Task Offloading in Edge-Cloud Computing Using a Q-Learning Algorithm

Somayeh Abdi*, Mohammad Ashjaei and Saad Mubeen
Department of Networked and Embedded Systems, Mälardalen University, Västerås, Sweden

Keywords: Task Offloading, Edge-Cloud Computing Continuum, Reinforcement Learning, q-Learning Algorithm.

Abstract: Task offloading is a prominent problem in edge−cloud computing, as it aims to utilize the limited capacity of fog servers and cloud resources to satisfy the QoS requirements of tasks, such as meeting their deadlines. This paper formulates the task offloading problem as a nonlinear mathematical programming model to maximize the number of independent IoT tasks that meet their deadlines and to minimize the deadline violation time of tasks that cannot meet their deadlines. This paper proposes two Q-learning algorithms to solve the formulated problem. The performance of the proposed algorithms is experimentally evaluated with respect to several algorithms. The evaluation results demonstrate that the proposed Q-learning algorithms perform well in meeting task deadlines and reducing the total deadline violation time.

1 INTRODUCTION

As the use of Internet of Things (IoT) devices proliferates, the need to move processing, storage, and decision-making closer to the data source becomes increasingly important. The edge-cloud computing paradigm addresses this critical need by integrating fog and cloud servers to process requests from edge devices (Aazam et al., 2018). This computing continuum provides distributed computing resources by locating fog servers close to the data source to reduce network congestion and latency. However, the computing capacity of fog servers is limited. Therefore, it also leverages resources provided by a centralized cloud to offer scalable resources.

Providing such distributed and scalable resources is a critical requirement for various systems such as traffic management systems, smart cities, smart healthcare, and industrial IoT. In these systems, as the level of autonomy increases, more and more data- and compute-intensive tasks need to be performed, and edge devices alone are not sufficient to meet the demands of these tasks. Hence, task offloading becomes a prominent challenge in edge-cloud computing as it aims at utilizing the limited capacity of fog servers and satisfying the QoS requirements of all tasks.

Task offloading refers to selecting appropriate resources in the fog or cloud layer to execute the tasks requested by the edge devices. The limited capacity of fog servers requires that delay-sensitive tasks be processed in the fog layer, while compute-intensive tasks are offloaded to the cloud layer. Meeting the QoS requirements of requests and utilizing the limited capacity of fog servers is significantly impacted by such task offloading. To solve a task-offloading problem, different problem-solving techniques such as mathematical programming, machine learning, heuristics, meta-heuristics, and game theory can be applied (Abdi et al., 2022).

In this paper, we formulate the task offloading problem in edge-cloud computing as a non-linear mathematical programming model. Then, for the formulated problem, we propose a Q-learning algorithm which is a popular reinforcement learning technique. Reinforcement learning algorithms can be used to learn effective scheduling policies to meet specific performance objectives while satisfying problem constraints (Watkins and Dayan, 1992). This paper focuses on deadline-constrained task offloading in edge-cloud. Considering a set of independent tasks, the objective is to maximize the number of tasks that meet their deadlines and minimize the deadline violation time of the tasks that cannot meet their deadlines. The main contributions in this paper are as follows:

- Formulate the task offloading problem in edge-cloud computing as a non-linear mathematical programming model. The objective is to maximize the number of tasks that meet their deadlines and minimize the deadline violation time.

Abdi, S., Ashjaei, M. and Mubeen, S.
Task Offloading in Edge-Cloud Computing Using a Q-Learning Algorithm.
DOI: 10.5220/0012590800003711

In *Proceedings of the 14th International Conference on Cloud Computing and Services Science (CLOSER 2024)*, pages 159-166
ISBN: 978-989-758-701-6; ISSN: 2184-5042

- Propose two Q-learning algorithms, called TOQL and PTOQL, to efficiently assign the IoT tasks to available resources in fog and cloud servers to solve the proposed mathematical model.
- Conduct extensive experiments to configure the hyper-parameters of the proposed Q-learning algorithms and to evaluate their performance in comparison with other algorithms.

2 RELATED WORKS

The problem of task offloading in edge-cloud computing has been studied under various optimization objectives, including energy efficiency, completion time (Mohammadi et al., 2023), and financial cost (Islam et al., 2021).

