Reinforcement Learning-Based Workload Scheduling for Edge Computing

tao zheng (zhengtao@zjsru.edu.cn)
Information Sciences Institute https://orcid.org/0000-0001-5363-8137
Jian Wan
School of Computer Science and Technology,Hangzhou Dianzi University
jilin Zhang
School of Computer Science and Technology,Hangzhou Dianzi University
Congfeng Jiang
School of computer science and technology,hangzhou dianzi university

Research Article

Keywords: Edge Computing, Task Offloading, Workload Scheduling, Reinforcement Learning

Posted Date: March 30th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-349535/v1

Reinforcement Learning-Based Workload Scheduling for Edge Computing

Tao Zheng*†, Jian Wan*, Jilin Zhang*,Congfeng Jiang*
* School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China
† College of Information Science and Technology, Zhejiang Shuren University, Hangzhou 310015, China

***Abstract*—Edge computing is a new paradigm for providing cloud computing capacities at the edge of network near mobile users. It offers an effective solution to help mobile devices with computation-intensive and delay-sensitive tasks. However, the edge of network presents a dynamic environment with large number of devices, high mobility of the end user, heterogeneous applications and intermittent traffic. In such environment, edge computing always encounters workload scheduling problem of how to efficiently schedule incoming tasks from mobile devices to edge servers or cloud servers, which is a hard and online problem. In this work, we focus on the workload scheduling problem with the goal of balancing the workload, reducing the service time and minimizing the failed task rate. We proposed a reinforcement learning-based approach, which can learn from the previous actions and achieve best scheduling in the absence of a mathematical model of the environment. Simulation results show that our proposed approach achieves the best performance in aspects of service time, virtual machine utilization, and failed tasks rate compared with other approaches. Our reinforcement learning-based approach can provide an efficient solution to the workload scheduling problem in edge computing.**

***Index Terms*—Edge Computing, Task Offloading, Workload Scheduling, Reinforcement Learning**

I. Introduction

Nowadays, with the increasing popularity of mobile devices, more novel sophisticated applications are emerging, such as face recognition, inter-active gaming and augmented reality [1]. However, duo to resource constraints (processing power, battery lifetime, storage capacity), mobile devices cannot meet the needs of running these novel sophisticated applications on local [2]. Considering the powerful computing and storage capabilities of the cloud server, one suitable solution is to offload these complicated mobile applications to cloud for processing, so called mobile cloud computing (MCC) [3]. MCC can efficiently address the problems of limited processing capabilities and limited battery of the mobile devices [4]. However, the cloud server is generally far away from mobile devices, MCC inevitably suffers from high latency and bandwidth limitation [5]. Moreover, according to the prediction of Cisco, the growth rate of mobile data required to be processed will far exceed the capacity of central clouds in 2021 [6].To resolve these issues, Edge computing has emerged as a promising technology that provides cloud computing capabilities at the edge of the network in close proximity to the mobile subscribers [7]. Compared with MCC, edge computing has the advantage of lower latency, lower core network load and more security [8].

Although Edge computing is a new technology with many advantages, it still has many problems to be solved [9]. The edge of network presents a very dynamic environment with large number of devices, high mobility of the end user, heterogeneous applications and intermittent traffic [10]. In such environment, edge computing always encounters the problem of how to efficiently schedule incoming tasks from mobile devices to edge servers and cloud servers [11]. To elaborate, the edges consist of terminal devices and Internet of Things (IoT), which are widely distributed, numerous heterogeneous and highly mobile. When these edge devices are running, they can generate a variety of different tasks. Duo to the resource constraint of edge devices, most of tasks need to be offloaded to the outside servers for processing. However, these offloaded tasks are unevenly distributed and random, which will lead to imbalanced workloads among edge servers and impair the performance of system [12]. For example, when massive amount of tasks are offloaded to the same edge server simultaneously, it is easy to cause single edge server paralysis and network congestion, while other edge servers may be in idle state. Therefore, how to schedule the incoming stream of offloaded tasks determines the overall efficiency and scalability of the edge computing system. Moreover, both the communications and computation resources also need to be allocated and scheduled efficiently for better system performance [13].

Our work focuses on the workload scheduling problem which can be defined as deciding on the destination computational unit for each offloaded task within an edge computing system. As we know, workload scheduling is an intrinsically hard, online problem [10]. Where to offload decision should take many parameters into consideration such as task property, network situation and computational resources [14] [15]. These parameters are also highly dynamic especially under unexpected variation of the load [16]. To solve problem, we propose reinforcement learning (RL)-based workload scheduling approach, which can learn from the previous actions and achieve best scheduling in the absence of a mathematical model of the environment. Finally, we carry out an experimental evaluation based on EdgeCloudSim [17].To demonstrate the performance of our approach, we evaluated with four opponent algorithms using crucial performance metrics such as service time, failed tasks rate, and virtual machine(VM) utilization. According to the results, our proposed method has competitive performance with respect to its opponents for the

cases studied. The contributions of this paper are summarized as follows:

We investigate workload scheduling in edge computing, aiming at balance the workload, reduce the service time and minimize failed task rate.

We analyze system model, which includes task model, network model, computational model, according to the Multi-tier edge computing architecture, and formulate the workload scheduling problem based on system model as an NP-hard problem.

We proposed a RL-based approach, which can learn from the previous actions and achieve best scheduling in the absence of a mathematical model of the environment.

The remainder of this paper is organized as follows. Section II briefly reviews the related work. Section III presents the system model and problem formulation. Section IV describes the proposed RL-based workload scheduling approach. Section V elaborates on the simulation experiment design and analyzes the results. Section VI concludes our study and provides possible directions for future research.

II. RELATED WORK

In edge computing, Mobile devices can offload most tasks to the edge server for execution, which efficiently address the problems of their limited resources and reduce the core network traffic load. However, improper task offloading not only brings imbalance workload among edge servers, but also increases task latency and energy consumption [18]. Therefore, properly scheduling computation tasks among edge servers are crucial to optimize the quality of services with high resource efficiency. The scheduling research is to choose the proper decision on the time and place where the task should be offloaded. There has been extensive work devoted to workload scheduling problem. Santoro et al. [19] propose a software platform called Foggy for workload orchestration and resource negotiation in fog computing environment. It schedules the execution location of tasks based on computational, storage or network resources. Anas et al. [20] take computational utilization and access probability into consideration, and develop a performance model based on queuing theory to address the workload balancing between service providers within a federated cloud environment. Ma et al. [21] consider cooperation among edge nodes and investigate the workload scheduling with the objective of minimizing the service response time as well as the outsourcing traffic in mobile edge computing. They propose a heuristic workload scheduling algorithm based on water-filling to reduce computation complexity. Fuzzy logic is an effective method to solve workload scheduling problem in edge computing, which has been discussed in recent years. Sonmez et al. [22] employ a fuzzy logic-based approach to solve the workload orchestration problem in the edge computing systems. Their approach takes into consideration the properties of the offloaded task as well as the current state of the computational and networking resources, and uses fuzzy rules to define the workload orchestration actions in terms of networking, computation and task requirements to decide the task execution location within the overall edge computing system.

Different from the existing work, We proposed a RL-based approach for workload scheduling in edge computing, which can learn from the previous actions and achieve best scheduling in the absence of a mathematical model of the environment, aiming at balance the workload among edge servers, reduce the service time and minimize failed task rate.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first introduce the Multi-tier edge computing architecture and analyze system model, which includes task model, network model, and computing model. Then, we formulate the workload scheduling problem based on system model. The edge computing architecture as depicted in Fig.1

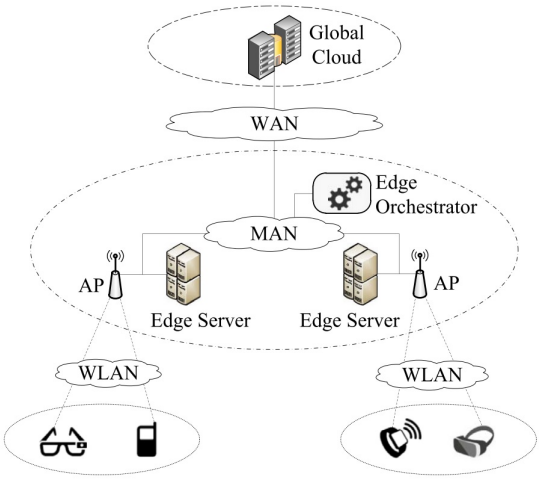


Fig. 1. Multi-tier edge computing architecture with edge orchestrator.

