. Katz. "Analysis and lessons from a publicly available google cluster trace." EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95 94 (2010).
24. Kiumarsi, Bahare, Kyriakos G. Vamvoudakis, Hamidreza Modares, and Frank L. Lewis. "Optimal and autonomous control using reinforcement learning: A survey." IEEE transactions on neural networks and learning systems 29, no. 6 (2017): 2042-2062.