The work in (Lou et al., 2023) formulates the workflow scheduling in edge-cloud computing with the objective of makespan optimization and proposes a heuristic algorithm to find near-optimal solutions for the scheduling problem. The work in (Saeed et al., 2023) proposes a genetic algorithm that minimizes the energy consumption and makespan of the workflows that are generated by IoT devices. The work in (You et al., 2016) proposes a mathematical programming model to minimize mobile energy consumption under latency constraints. The work in (Misra and Saha, 2019) formulates the problem of task offloading in fog computing as an integer-linear programming (ILP) model and proposes a greedy-heuristic-based approach to efficiently solve the problem. The proposed algorithm, called Detour, makes optimal decisions on the local or remote execution of tasks and selects the optimal fog nodes for tasks. It also selects the optimal path for offloading tasks.

The work in (Nguyen et al., 2019) proposes a particle swarm optimization algorithm that minimizes the cost of executing Bag-of-Tasks (BoT) applications in fog-cloud computing. The work in (Nan et al., 2022) proposes a genetic algorithm for task offloading in edge-cloud computing to minimize energy consumption. The work in (Smys and Ranganathan, 2020) proposes a game theory-based scheduling algorithm to minimize the waiting time of tasks. The work in (Jararweh et al., 2018) uses Logistic regression, a supervised learning algorithm, to balance the load of edge nodes and enhance a dynamic resource allocation strategy. The work in (Bouet and Conan, 2018) uses a clustering approach, an unsupervised learning algorithm, to group resources based on the distance between edge nodes to minimize the response delay.

Some works have proposed reinforcement learning algorithms for task offloading in edge-cloud computing (Hortelano et al., 2023), (Hortelano et al., 2023). The work in (Xue et al., 2022) proposes a hybrid algorithm of deep reinforcement learning and a genetic algorithm to minimize the execution time of workflows. The work in (Wang et al., 2022) proposes a task scheduling method based on reinforcement learning to minimize the completion time of parallel tasks. It also applies digital twins to simulate the results of different actions made by an agent. The work in (Razaq et al., 2022) proposes a reinforcement learning algorithm for task scheduling in fog-cloud computing to satisfy the required latency and security requirements.

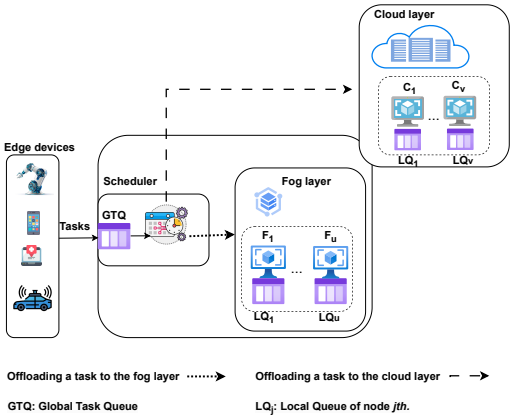

Figure 1: Three-tier architecture used in the system model.

However, these works do not consider the waiting time of tasks when some tasks are assigned to the same fog node. This paper proposes a Q-learning algorithm to maximize the number of IoT tasks that meet their deadline considering the waiting time of tasks when some tasks are assigned to the same node. This consideration leads to load balancing among available fog nodes to decrease the total waiting time for tasks. Moreover, it minimizes the violation time of the tasks that cannot meet their deadline.

3 SYSTEM MODEL

The proposed edge-cloud computing continuum is shown in Figure 1, which is a three-tier architecture consisting of edge devices, a fog layer, and a cloud layer. We assume that edge devices, such as healthcare devices, request some tasks to be executed in the higher computing layers. The requests from different edge devices are submitted to a scheduler located in the fog layer and added to the global task queue in this component. The scheduler decides on the computing layers and nodes where the tasks will be executed.

The fog layer is closer to the source of gen-

erated tasks and it provides a set of fog nodes $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, .., \mathcal{F}_u\}$. The cloud layer provides scalable computing resources in comparison to the fog layer which provides limited computing resources. The cloud layer provides a set of cloud nodes $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, .., \mathcal{C}_v\}$. Generally, we assume that the fog-cloud computing continuum provides computing resources $\mathcal{N} = \{\mathcal{F} \cup \mathcal{C}\}$. We assume that m computing layers provide computing nodes, and $k \in \{1, .., m\}$ is used as the index of computing layers. The notation N_k denotes the set of computing nodes provided by kth computing layer. The notation N_{kj} denotes jth node in the kth computing layer. The tuple $N_{kj} :< pf_{kj}, bw_{kj} >$ describes the characteristics of node N_{kj}, where the notation pf_{kj} indicates the CPU processing capacity of a node N_{kj}, i.e., the number of instructions that the node can process per second (MIPS). The communication bandwidth of node N_{kj} is denoted by bw_{kj}. Moreover, each node has a local queue, where tasks assigned to the node are added to it and processed based on the First Comes First Serves (FCFS) policy.