A. Multi-tier edge computing architecture

As shown in Fig.1, we construct a multi-tier edge computing system architecture which incorporates computational resources at various levels, as well as different ranges of networks, such as local area network (LAN), metropolitan area network (MAN) and wide area network (WAN). The first tier is edge device layer, which is composed of a large number of mobile devices, IoTs and other edge devices. They communicate with local edge server via wireless local area network (WLAN). The second tier is edge server layer, which consist of edge servers and the edge orchestrator. Edge servers are interconnected through MAN. The third tier is cloud server layer, which is the global cloud server distributed around world and provide cloud service through WAN.

In this architecture, each edge server can provide computing services for users within its WLAN coverage, while cloud server can provide remote computing services for all users. Moreover, nearby edge servers can also provide computing services for neighbor LAN users in domain. When an edge server cannot provide sufficient resources or computation

capacity, the computation tasks can be scheduled to the nearby edge servers that are under-utilized or the cloud for processing. In this process, the edge orchestrator acts as the workload coordinator among servers to monitor the environment information and make scheduling decisions for each task to coordinate workload of servers.

To elaborate, we take the university campus application scenario as an example. According to this architecture, the students and other participants are assumed to carry and/or wear the mobile devices, such as smart phone, Google glasses, Smart Bracelet, etc., on which applications run and continuously generate tasks to be processed. Duo to the limited processing capacity of a single device, these devices have to offload some tasks to external servers. To meet this need, a number of edge servers are deployed at different places (such as classroom buildings, dormitories, and library) on campus to provide coverage for request services. In the real scene, the places on campus may have different user densities according to the time of day. For example, students concentrated their classes in the classroom buildings in the morning, and they may gather together in the library for reading in the afternoon, and most of them are likely stay in dormitories at night. The user density can directly affect the amount of requested and workload. Typically, the user's mobile devices select the nearest local edge server to offload tasks via the WLAN, such as Wi-Fi hotspots or cellular base stations. In our system, we define the places with three type attractiveness levels, which are based on user's device density. If the attractiveness level of the place is high, such as an auditorium during the conference, lots of students may wear the mobile device and gather there. Duo to the high device density and excessive task offloading, both the network and the computational resources are likely to be congested in LAN. To deal with this problem, the local edge server also needs to offload these tasks which beyond its capacity to the nearby edge servers or cloud server. However, how to schedule these tasks to achieve the desired optimization objectives is not an easy problem.

B. System model

To investigate workload scheduling problem, we should first model the edge computing system according to the above architecture, including task model, network model and computing model. We set that the system consists of k mobile devices, m edge servers and one remote cloud server. k and m denote the number of mobile devices and edge servers respectively. For each device, there is only one local edge server which can be accessible via WLAN and $(m-1)$ nearby edge severs which can be accessible via MAN. Each task offloaded from mobile device can be scheduled to execute on local edge server, nearby edge server, or cloud server. In general, the local server is the first choice due to the advantages of short distance and low latency. However, when local edge server cannot provide sufficient resources or computing power, the corresponding tasks should be scheduled to the nearby edge servers or cloud for processing. In this process, the edge orchestrator acts as the workload coordinator among servers to monitor the environment information and make scheduling decisions for each task to coordinate workload of servers.

1) Task Model

Each mobile device can produce different tasks, which may be compute-intensive, I/O intensive or both. To better represent the tasks, we describe a $task_i$ in a four-field notation $(\partial_i, \beta_i, c_i, \tau_i)$, which ∂_i denotes the input data size (in bits), β_i denotes the output data size (in bits), and c_i denotes the total number of CPU cycles that is required to complete the $task_i$, and τ_i denotes the delay constraint of $task_i$. We assume that all the tasks are indivisible and offloaded as a whole to edge server or cloud for execution, and the tasks are independent of each other.

2) Network Model

As shown in Fig. 1, In our edge computing system, tasks can be scheduled to execute on any server, such as local edge server, nearby edge servers or cloud server. Therefore, tasks may pass through multiple networks during scheduling process, such as WLAN, MAN or WAN. Considering the difference of bandwidth, transmission rate and interference among different networks, these will have a certain impact on the transmission delay and performance of task scheduling. We calculate the communication delay generated by scheduling tasks to local edge server, nearby edge servers and cloud server, respectively: If the local edge server is selected as offloading destination, the mobile device first uploads the computing to the local edge server through a wireless channel, and then edge server returns the result to the mobile device after the task is completed. In this process, the communication delay on WLAN is mainly caused by transmission delay of $task_i$, which can be expressed as:

$$\begin{gathered} t^i_{com} \approx t^i_{td} = t^i_{up} + t^i_{down} \\ t^i_{up} = \frac{\partial_i}{R_{wlan}}; \quad t^i_{down} = \frac{\beta_i}{R_{wlan}}; \end{gathered} \tag{1}$$

where, t^i_{com} represent the communication delay of $task_i$, t^i_{td} represent the transmission delay of $task_i$, t^i_{up} represent the upload time of $task_i$, and t^i_{down} represent the result download time of $task_i$,R_{wlan} represent WLAN upload and download transmission rate. In our system, we assume that the upload and download transmission rate between mobile devices and local edge server are the same. The WLAN transmission rate is quantified as:

$$R_{wlan} = \omega_{wlan} \log_2 (1 + \frac{p_n h_{n,s}}{N_0 + \sum_{n=1}^{M} p_n h_{n,s}}) \tag{2}$$

Where, ω_{wlan} is the bandwidth of WLAN, p_n is the transmission powers of edge device, $h_{n,s}$ is the channel gains between the n_{th} edge device and the s_{th} edge server, N_0 is the noise power.

According to the above formulas, we can see that the WLAN transmission rate is closely related to the bandwidth. For example, if too many mobile devices choose the same

channel to communicate with the base station and the interference between them will increase, which will affect the transmission rate.

If the nearby edge server is selected as offloading destination, the $task_i$ will be scheduled to the nearby edge server via the MAN, the transmission delay is mainly affected by the MAN bandwidth. The transmission delay of the $task_i$ on MAN can be expressed as:

$$t^i_{td} = t^i_{up} + t^i_{down} = \frac{\partial_i + \beta_i}{R_{man}} \tag{3}$$

Where, R_{man} represent the MAN transmission rate.

If the remote cloud server is selected as offloading destination, the $task_i$ will be uploaded to the cloud server via the WAN. In general, there is a long distance between the cloud server and the user, the propagation delay of signal back and forth cannot be ignored. Therefore, the communication delay includes transmission delay t_{td} and propagation delay t_{pd}. The communication delay of the on WAN can be expressed as:

$$\begin{aligned} t^i_{com} &= t^i_{td} + t^i_{pd} \\ &= t^i_{up} + t^i_{down} + t^i_{pd} \\ &= \frac{\partial_i + \beta_i}{R_{wan}} + t^i_{pd} \end{aligned} \tag{4}$$

The propagation delay is much longer than the transmission delay ($t_{pd} \gg t_{td}$). Thus, the communication delay is mainly determined by the propagation delay.

3) Computing Model

For workload scheduling, an important indicator is service time, which represents the time it takes for the task to upload to the server until it is completed. Service time consists of task waiting time and execution time. In this section, we will calculate the service time required on the local edge server, nearby edge server and the cloud, respectively. When a task is uploaded to the local edge server, it first will be generally arranged in the task queue of server to wait for processing. There is a waiting time between the uploading to server and starting execution. The waiting time is related to the VM utilization of the local edge server. If VM utilization is low, which means that the current local edge server is relatively idle, the task can get VM resources quickly and its waiting time will be short. On the contrary, if VM utilization is high, which means that the current local edge server is relatively busy, and the waiting time will be long. Thus, the service time of $task_i$ on local edge server can be expressed as:

$$t^i_s = t^i_{wait} + t^i_{exe}, \;\; t^i_{exe} = \frac{c_i}{f_l} \tag{5}$$

Where,c_i is the total number of CPU cycles that is required to complete the $task_i$. f_l represents local edge server computing power. t^i_{wait} is waiting time between the uploading to local edge server and starting execution

Similarly, when the task is migrated to the nearby edge server, the service time of $task_i$ on nearby edge server can be expressed as

$$t^i_s = t^i_{wait} + t^i_{exe}, \;\; t^i_{exe} = \frac{c_i}{f_{nb}} \tag{6}$$

Where, f_{nb} represents nearby edge server computing power. t^i_{wait} represents waiting time between the uploading to nearby edge server and starting execution. When the $task_i$ is offloaded to the cloud, we consider that cloud resources are always sufficient, the waiting time of tasks can be ignored. Thus, the service time on cloud sever can be expressed as

$$t^i_s \approx t^i_{exe} = \tfrac{c_i}{f_c} \tag{7}$$

Where, f_c represents cloud computing power. According to the above analysis, as we know that the main factors that affect the service time are the amount of computation required for the task, VM utilization and computing power of the server.