We assume that a set of independent IoT tasks $T = \{\tau_1, .., \tau_n\}$ are submitted to the scheduler and added to the global task queue in the scheduler. The tuple $\tau_i :< cs_i, id_i, od_i, d_i, p_i >$ describes the characteristics of task τ_i, where the notation cs_i indicates the computation size of task τ_i in Million Instructions (MI). The notations id_i and od_i are the input and output data size of task τ_i, respectively. The notations d_i and p_i indicate the deadline and priority of task τ_i, respectively. The tasks are assigned priorities according to their importance or criticality.

4 PROBLEM FORMULATION

We formulate the problem of task offloading in a heterogeneous fog-cloud computing system as a mathematical programming model. We assume that a set of tasks T is submitted to the scheduler and it assigns the tasks to the nodes in the fog or cloud layer to execute them. To formulate task assignment, we define decision variable x_{ikj}; $x_{ikj} = 1$ if and only if task τ_i is assigned to node N_{kj}; otherwise $x_{ikj} = 0$. This assignment is formulated in Eq. (1) and ensures that each task is assigned to a node in the fog or cloud layer.

$$\sum_{k=1}^{m} \sum_{\forall j \in N_k} x_{ikj} = 1 \quad \forall i \in T \tag{1}$$

We consider non-preemptive resource allocation. It means that once a task starts its execution on a node, it will finish without any interruption. If task τ_i is assigned to node N_{kj}, then $x_{ikj} = 1$ and its execution time is determined as follows:

$$\xi_{ikj} >= x_{ikj} \cdot (cs_i / pf_{kj}) \tag{2}$$
$$\forall i \in T,\ \forall k \in \{1, .., m\},\ \forall j \in N_k$$

Moreover, we assume that each node can process only one task at a time and assigned tasks to a node are executed sequentially. Therefore, when task τ_i is assigned to node N_{kj}, it is added to the queue of the node and waits until the node processes it. Suppose the set T'_{kj} indicates the tasks assigned to node N_{kj} before task τ_i is assigned to node N_{kj}, then the waiting time of task τ_i is formulated as follows:

$$w_{ikj} >= m_{tkj} \cdot x_{ikj} \tag{3}$$
$$\forall i \in T,\ \forall t \in T'_{kj},\ \forall k \in \{1, .., m\},\ \forall j \in N_k$$

Hence, the completion time of a task includes the waiting time of the task, the execution time of the task, and the input and output data transfer time. Eqs. (4) and (5) formulate the data transfer time for input and output data of task τ_i to/from node N_{kj}, respectively.

$$\beta_{ikj} >= x_{ikj} \cdot (id_i) / bw_{kj} \tag{4}$$
$$\zeta_{ikj} >= x_{ikj} \cdot (od_i) / bw_{kj} \tag{5}$$
$$\forall i \in T,\ \forall k \in \{1, .., m\},\ \forall j \in N_k$$

In this work, we consider the overlap between the waiting time of a task and the input data transfer time. Therefore, the completion time of task τ_i is formulated as follows:

$$m_{ikj} >= x_{ikj} \cdot (\xi_{ikj} + \max\ (\beta_{ikj}, w_{ikj}) + \zeta_{ikj}) \tag{6}$$
$$\forall i \in T,\ \forall k \in \{1, .., m\},\ \forall j \in N_k$$

The objective function of the proposed mathematical model is to maximize the number of tasks that meet their deadlines and minimize the deadline violation times of tasks that cannot meet their deadlines. Therefore, we define the decision variable δ_i to identify whether the deadline of the task τ_i is met. Eq. (7) identifies the value of this decision variable. If the deadline of task τ_i is met then $\delta_i = 1$; otherwise $\delta_i = 0$. To define the deadline violation time of the task τ_i, we define the decision variable ϑ_i in Eq. (8).

$$m_{ikj} \cdot \delta_i \leq d_i \tag{7}$$
$$\vartheta_i >= (1 - \delta_i) \cdot (m_{ikj} - d_i) \tag{8}$$
$$\forall i \in T,\ \forall k \in \{1, .., m\},\ \forall j \in N_k$$

If the deadline of the task τ_i is met, then $\delta_i = 1$ and $\vartheta_i = 0$; otherwise $\delta_i = 0$ and ϑ_i represents the time when the completion time of task τ_i exceeds its deadline. Based on the defined constraints and decision variables, the objective function of the model is

formulated in Eq. (9) which is a multi-objective optimization problem. It involves maximizing the number of tasks that meet their deadlines and minimizing the deadline violation time for tasks that cannot meet their deadlines.