4) Definition of task Failure Each task has its own delay constraint τ, if the running time exceeds this constraint value, the task is considered to be failure. We define the condition for task failure as follows:

$$F = \{task_i | t^i_{com} + t^i_{exe} > \tau_i, i \in N\} \tag{8}$$

C. Problem Formulation and Analysis

Based on the analysis of the task, network and computing model, we calculator the total delay of task scheduling to local edge server, nearby edge server and cloud server respectively. The total delay of $task_i$ execution on local edge server:

$$\begin{aligned} t^i_{local} &= t^i_{com} + t^i_s \\ &= t^i_{up} + t^i_{wait} + t^i_{exe} + t^i_{down} \\ &= \tfrac{\partial_i + \beta_i}{R_{wlan}} + \tfrac{c_i}{f_l} + t^i_{wait} \end{aligned} \tag{9}$$

The total delay of $task_i$ execution on nearby edge server:

$$\begin{aligned} t^i_{nb} &= t^i_{com} + t^t_s \\ &= t^i_{up} + t^i_{wait} + t^i_{exe} + t^i_{down} \\ &= \tfrac{\partial_i + \beta_i}{R_{man}} + \tfrac{c_i}{f_{nb}} + t^i_{wait} \end{aligned} \tag{10}$$

The total delay of $task_i$ execution on cloud server:

$$\begin{aligned} t^i_{cloud} &= t^i_{com} + t^t_s \\ &= t^i_{up} + t^i_{exe} + t^i_{down} + t^i_{pd} \\ &= \tfrac{\partial_i + \beta_i}{R_{wan}} + \tfrac{c_i}{f_c} + t^i_{pd} \end{aligned} \tag{11}$$

According to total delay of local edge server, nearby edge server and cloud server. Our optimization objective of the workload scheduling problem is to reduce the total delay and failure rate of the task. The optimization problem is formulated as follows:

$$\begin{aligned} \min \; t = \sum_{i=1}^{n} & \lambda_1 t^i_{local} + \lambda_2 t^i_{nb1} ... \lambda_m t^i_{nb(m)} + \lambda_{m+1} t^i_{cloud} \\ & s.t \;\; t^i_{com} + t^i_{exe} < \tau_i \\ & \lambda_l, \ldots \lambda_{m+1} \in \{0, 1\} \end{aligned} \tag{12}$$

Where, $\lambda_1, \ldots \lambda_{m+1}$ represent the scheduling decision variables. Since tasks are indivisible, these scheduling decision variable are integer variables of 0 and 1, but only one decision variable can be 1 in each decision, which represents the selected offloading destination. For example, If λ_1=1 and the

others are 0, which means the local edge server is selected as the offloading destination. If λ_i=1, $(i \in [2, m])$, which means the i_{th} nearby edge server is selected. If λ_{m+1}=1 and the others are 0, which means the cloud is selected as offloading destination. Since the decision variables are integer variables, the optimal solution can be found by traversing when the number of tasks is small. However, with the increasing of tasks, the scale of solution space will increase rapidly and become too large. In this case, Equation (12) is no longer a convex optimization problem, but a NP-hard [23]. Moreover, the effect of task scheduling is affected by many parameters, such as network bandwidth, VM utilization, and task attributes. These parameters can be highly dynamic especially under unexpected variation of the load. In the next subsection, we propose a task scheduling method based on reinforcement learning to solve the problem.

IV. THE RL-BASED WORKLOAD SCHEDULING APPROACH

In this section, we will introduce the key steps and algorithm of RL-based workload scheduling approach in the edge computing.

A. Reinforcement Learning

Reinforcement Learning is a branch of Machine Learning, which focuses on how to achieve the optimal goals through learning in a dynamic environment. In most machine learning, the learner is not told in advance which actions to take, but instead need to find out which actions will bring the most rewards through trial and error. In RL, the agent, as a learner, perceives the current state of environment and select actions to take. When the action is executed, the agent will receive a reward or punishment from environment according to the effect of action. If receiving a reward from the environment, the agent will increase the tendency to take this action in the future to obtain more rewards. On the contrary, if receiving a punishment, the agent will reduce the tendency to take this action. The Reinforcement Learning Paradigm is shown in Fig.2.

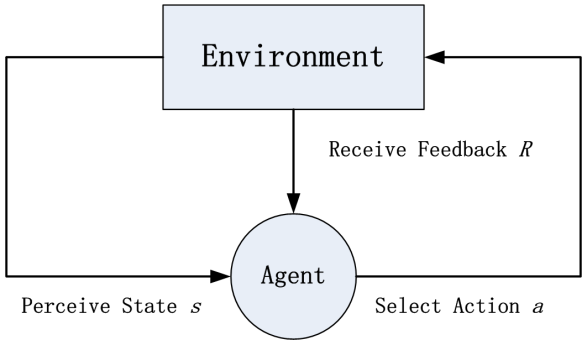


Fig. 2. The Reinforcement Learning Paradigm.

The process of interaction between agent and environment in reinforcement learning can be formalized in Markov decision process (MDP) as $\{S, A, P_a(s, s\prime), \gamma, R\}$, where S represents a (finite) state space: $S = \{s_1, s_2 \cdots s_t, s_{t+1}, \cdots s_n\}$,and s_t denotes the state at step t. A represents a (finite) action space $A = \{a_1, a_2, \cdots a_t, a_{t+1} \cdots a_n\}$,and a_t denotes the action taken at step t. $P_a(s, s\prime)$ represents the probability of transferring to next state $s\prime$ after taking action a in the state s. γ represents discount factor, R represents a reward function. According to the Markov property, the next state and reward depend only on the previous state and action, expressed as:

$$P_a(s, s\prime) = P\{s_{t+1} = s\prime, R_{t+1} = r | s_t = s, A_t = a\} \quad (13)$$

To maximize the cumulative reward, agent needs to balance the exploration and exploitation steps. In exploration step, the agent tries the actions that have not been selected before and explore new state to obtain higher reward. In exploitation step, the agent takes the best action that has already observed so far. However, the dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task.

B. Q-Table Definition

In our RL approach, the agent takes actions based on values present in the Q-table, which contains states, actions and Q-values. At beginning, the Q-table is initialized with zero or random values. The agent randomly selects previously unselected actions in the exploration phase to obtain higher reward, and updating the Q-values in the Q-table according to the rewards obtained after taking the action. In exploitation step, the agent will select the action with the highest Q-value in a given state to maximize the reward. Therefore, the Q-table plays a vital role in the RL process. We first define the Q-table and then introduce the contents of the table. Our Q-table is represented as follows:

TABLE I
THE Q-TABLE STRUCTURE

State Space	**Action Space**		
	a_l	a_e	a_c
s_1	$Q(s_1, a_l)$	$Q(s_1, a_e)$	$Q(s_1, a_c)$
s_2	$Q(s_2, a_l)$	$Q(s_2, a_e)$	$Q(s_2, a_c)$
s_3	$Q(s_3, a_l)$	$Q(s_3, a_e)$	$Q(s_3, a_c)$
$\cdots$	$Q(s_t, a_l)$	$Q(s_t, a_e)$	$Q(s_t, a_c)$
s_n	$Q(s_n, a_l)$	$Q(s_n, a_e)$	$Q(s_n, a_c)$

C. State Space

In RL approach, the first step is to define the state of the system environment. As far as we know, our multi-tier edge computing system has thousands of time-varying parameters at runtime. Therefore, how to select key parameters to accurately describe the current system environment is crucial to task scheduling. According to the previous analysis, the VM utilization status of the servers and the network conditions from users to servers should be considered. These parameters can be highly dynamic especially under unexpected variation of the workload. Moreover, the characteristics of task also play an important role on the system performance [24]. In our work, the state space S contains all of environment states, and each

state includes three parameters: server states, network states and tasks characteristic.

$$\begin{aligned} S &= \{s_1, s_2 \cdots s_n\} \\ s_t &= \{server_t, network_t, task_t\}, s_t \in S \end{aligned} \tag{14}$$

Where, $server_t$ represents the states of all servers at step t: $server_t = \{u_1, u_2 \cdots u_n\}$, the u_n is the VM utilization of n_{th} server. The $network_t$ is the states of networks at step t: $network_t = \{\omega^t_{WAN}, \omega^t_{MAN}, \omega^t_{WLAN}\}$, the ω^t_{WAN}, ω^t_{MAN} and ω^t_{WLAN} represent the bandwidth of WAN,MAN and WLAN at step t, respectively. $task_t$ is the task that need to be scheduled at step t : $task_t = \{\partial_t, \beta_t, c_t, \tau_t\}$, which ∂_t denotes the input data size (in bits), β_t denotes the output data size (in bits), and c_t denotes the total number of CPU cycles that is required to complete the task, and τ_t denotes the corresponding delay constraint of $task_t$.