$$\text{Max } (w_1 \cdot \sum_{i \in T} \delta_i \cdot q_i - w_2 \cdot \sum_{i \in T} \vartheta_i \cdot q_i) \quad (9)$$

Subject to:

Constraints (1)-(8)

where w_1 and w_2 are the weights assigned to the objectives and $w_1 + w_2 = 1$. Indeed, the weights reflect the importance or priority of each objective. Moreover, we consider the priority of tasks in the objective functions and q_i indicates the normalized priority of the task τ_i using the softmax function in Eq. (10) to prevent sharp changes in the learning process that may arise due to large reward or penalty values.

$$q_i = \frac{e^{p_i}}{\sum_{j=1}^{n} e^{p_j}} \quad (10)$$

5 Q-LEARNING ALGORITHM FOR TASK OFFLOADING

In this section, we propose a Q-learning algorithm which is a type of reinforcement learning (RL) algorithm to solve the formulated problem in Section 4. Reinforcement learning is a type of machine learning technique in which an agent learns how to maximize the cumulative reward by taking actions and receiving feedback in the form of rewards or penalties (Sutton et al., 1998).

The proposed RL model is represented by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R} >$, where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ define a finite set of the system space, a finite set of actions, and the reward associated with taking an action in a given state, respectively. In this paper, the system state space is comprised of all tasks waiting to be assigned. In each step, the agent selects a task out of all the tasks in the queue and takes action for the task. Therefore, the system state space $\mathcal{S}$ is represented as follows:

$$\mathcal{S} = \left(s(\tau_1), .., s(\tau_i), .., s(\tau_n) \right) \quad (11)$$

A task is selected from all the tasks waiting to be assigned according to the current state. After assigning a task to a node in the fog/cloud layer, the system transfers to the next state.

The action space describes the set of possible decisions that the agent can make in a given state. Considering a given state, e.g., $s(\tau_i)$, selecting a node in the fog layer, i.e., $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, .., \mathcal{F}_u\}$ or cloud layer, i.e., $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, .., \mathcal{C}_v\}$ is defined as an action for the given state. Therefore, selecting a computing layer and a node from it is defined as an action the agent takes in a given state. After a task is assigned to a node in the fog or cloud layer, the task is added to the queue of the node, and the state of the system changes to the next state. Moreover, the start and completion times of tasks on the allocated nodes are used as parts of the state/action description as follows:

$$\mathcal{S}/\mathcal{A} = \left(.., \overbrace{\left[N_{kj}, w_{ikj}, m_{ikj} \right]}^{s(\tau_i)}, .. \right) \quad (12)$$

This state/action space indicates that task τ_i is assigned to node N_{kj} where the notations w_{ikj} and m_{ikj} are the start time and the completion time of the task on the allocated node according to Eqs. (3) and (6), respectively.

The RL agent explores the environment by taking actions from the action set in given states and receiving feedback from the environment. This feedback is defined in terms of the reward function and the precise definition of the reward function plays a vital role in the learning process.

As mentioned in Section 4, our objective is to maximize the number of tasks that meet their deadlines and minimize the deadline violation time of tasks that cannot meet their deadlines. According to the defined objective function in Eq. (9), the reward function for taking an action in state $s(\tau_i)$ is defined as follows:

$$\mathcal{R} = \begin{cases} w_1 \cdot q_i & \delta_i = 1 \\ -w_2 \cdot \vartheta_i \cdot q_i & \delta_i = 0 \end{cases} \quad (13)$$

where the agent receives a positive reward $w_1 \cdot q_i$ if task τ_i meets its deadline, i.e., $\delta_i = 1$; otherwise it receives a penalty $-(w_2 \cdot \vartheta_i \cdot q_i)$. In this formula, q_i is the normalized priority of the task τ_i. We consider the priority of tasks in receiving rewards or penalties to maximize the number of high-priority tasks that meet their deadlines and to decrease their deadline violation time. Moreover, w_1 and w_2 are positive constants.

This paper uses model-free reinforcement learning, where the agent tries to learn the consequences of its actions using algorithms such as policy gradient and Q-learning. Indeed, a model-free algorithm does not estimate the transition probability distribution. Therefore, the agent interacts with the environment by taking actions and receiving rewards to learn to optimize its policy. We use the Q-learning algorithm, which is a popular model-free, value-based reinforcement learning algorithm (Watkins and Dayan,

1992). This algorithm uses a Q-table to learn the optimal action to take in a given state.