In order to prevent the state space from being too large, we should minimize the dimension of state space while ensuring the accurate description of the state. In the selection of target server, the servers can be divided into local edge server, nearby edge servers and cloud server according to different network connections. For nearby edge servers, the network transmission cost of offloading to any one is the same [25]. Therefore, we can select the nearby edge server with the least loaded as target server. For cloud server, it is generally considered to be abundant in resources, and its changes have little impact on our state space. For the task, we select its execution length and size as influent factors. Finally, we select WAN, MAN and WLAN bandwidth, the utilization of local edge server, the least loaded nearby edge server, task length and data size as the state space. The state can be defined as

$$s_t = \{u_{local}, u^{\min}_{edge}, \omega^t_{WAN}, \omega^t_{MAN}, \omega^t_{WLAN}, \partial_t, c_t\}, s_t \in S \tag{15}$$

where, u_{local} is the local edge VM utilization, and $u^{\min}_{edge}$ is the one with min VM utilization in the nearby edge servers.

D. Action Space

The next step is to define the action space A. Actions determine the transfer rules between states. When a task arrives, it needs to decide which the suitable server should be choice to execute the task. Therefore, the action space includes all servers. In our system, the server consists of local edge server, multiple nearby edge servers and a cloud server. In nearby edge servers, we select the nearby edge server with the least loaded. At the decision step t, the action space can be simplified as follows:

$$A_t = \{a_l, a_e, a_c\} \tag{16}$$

Where, a_l represent the action that allocate task to the local edge server, a_e represent the action that allocate task to the nearby edge server with the least loaded, a_c represent the action that allocate task to the cloud server.

E. Reward Function

The reward function is used to describe the immediate reward value from one state to another state after taking an action [26]. It can make an evaluation for the selected action. We use the feedback information after task execution to determine whether it's a positive or negative reward. However, due to the delay of the feedback information, we cannot get them in time before entering the next state. Therefore, we adopt the lag processing method that is to record the task state first and then update the Q-Table after receiving feedback information. In this paper, our optimization objective is to reduce the total delay and failure rate of the task. As far as we can see, the main reason of task failure is the unreasonable task allocation, which leads to the task waiting for execution or the transmission time is too long and exceed its deadline. Therefore, we can evaluate the feedback based on task states. If the task is completed, the reward = 1, which means positive reward; If the task is fail, the reward = -1, which means negative reward; If the task is not finished, reward = 0, which means that it cannot be determined at present.

F. Q-value updating

The Q-Learning is one of well-known algorithm in reinforcement learning, which works by iterative updating Q-value to express the expected reward of taking a given action in a given state, in order to achieve the optimal strategy [27]. In Q-Learning algorithm, each state-action pair has an associated expected reward Q-value, known as Q-value function $Q(s, a)$. The agent will select the best action according to the Q-values in state .When an action is selected to execute, the corresponding Q-value function will be updated based on the reward received. The update rule for $Q(s, a)$ is the following:

$$\begin{aligned} Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \\ &\alpha \left[R_t + \gamma \max(Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t)\right] \end{aligned} \tag{17}$$

In this expression, the subscript t represents the steps of iterations, which means the old value $s_t \in S$, $a_t \in A$; $\alpha \in \{0, 1\}$ is the learning rate, which represents the willingness to update the current Q-value. For example, if $\alpha = 0$, the agent wants to update nothing. If $\alpha = 1$, the agent wants to update the current Q value as much as possible, and it is usually chosen for a small value, such as $\alpha = 0.1$ [28]. $\gamma \in \{0, 1\}$ is the discount factor, which is used for function converge. If γ is closer to 0, the agent will tend to consider only immediate reward. If γ is closer to 1, the agent will consider future reward with greater weight. $Q(s_{t+1}, a_{t+1})$ is the estimate Q-value of the next state from the greedy algorithm. In the training process, the action a_t is selected to execute in the current state s_t and transfer to the next state s_{t+1}, the corresponding Q-value function $Q(s_t, a_t)$ will be updated based on the current reward received and the estimated Q-value of the next state. In action selection decision, the agent uses a certain strategy (such as ε-greedy) to select the action and rapidly converge to the optimal direction by constantly updating the state space, and makes

its Q-value function (i.e., Q-value) approach to the optimal direction continuously [29].

G. RL-based workload scheduling algorithm

Our RL-based Workload Scheduling algorithm is presented in Algorithm 1, which is divided into five steps and we describe each step as follow:

Step 1(Input and Initialize parameters): The first step is to input variables and initialize parameters. The variables include $N_{\min}$, $N_{\max}$, Δ , α , γ , ε . $N_{\min}$ is the min number of mobile devices, $N_{\max}$ is the max number of mobile devices, Δ represents the number of devices increased each time. Thus, the number of iterations can be calculated as $n = \frac{N_{\max}-N_{\min}}{\Delta}$; we also need to set the learning rate α, the discount factor γ and the greedy coefficient ε, and initialize the Q-table, state space, action space and Q-values.

Step 2(Obtain Current Status): According to the parameters set in the first step, start the system and obtain current status. Observe all the nearby edge servers to find the one with the least VM utilization $u_{edge}^{\min}$, Obtain current local edge VM utilization u_{local} , the current bandwidth of WAN,MAN and WLAN, and the current$task_t$.

Step 3(Quantified Indicators): In order to reduce the dimension of the state space, the observed key indicators need to be quantified to build current state $s_t = \{u_{local}, u_{edge}^{\min}, \omega_{WAN}^t, \omega_{MAN}^t, \omega_{WLAN}^t, \partial_t, c_t\}$. Record all running states to fill state space S ,$s_t \in S$.

Step 4(Take Action): In this step, Agent applies the $\varepsilon-greedy$ approach to select action. if probability $p \leq \varepsilon$,agent takes new action that has not been selected before to obtain higher reward; Otherwise, the agent takes the action that has the highest Q-value so far;

Step 5(Update Q-values): After take action, the agent will receive a reward from environment and then update the Q-value of the current state according to equation (17).

TABLE II
THE DESCRIPTION OF RL-BASED WORKLOAD SCHEDULING ALGORITHM

Algorithm 1: RL-based Workload Scheduling Algorithm

Input: $N_{\min}, N_{\max}, \Delta, \alpha, \gamma, \varepsilon$
Output: Scheduling Decision
Initialize the Q-table, state space, action space and Q-values.
for number of devices n = $N_{\min}$ ... $N_{\max}$ **do**
  Observe current $u_{local}, \omega_{WAN}, \omega_{MAN}, \omega_{WLAN}, task_t$
  for each edge sever n = 1...m+1 **do**
    To find the least loaded edge host $\rightarrow u_{edge}^{\min}$
  end for
  Quantify the key indicators to build current state s_t.
  Record all running states to fill state space S.
  if p $\leq \varepsilon$ **then** agent takes new action that has not been selected before to obtain higher reward;
  else agent takes the best action that has already observed so far.
  After take action a_t ,receive the reward R and update Q-values:
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[R_t + \gamma \max(Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t)\right]$
  $s \leftarrow s'$
end for

V. EVALUATION RESULTS

In this section, we conducted a large number of experiments to evaluate the performance of proposed RL-based workload Scheduling approach. To illustrate the effectiveness of our approach, we compare our approach with other approaches in term of the average number of failed tasks, average service time, average VM utilization and average network delay. Before presenting the evaluation results, we first introduce the evaluation setup.

A. Evaluation Setup

In this work, all experiments are performed on the EdgeCloudSim [17], which provide a simulation environment specific to Edge Computing scenarios. Based on CloudSim [30], EdgeCloudSim adds considerable functionality such as network modeling, load generation as well as workload orchestrator, so that it can accurately simulate the real edge computing environment.

In the experiment, we adopt edge computing architecture is a two-tier with orchestrator, which is composed of 1 cloud servers, 14 edge servers and a large number of mobile devices. The cloud server has 4 Virtual Machine (VM) with 100 giga instruction per seconds (GIPS) of CPU power, 32GB of random access memory (RAM) and 1000 GB storage. Each edge server has one host that operates 4 VM with 10 GIPS of processing capacity, 2 GB of RAM and 16GB storage to handle the offloaded tasks. In order to simulate different loads, we initialize the number of mobile devices as $N_{\min} = 200$, and increase 200 each time to reach the maximum $N_{\max} = 2000$.

Each mobile device can run four different applications, namely augmented reality, infotainment, and health monitoring applications. Each application represents different task size, latency tolerance, and task length. The task has random input/output file sizes to upload/download and have random lengths.

According to the attractiveness of user, we divide places into three different levels, which directly affects the dwell time that the user spends in the related places. We set the mean waiting time of type 1,2 and 3 respectively as 60,30 and 15 minutes. We also set WAN Propagation Delay as 100ms and Lan Internal Delay as 5ms.We set the related parameters in simulations in table III.

B. Results and Analysis

In order to illustrate the effectiveness of our proposed RL-based workload Scheduling approach. We compared it with Utilization-based, Bandwidth-based and Fuzzy-based scheduling approaches in terms of average service time, and average VM utilization, failed tasks rate. In the following figures, we use rl, util, bw, and fuzzy abbreviations for RL-based, Utilization-based, Bandwidth-based and Fuzzy-based work-

TABLE III
THE RELATED PARAMETERS IN SIMULATIONS

Parameters		Values
Cloud Sever		has 4 VM
Each VM in Cloud sever	CPU	100GIPS
	RM	32GB
	Storage	1000GB
Edge Server		has 4 VM
Each VM in Edge Server	CPU	10GIPS
	RM	2GB
	Storage	16GB
Number of Mobile Devices		200 to 2000
The mean waiting time	in type 1 place	60 minutes
	in type 2 place	30 minutes
	in type 3 place	15 minutes
WAN Propagation Delay		100ms
Lan Internal Delay		5ms

load scheduling approaches respectively. The results are shown in Fig.3.

Fig.3(a) shows the average service time of different approaches. We can see that the average service time for each approach increases with the number of mobile devices. At the beginning, the highest of the average service time is Bandwidth-based approach, which is 1.4 seconds, followed by Utilization-based approach is 1.1 seconds, and Fuzzy-based and RL-based approaches are both around 0.8 seconds. When the number of devices increases to 2000, the average server of Utilization-based approach is the highest, reaching 4.2 seconds, followed by Fuzzy-based approach is 3.1 seconds, and the third is Bandwidth-based approach is 2.5 seconds. The RL-based approach is 2.9 seconds, which is the lowest average service time among these approaches. Compared with other approaches, our proposed RL-based approach achieves the lowest average service time during the whole experiments, which shows that our proposed RL-based approach makes the task distribution more balanced.

Fig.3(b) shows the average VM Utilization of different approaches. At the beginning, the lowest of the average VM utilization is Bandwidth-based approach, and the average VM utilization of each approach increases with the number of devices. When the number of devices increases to 1000, the lowest of the average VM utilization is replaced by RL-based approach, and stayed there until the end, reaching about 24%.

Fig.3(c) shows the failed tasks rate of different approaches. As can be seen from the figure, when the number of mobile devices changes from 200 to 1200, the failed tasks rate of all approaches can be kept below 1% and relatively stable. When the number of mobile devices changes from 1400 to 2000, the failed tasks rate of each approach increases with the number of mobile devices. Among them, the Utilization-based approach has the fastest growing, and reaches 15.1% at 2000 devices, followed by Bandwidth-Based approach, reaching 5.9%. The growth of Fuzzy-Based approach is relatively slow and reaches 4%. Compared with other approaches, our proposed RL-based approach achieves the lowest failed task rate of only 2.9%.

Although we can see the failed tasks rate of different approaches from Fig.3(c), we don't know the reasons of task failure from figure. Next, we will analyze the reasons for task failure. In our simulations, Task failure is due to the following reasons: 1)The lack of VM capacity. If the VM utilization is too high, the server does not have enough VM capacity to meet new coming tasks, which make task waiting time too long to failure.

2) The mobility of users. If the user leaves his location before getting the response of the previously requested tasks, the tasks will fail because the user is out of the previous WLAN coverage and cannot receive the response from servers.

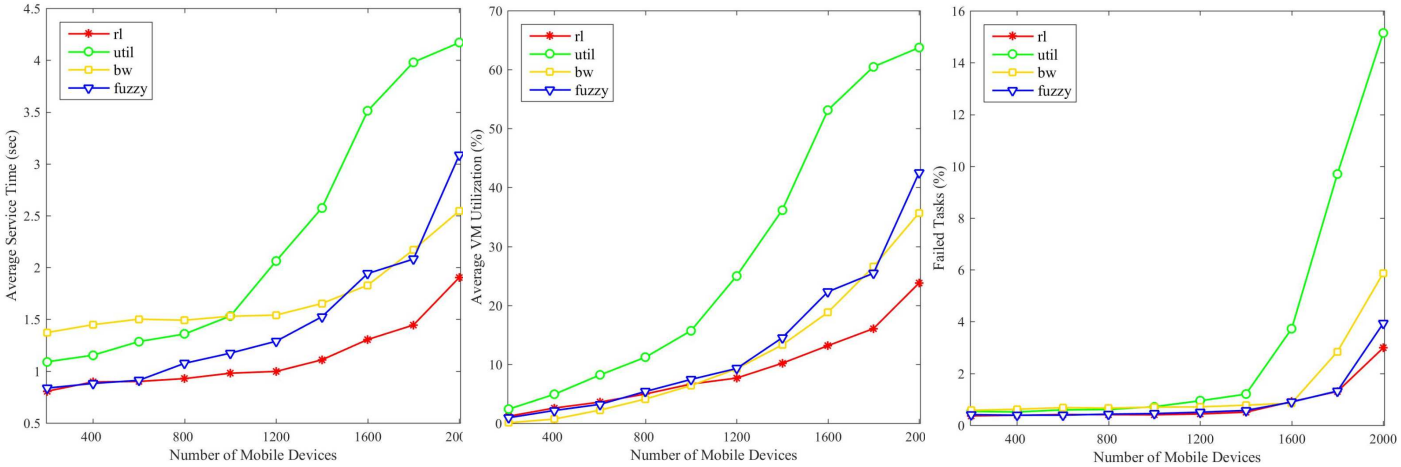
3) The lack of network resource. If a large number of users simultaneously use the same network resources (such as WLAN, MAN or WAN), it will cause insufficient network bandwidth or even network congestion, resulting in packets loss, and eventually lead to task failure.

The reasons for tasks failure are shown in Fig.4. It can be clearly seen that, In the fuzzy-based approach, the main reason for task failure is due to lack of VM capacity, followed by lack of MAN resource. In the Bandwidth-Based approach, the main reason for task failure is due to lack of WAN resource. However, when the number of users reaches 1600, the mobility of users becomes the main reason for task failure. In the Utilization-based approach, the main reason for task failure is due to mobility of users and lack of WAN resource when the number of users reaches 1600. In our RL-based approach, the main reason for task failure is due to lack of MAN resource, followed by VM capacity and mobility of users. In all approaches, there is almost no transmission failure in the WLAN, so the reason of the tasks failure due to the WLAN can be negligible.