The main idea behind Q-learning is to learn a Q-table, denoted as $Q(s,a)$. This table represents the expected cumulative reward of taking action a in state s. Before learning begins, the Q-values are initialized to a fixed value, e.g. 0. Then, in each step, the agent selects an action a in state s, receives a reward, and the system transfers to a new state s'. The Q-value then is updated based on the received rewards and the estimated future rewards. The Q-learning update rule is based on the Bellman equation (Sutton et al., 1998) as follows:

$$Q(s,a) = (1-\alpha)\cdot Q(s,a) + \alpha \cdot \left(\mathcal{R} + \gamma \cdot \max_{a'} Q(s',a')\right) \tag{14}$$

where α, γ, and $\mathcal{R}$ are the learning rate, the discount factor for future rewards, and the immediate reward received after taking the action a in the state s, respectively. The learning rate, i.e. $0 \leq \alpha \leq 1$, determines the impact of newly acquired information (reward ($\mathcal{R}$) and the maximum Q-value of the next stat ($\max_{a'} Q(s',a')$)) on the current Q-value. The discount factor, i.e. $0 \leq \gamma \leq 1$, indicates the extent to which the agent considers future rewards in its decision-making process. When the agent takes an action $a \in \mathcal{A}$ in the state $s \in \mathcal{S}$, the system changes its state from s to s'. Here, a' is the action chosen in the next state s', and $\max_{a'} Q(s',a')$ represents the maximum Q-value in the next state.

The goal of a Q-learning algorithm is to learn the optimal policy and follow it thereafter. The optimal policy is defined by the learned Q-values and is to select the action with the highest Q-value for a given state as follows:

$$\pi(a|s)^* = \arg\max_a Q(s,a) \tag{15}$$

Afterwards, the agent follows the optimal policy to decide which action to take in a given state. During the learning process, we use the epsilon-greedy policy (Wunder et al., 2010) to keep a balance between exploration to discover the environment and exploitation of prior knowledge. The policy chooses a random action with probability ε, i.e., exploration, and with probability $1-\varepsilon$, it selects the action with the maximum Q-value, where $0 \leq \varepsilon \leq 1$ is the exploration parameter.

The proposed task offloading algorithm is shown in Algorithm 1, called TOQL. It takes the learning rate α, discount factor γ, epsilon ε, number of IoT tasks n, available nodes in the fog layer and cloud layer $\mathcal{N}$ as inputs. The objective of the algorithm is to learn the optimal policy so that the best fog or cloud nodes are selected for offloading a given set of tasks. Hence, the output of the proposed algorithm is the Q-table values and the optimal policy. The algorithm initially creates a Q-table containing values 0. Afterwards, the algorithm performs episodes repeatedly until reaches the termination condition which is a predefined number of time steps. The TOQL algorithm performs the following steps for all episodes. First, a set of tasks T is generated, and the state space $\mathcal{S}$ and the state of available resources $\mathcal{N}$ are initialized by resetting the environment. That is, the state space $\mathcal{S}$ is initialized by a set of tasks T awaiting to be assigned to the resources, and the local queues of available resources $\mathcal{N}$ are reset. Then, the tasks are assigned to the available resources in the fog-cloud computing continuum and the Q-table is updated using Algorithm 2. In this algorithm, for each $s(\tau_i) \in \mathcal{S}$, the agent takes an action based on the epsilon-greedy policy. This means if the random number x is smaller

Data: α: learning rate, γ discount factor, ε: a small value, n, $\mathcal{N}$
Result: A Q-table defining estimated optimal policy π^*
Initialize $Q(s,a) \leftarrow 0$
for *each episode* **do**
  Generate task set $T = \{\tau_1, .., \tau_n\}$
  Initialize state space $\mathcal{S}$ and available resource $\mathcal{N}$
  Assign tasks to the resources and update Q-table using Algorithm 2
end

Algorithm 1: TOQL algorithm.