Given the above observation, our proposed RL-based workload Scheduling approach has the best performance in aspects of service time, and VM utilization, failed tasks rate among these four approaches. Especially, When the number of users is large (above 1600), our proposed RL-based approach still can keep the lowest task failure rate, which shows that the task allocation is more reasonable and effective, and ensures the stability of the system

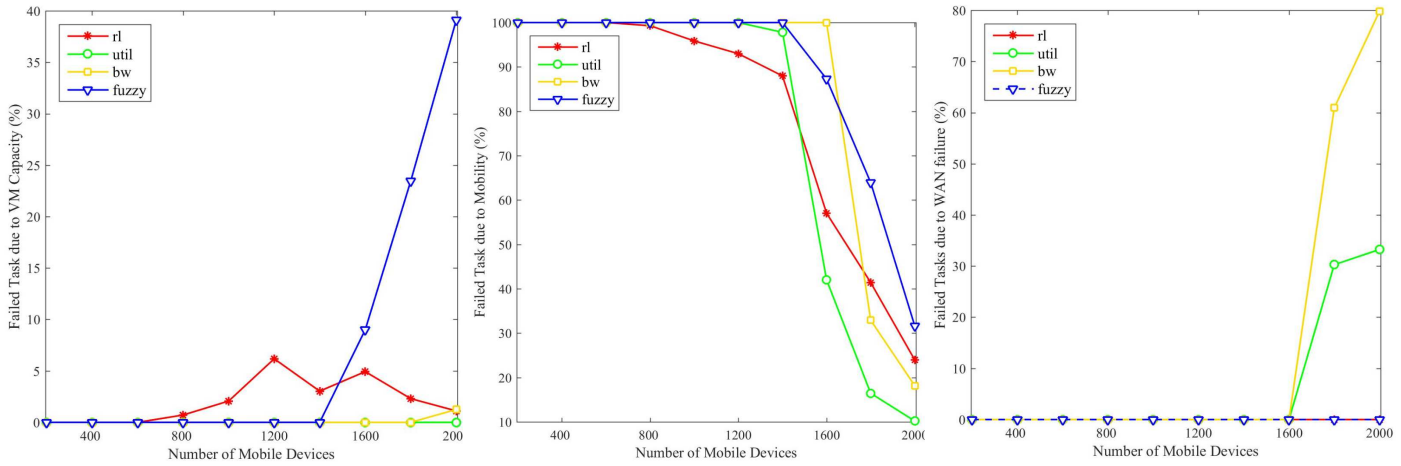
VI. CONCLUSION

In this paper, we have investigated workload scheduling in edge computing, aiming at balance the workload, reduce the service time and minimize failed task rate. Considering that edge computing system is a very dynamic environment and affected by several factors. We analyze system model, which includes task model, network model, computational model, according to the Multi-tier edge computing architecture, and formulate the workload scheduling problem based on system model as a hard and online problem. To deal with the challenges of workload scheduling problem, we proposed a reinforcement learning-based approach, which can learn from the previous actions and achieve best scheduling in the absence

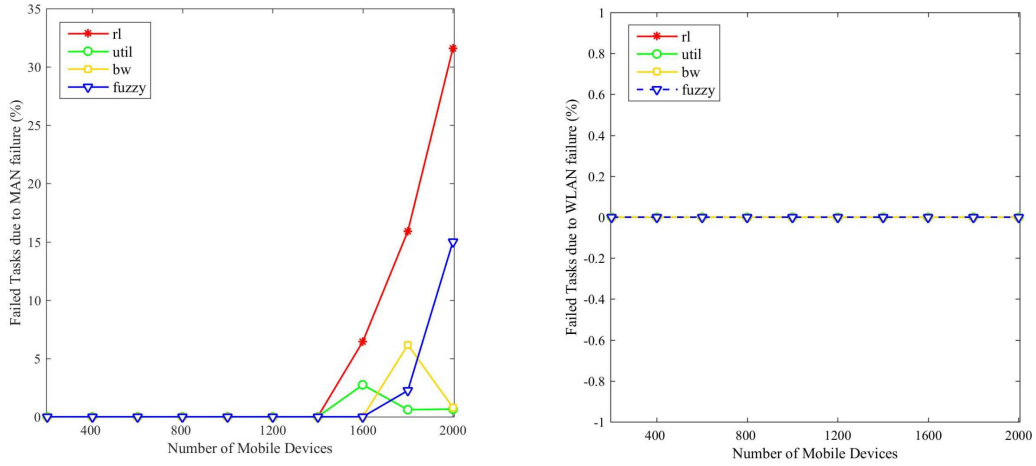


(a) Average Service time. (b) Average VM Utilization. (c) Failed Tasks.

Fig. 3. Comparison of the three aspects: (a) Average Service Time. (b) Average VM Utilization. (c) Failed Tasks.



(a) Failed task duo to VM Capacity. (b) Failed Task duo to Mobility. (c) Failed Task duo to WAN failure.



(d) Failed task duo to MAN failure. (e) Failed Task duo to WLAN failure.

Fig. 4. The Reasons for Task Failure.

of a mathematical model of the environment. Simulation results show that our proposed approach achieves the best performance in aspects of service time, virtual machine utilization, and failed tasks rate compared with other approaches. Our reinforcement learning-based approach can provide an efficient solution to the workload scheduling problem in edge computing.

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017, doi: 10.1109/COMST.2017.2745201.

[2] Z. Ning, P. Dong, X. Kong and F. Xia, "A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things," in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4804-4814, June 2019, doi: 10.1109/JIOT.2018.2868616.

[3] H. Li, G. Shou, Y. Hu and Z. Guo, "Mobile Edge Computing: Progress and Challenges," 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Oxford, UK, 2016, pp. 83-84, doi: 10.1109/MobileCloud.2016.16.

[4] Z. Sanaei, S. Abolfazli, A. Gani and R. Buyya, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges," in IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 369-392, First Quarter 2014, doi: 10.1109/SURV.2013.050113.00090.

[5] M. Satyanarayanan, "The Emergence of Edge Computing," in Computer, vol. 50, no. 1, pp. 30-39, Jan. 2017, doi: 10.1109/MC.2017.9..

[6] Cisco, "Cisco global cloud index: Forecast and methodology, 2016-2021," White Paper, 2018.

[7] W. Shi and S. Dustdar, "The Promise of Edge Computing," in Computer, vol. 49, no. 5, pp. 78-81, May 2016, doi: 10.1109/MC.2016.145..

[8] T. Zheng, J. Wan, J. Zhang, C. Jiang and G. Jia, "A Survey of Computation Offloading in Edge Computing," 2020 International Conference on Computer, Information and Telecommunication Systems (CITS), Hangzhou, China, 2020, pp. 1-6, doi: 10.1109/CITS49457.2020.9232457.

[9] M. Caprolu, R. Di Pietro, F. Lombardi and S. Raponi, "Edge Computing Perspectives: Architectures, Technologies, and Open Security Issues," 2019 IEEE International Conference on Edge Computing (EDGE), Milan, Italy, 2019, pp. 116-123, doi: 10.1109/EDGE.2019.00035.

[10] L. Zhao et al., "Vehicular Computation Offloading for Industrial Mobile Edge Computing," in IEEE Transactions on Industrial Informatics, doi: 10.1109/TII.2021.3059640.

[11] Y. Yin, W. Zhang, Y. Xu, H. Zhang, Z. Mai and L. Yu, "QoS Prediction for Mobile Edge Service Recommendation With Auto-Encoder," in IEEE Access, vol. 7, pp. 62312-62324, 2019, doi: 10.1109/ACCESS.2019.2914737.

[12] A. Al-Dulaimy, J. Taheri, A. Kassler, M. R. Hoseiny Farahabady, S. Deng and A. Zomaya, "MULTISCALER: A Multi-Loop Auto-Scaling Approach for Cloud-Based Applications," in IEEE Transactions on Cloud Computing, doi: 10.1109/TCC.2020.3031676.

[13] L. P. Qian, B. Shi, Y. Wu, B. Sun and D. H. K. Tsang, "NOMA-Enabled Mobile Edge Computing for Internet of Things via Joint Communication and Computation Resource Allocations," in IEEE Internet of Things Journal, vol. 7, no. 1, pp. 718-733, Jan. 2020, doi: 10.1109/JIOT.2019.2952647.

[14] H. Flores et al., "Large-scale offloading in the Internet of Things," 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kona, HI, 2017, pp. 479-484, doi: 10.1109/PERCOMW.2017.7917610..

[15] H. Gao, Y. Xu, Y. Yin, W. Zhang, R. Li and X. Wang, "Context-Aware QoS Prediction With Neural Collaborative Filtering for Internet-of-Things Services," in IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4532-4542, May 2020, doi: 10.1109/JIOT.2019.2956827.

[16] L. Zhao, K. Yang, Z. Tan, X. Li, S. Sharma and Z. Liu, "A Novel Cost Optimization Strategy for SDN-Enabled UAV-Assisted Vehicular Computation Offloading," in IEEE Transactions on Intelligent Transportation Systems, doi: 10.1109/TITS.2020.3024186.

[17] C. Sonmez, A. Ozgovde and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of Edge Computing systems," 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, 2017, pp. 39-44, doi: 10.1109/FMEC.2017.7946405..

[18] J. Zhang et al., "Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks," in IEEE Internet of Things Journal, vol. 5, no. 4, pp. 2633-2645, Aug. 2018, doi: 10.1109/JIOT.2017.2786343.

[19] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini and S. Cretti, "Foggy: A Platform for Workload Orchestration in a Fog Computing Environment," 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, China, 2017, pp. 231-234, doi: 10.1109/CloudCom.2017.62..

[20] A. Anas, M. Sharma, R. Abozariba, M. Asaduzzaman, E. Benkhelifa and M. N. Patwary, "Autonomous Workload Balancing in Cloud Federation Environments with Different Access Restrictions," 2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), Orlando, FL, 2017, pp. 636-641, doi: 10.1109/MASS.2017.68.

[21] X. Ma, A. Zhou, S. Zhang and S. Wang, "Cooperative Service Caching and Workload Scheduling in Mobile Edge Computing," IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, Toronto, ON, Canada, 2020, pp. 2076-2085, doi: 10.1109/INFOCOM41043.2020.9155455..

[22] Sonmez, Cagatay , A. Ozgovde , and C. Ersoy . "Fuzzy Workload Orchestration for Edge Computing." IEEE Transactions on Network & Service Management (2019):1-1.

[23] Q. He et al., "A Game-Theoretical Approach for User Allocation in Edge Computing Environment," in IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 3, pp. 515-529, 1 March 2020, doi: 10.1109/TPDS.2019.2938944.

[24] W. Hussain, F. K. Hussain, O. Hussain, R. Bagia, E. Chang and A. Romanovsky, "Risk-based framework for SLA violation abatement from the cloud service provider's perspective," in The Computer Journal, vol. 61, no. 9, pp. 1306-1322, Sept. 2018, doi: 10.1093/comjnl/bxx118.

[25] H. Gao, C. Liu, Y. Li and X. Yang, "V2VR: Reliable Hybrid-Network-Oriented V2V Data Transmission and Routing Considering RSUs and Connectivity Probability," in IEEE Transactions on Intelligent Transportation Systems, doi: 10.1109/TITS.2020.2983835.

[26] W. Hussain and O. Sohaib, "Analysing Cloud QoS Prediction Approaches and Its Control Parameters: Considering Overall Accuracy and Freshness of a Dataset," in IEEE Access, vol. 7, pp. 82649-82671, 2019, doi: 10.1109/ACCESS.2019.2923706.

[27] Z. Liu, H. Zhang, B. Rao and L. Wang, "A Reinforcement Learning Based Resource Management Approach for Time-critical Workloads in Distributed Computing Environment," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 252-261, doi: 10.1109/BigData.2018.8622393..

[28] L. Liu and U. Mitra, "On Sampled Reinforcement Learning in Wireless Networks: Exploitation of Policy Structures," in IEEE Transactions on Communications, vol. 68, no. 5, pp. 2823-2837, May 2020, doi: 10.1109/TCOMM.2020.2974216..

[29] Y. Duan, Q. Duan, G. Hu and X. Sun, "Refinement from service economics planning to ubiquitous services implementation," 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), Okayama, Japan, 2016, pp. 1-6, doi: 10.1109/ICIS.2016.7550909.

[30] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Softw. Pract. Exp., vol. 41, no. 1, pp. 23–50, Jan. 2011.

Figures

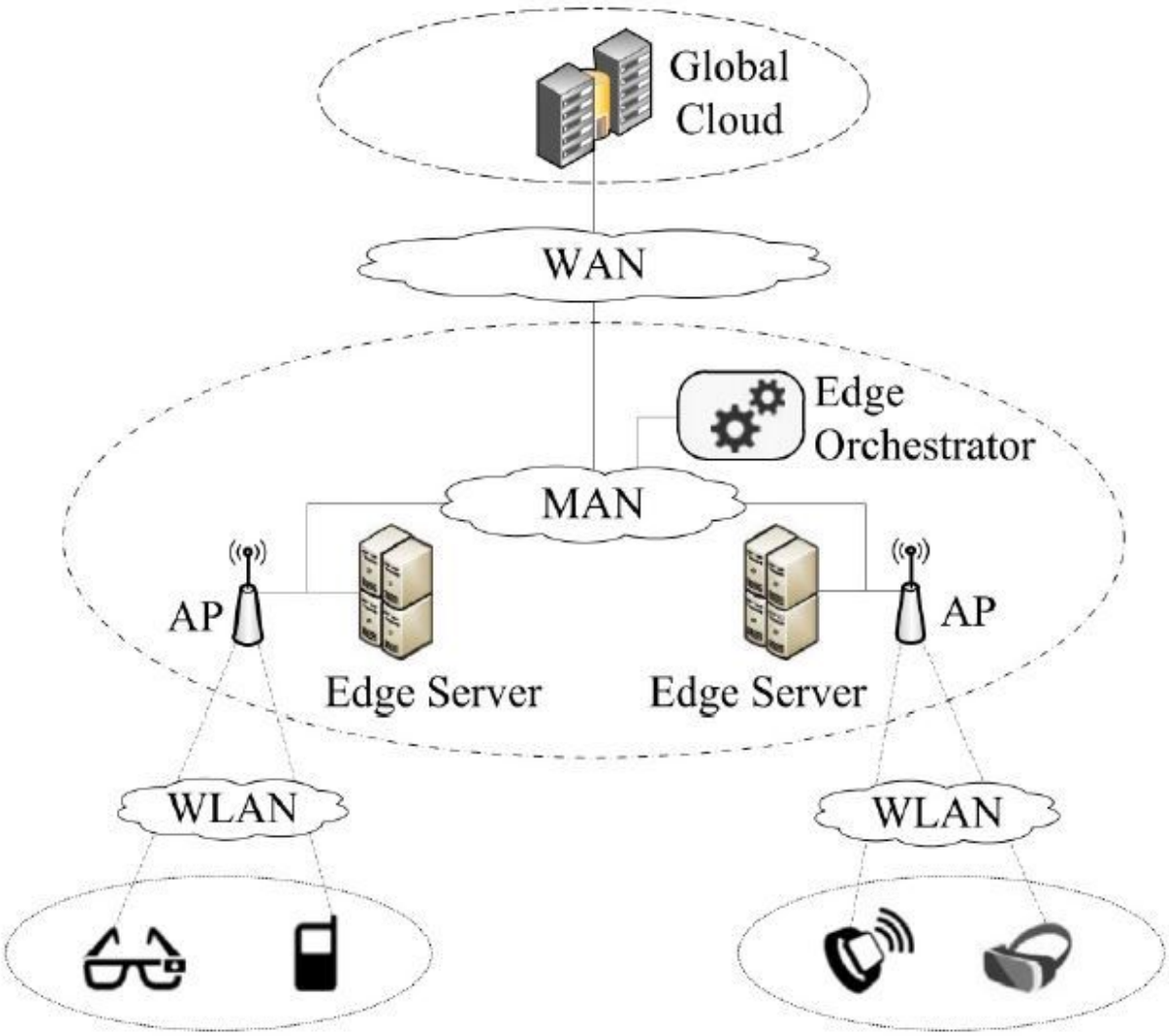


Figure 1

Multi-tier edge computing architecture with edge orchestrator.