Data: State space S, Q-table, α, γ, ε, $\mathcal{N}$
Result: The Q-table
for *each* $s(\tau_i) \in S$ **do**
  $x \leftarrow$ uniform random number between 0 and 1
  if $x < \varepsilon$ **then**
    $a \leftarrow$ random action from the action space
  else
    $a \leftarrow \max Q(s,a')$
  end
  Take action a and add task τ_i to the queue of the selected node
  Get the start and completion time of task τ_i on the allocated node using Eqs. (3)-(6)
  Calculate immediate reward $\mathcal{R}$ using Eq. (13)
  Transfer to the next state s'
  Update Q-table using Eq. (14)
end

Algorithm 2: Epsilon-greedy resource allocation algorithm.

than ε then a random computing layer and a random node from the layer are selected and task τ_i is assigned to the node. Otherwise; the agent takes an action with the highest Q-value for the task τ_i, which this decision is based on the prior knowledge in the Q-table. In the next step, the start time and completion time of task τ_i on the allocated node are calculated according to Eqs. (3)-(6). The immediate reward $\mathcal{R}$ is calculated based on the completion time of task τ_i using Eq. (13) and the system state transfers to new state s'. Finally, the agent updates the Q-table values using Eq. (14).

Data: α: learning rate, γ discount factor, ε: a small value, n, $\mathcal{N}$
Result: A Q-table defining estimated optimal policy π^*
Initialize $Q(s,a) \leftarrow 0$
for *each episode* **do**
  Generate task set $T = \{\tau_1, .., \tau_n\}$
  Sort task set T based on the priority of the tasks.
  Initialize state space $\mathcal{S}$ and available resource $\mathcal{N}$
  Assign tasks to the resources and update Q-table using Algorithm 2
end

Algorithm 3: PTOQL algorithm.

We also propose the PTOQL algorithm, shown in Algorithm 3, which sorts the tasks based on their priorities and then assigns them to the available resources. In this algorithm, a set of tasks T is generated. Then the tasks set T is sorted based on the priority of tasks in descending order. In the next step, the state space $\mathcal{S}$ is initialized by sorted tasks, and the state of available resources $\mathcal{N}$ is initialized by resetting the environment. Then, the tasks are assigned to the available resources in the fog-cloud computing continuum and the Q-table is updated using Algorithm 2. The idea behind this algorithm is to assign tasks with higher priority to the available resources before tasks with lower priority to reduce the waiting time for high-priority tasks.

6 PERFORMANCE EVALUATION

For the simulation experiments performed in this work, we implemented the proposed algorithm in Python on a laptop with Intel Core i7 2.3 GHz, 32 GB RAM, and a Windows 10 operating system.

We consider a fog-cloud computing continuum providing several heterogeneous fog-cloud nodes in which the total number of nodes in the fog-cloud layer

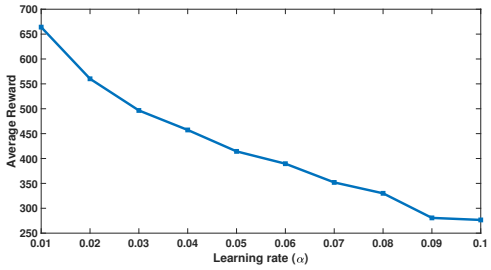

Figure 2: The reward convergence analysis over changes in α.

is varied from 30 to 90. The communication bandwidth of the link from IoT devices to fog nodes is assumed to be 1000 Mbps, while the communication bandwidth between IoT devices and cloud nodes are assumed to be 100 Mbps. To ensure the heterogeneity of fog and cloud nodes, the CPU processing capacity of each fog node is assumed to be uniformly distributed in $[3000-6000]$ MIPS, while the CPU processing capacity of each cloud node is assumed to be uniformly distributed in $[7000-12000]$ MIPS (Fizza et al., 2019). The number of tasks ranges from 100 to 500, and we consider two types of tasks, compute-intensive (type 1) and data-intensive (type 2) tasks. For data-intensive tasks, the computation size, i.e. cs_i is assumed to be in the range of $[100-400]$ MI, while for compute-intensive tasks, it is in the range of $[1000-4000]$ MI.

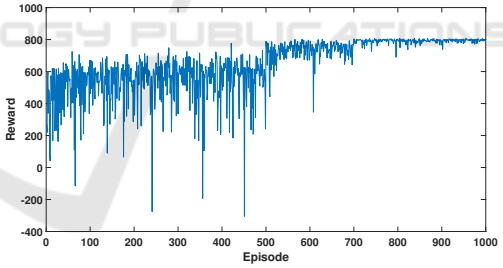

Figure 3: The reward convergence analysis over episodes.

To choose an appropriate learning rate, i.e. α, the convergence of the TOQL algorithm was analyzed with different learning rates in the range $[0.01-0.1]$, as shown in Figure 2. According to this experiment, the best learning rate is 0.01 with an average reward of 664. Therefore, $\alpha = 0.01$ was considered in the experiments. Figure 3 shows the reward convergence analysis of the TOQL algorithm. As can be observed, the reward of the algorithm converges after 700 training episodes, and the reward fluctuation decreases significantly after that.

Figure 4 shows the reward convergence analysis of the TOQL algorithm over changes in the discount factor, i.e., γ. As can be seen, the best results were

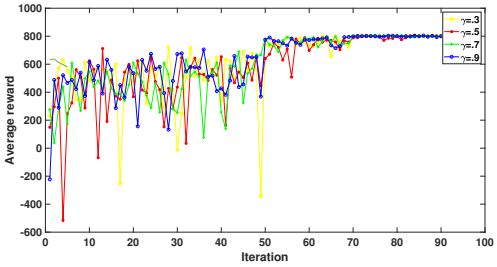

Figure 4: The reward convergence analysis over changes in γ.

observed for $\gamma = 0.9$, as the reward for this value of γ fluctuates less than other values, and converges after 50 iterations. In this experiment, the averages of 10 episodes are reported as an iteration.

In the experiments, a variable number of tasks and available nodes was considered and formed into different groups: Group 1 (100×30), Group 2 (200×45), Group 3 (300×60), Group 4 (400×75), and Group 5 (500×90), in the structure (n×m), where n and m are the number of IoT tasks and available nodes in the fog-cloud computing, respectively. The performance of PTOQL and TOQL are compared to Detour (Misra and Saha, 2019), the First Come First Serve (FCFS), and the Earliest Deadline First (EDF) algorithms.

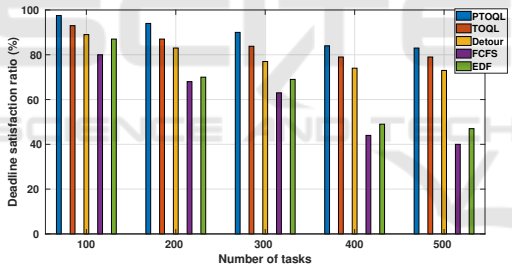

Figure 5: The deadline satisfaction ratio.

Detour makes optimal decisions on the local or remote execution of tasks and then selects the optimal fog nodes for task offloading. It also selects the optimal path for offloading tasks. We ignore the optimal path selection phase since this paper focuses on the task scheduling process.

Figure 5 shows the percentage of deadlines met for the PTOQL, TOQL, Detour, FCFS, and EDF algorithms over changes in the number of tasks. As can be seen, when the number of tasks increases, the percentage of tasks that meet their deadline decreases. Since PTOQL and TOQL consider the workload of the tasks, the data transmission overhead, and the waiting time of the IoT tasks, overall, the deadline for a higher percentage of tasks is met compared to other algorithms. However, the performance of the PTOQL algorithm is better than TOQL since it sorts the tasks based on their priorities and then takes action for each task. Furthermore, the performance gap between the proposed PTOQL and TOQL algorithms and other algorithms increases, indicating the superiority of their performance as the system load increases.

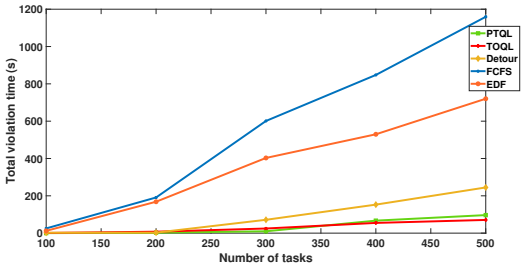

Figure 6: Total violation time.

Figure 6 illustrates the total violation time of the PTOQL, TOQL, Detour, FCFS, and EDF algorithms over changes in the number of tasks. As can be seen, when the number of tasks increases, the total violation time significantly increases for FCFS and EDF algorithms. The PTOQL and TOQL algorithms try to find nodes with minimum violation time by considering the communication time and waiting time of tasks when tasks cannot meet their deadline. Therefore, as the system load increases, their performance is significantly better than other algorithms. Although the TOQL algorithm performs better than the PTOQL algorithm as the system load increases, the performance gap between the PTOQL and TOQL algorithms is not significant.

7 CONCLUSION

This paper formulated the problem of task offloading in the edge-cloud computing continuum as a non-linear mathematical programming model. This model maximizes the number of IoT tasks that meet their deadlines and minimizes the deadline violation time for those tasks that cannot meet their deadline. This paper also proposed two Q-learning algorithms, called TOQL and PTOQL, to solve the formulated problem. The proposed TOQL algorithm considers the priority of tasks, the workload of tasks, and input/output data transmission time to select appropriate nodes from fog-cloud computing layers. Furthermore, it considers the waiting time of tasks as each node can process tasks sequentially. The PTOQL sorts the tasks based on their priorities and then assigns the tasks to available resources like the TOQL algorithm. Experimental evaluation shows that the proposed Q-learning algorithms significantly outper-

form the baseline algorithms in terms of the percentage of tasks that meet their deadlines and in reducing the violation time of tasks when completion time exceeds their deadlines. For future work, we are going to implement the task offloading problem with a deep Q-learning algorithm.

ACKNOWLEDGEMENTS

The work in this paper is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the PROVIDENT and INTERCONNECT projects and KKS foundation through the projects DPAC and FIESTA.

REFERENCES

Aazam, M., Zeadally, S., and Harras, K. A. (2018). Offloading in fog computing for iot: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems*, 87:278–289.

Abdi, S., Ashjaei, M., and Mubeen, S. (2022). Cognitive and time predictable task scheduling in edge-cloud federation. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE.

Bouet, M. and Conan, V. (2018). Mobile edge computing resources optimization: A geo-clustering approach. *IEEE Transactions on Network and Service Management*, 15(2):787–796.

Fizza, K., Auluck, N., and Azim, A. (2019). Improving the schedulability of real-time tasks using fog computing. *IEEE Transactions on Services Computing*, 15(1):372–385.

Hortelano, D., de Miguel, I., Barroso, R. J. D., Aguado, J. C., Merayo, N., Ruiz, L., Asensio, A., Masip-Bruin, X., Fernández, P., Lorenzo, R. M., et al. (2023). A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems. *Journal of Network and Computer Applications*, 216:103669.

Islam, A., Debnath, A., Ghose, M., and Chakraborty, S. (2021). A survey on task offloading in multi-access edge computing. *Journal of Systems Architecture*, 118:102225.

Jararweh, Y., Issa, M. B., Daraghmeh, M., Al-Ayyoub, M., and Alsmirat, M. A. (2018). Energy efficient dynamic resource management in cloud computing based on logistic regression model and median absolute deviation. *Sustainable Computing: Informatics and Systems*, 19:262–274.

Lou, J., Tang, Z., Jia, W., Zhao, W., and Li, J. (2023). Startup-aware dependent task scheduling with bandwidth constraints in edge computing. *IEEE Transactions on Mobile Computing*.

Misra, S. and Saha, N. (2019). Detour: Dynamic task offloading in software-defined fog for iot applications. *IEEE Journal on Selected Areas in Communications*, 37(5):1159–1166.

Mohammadi, S., PourKarimi, L., Droop, F., De Mecquenem, N., Leser, U., and Reinert, K. (2023). A mathematical programming approach for resource allocation of data analysis workflows on heterogeneous clusters. *The Journal of Supercomputing*, pages 1–30.

Nan, Z., Wenjing, L., Zhu, L., Zhi, L., Yumin, L., and Nahar, N. (2022). A new task scheduling scheme based on genetic algorithm for edge computing. *Computers, Materials & Continua*, 71(1).

Nguyen, B. M., Thi Thanh Binh, H., The Anh, T., and Bao Son, D. (2019). Evolutionary algorithms to optimize task scheduling problem for the iot based bag-of-tasks application in cloud–fog computing environment. *Applied Sciences*, 9(9):1730.

Razaq, M. M., Rahim, S., Tak, B., Peng, L., et al. (2022). Fragmented task scheduling for load-balanced fog computing based on q-learning. *Wireless Communications and Mobile Computing*, 2022.

Saeed, A., Chen, G., Ma, H., and Fu, Q. (2023). A memetic genetic algorithm for optimal iot workflow scheduling. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 556–572. Springer.

Smys, D. S. and Ranganathan, D. G. (2020). Performance evaluation of game theory based efficient task scheduling for edge computing. *Journal of IoT in Social, Mobile, Analytics, and Cloud*, 2(1):50–61.

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Wang, X., Ma, L., Li, H., Yin, Z., Luan, T., and Cheng, N. (2022). Digital twin-assisted efficient reinforcement learning for edge task scheduling. In *2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*, pages 1–5. IEEE.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8:279–292.

Wunder, M., Littman, M. L., and Babes, M. (2010). Classes of multiagent q-learning dynamics with epsilon-greedy exploration. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1167–1174.

Xue, F., Hai, Q., Dong, T., Cui, Z., and Gong, Y. (2022). A deep reinforcement learning based hybrid algorithm for efficient resource scheduling in edge computing environment. *Information Sciences*, 608:362–374.

You, C., Huang, K., Chae, H., and Kim, B.-H. (2016). Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411.