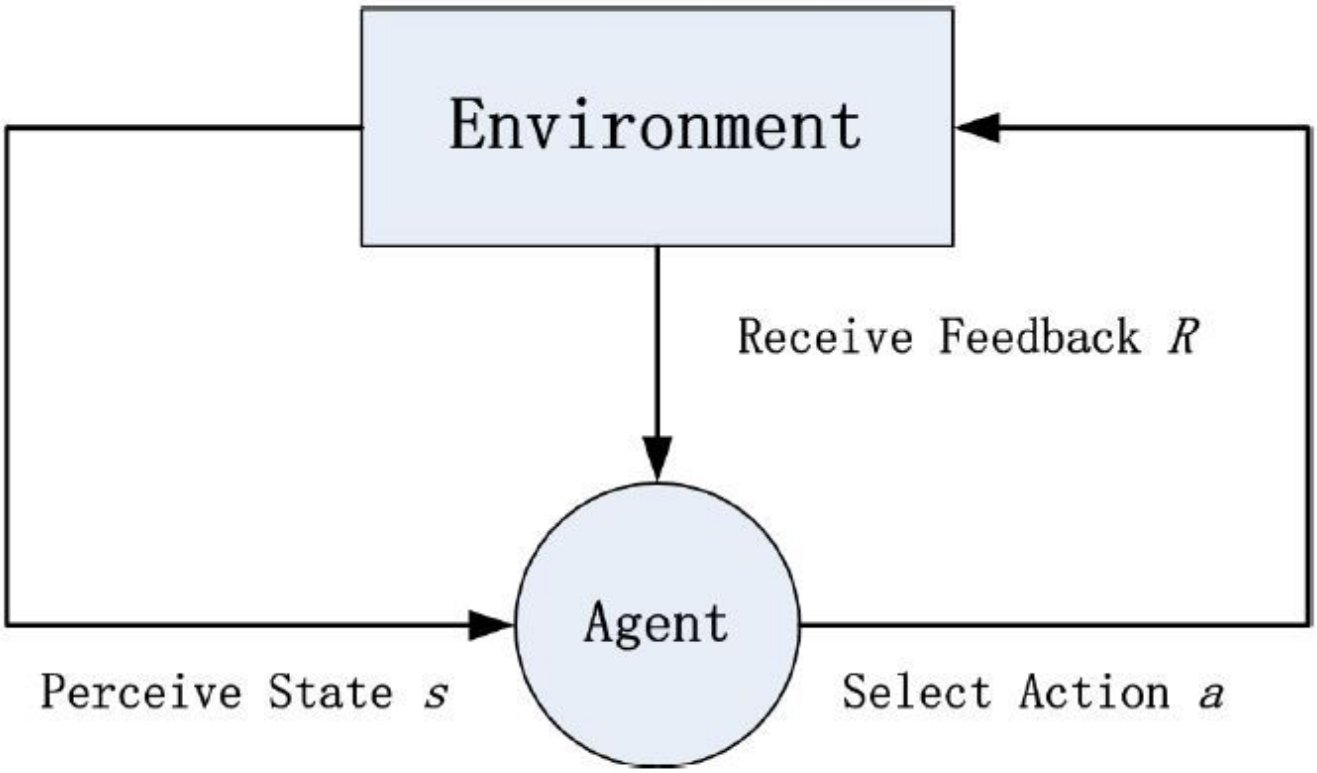
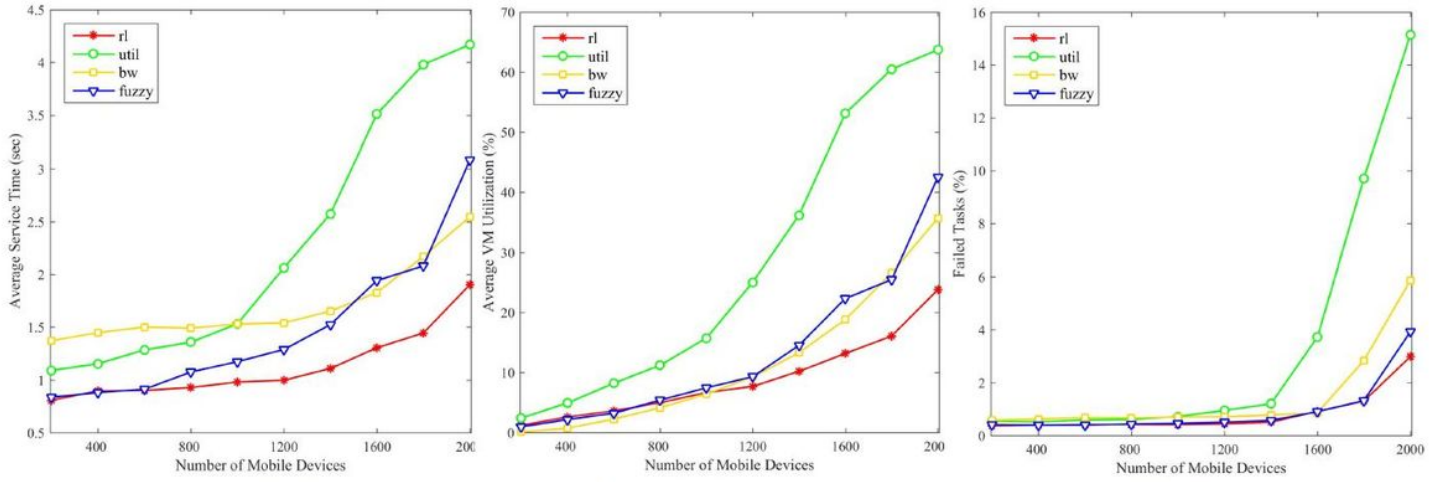


Figure 2

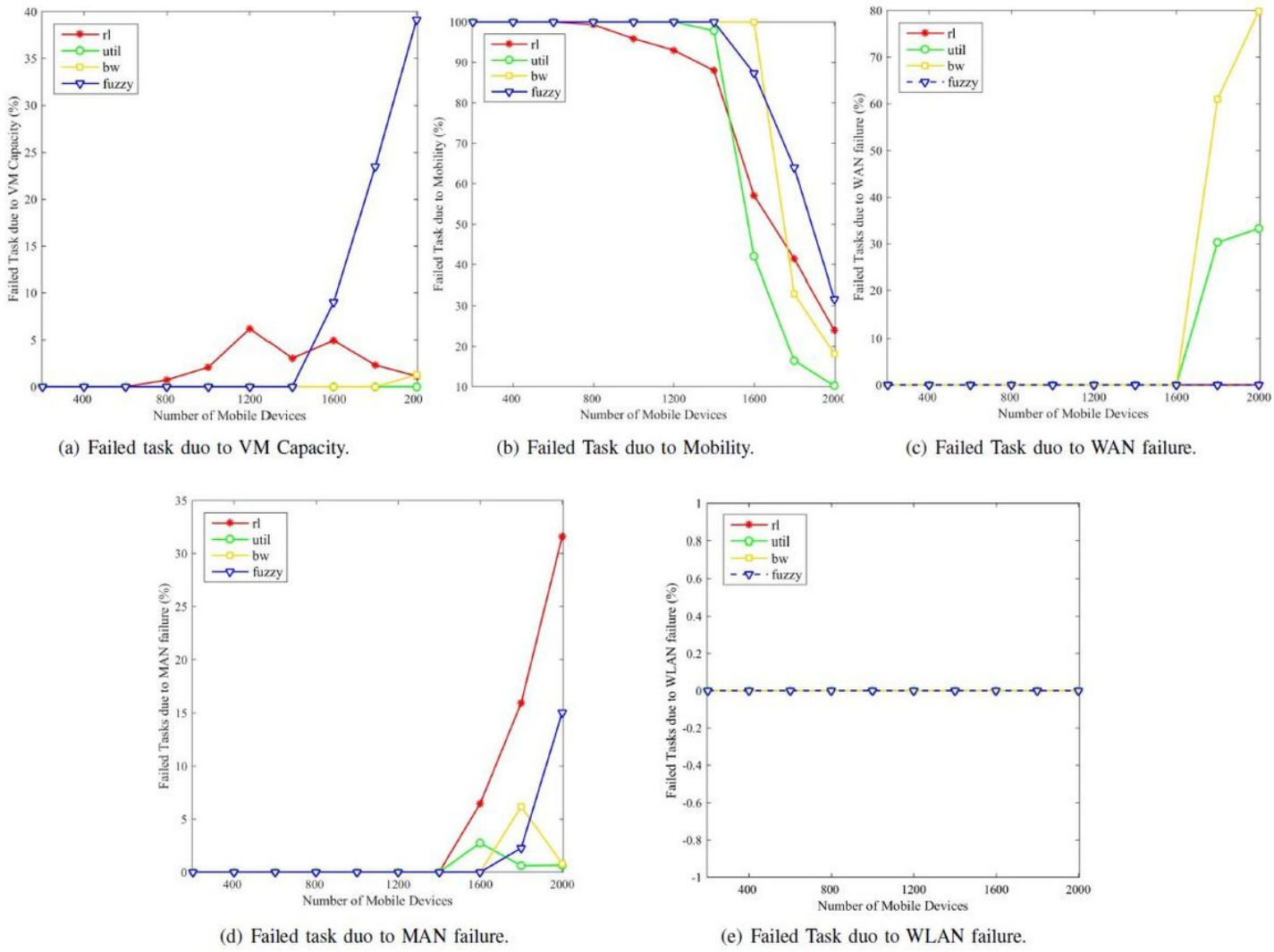
The Reinforcement Learning Paradigm.



(a) Average Service time. (b) Average VM Utilization. (c) Failed Tasks.

Figure 3

Comparison of the three aspects: (a) Average Service Time. (b) Average VM Utilization. (c) Failed Tasks.



(a) Failed task duo to VM Capacity.

(b) Failed Task duo to Mobility.

(c) Failed Task duo to WAN failure.

(d) Failed task duo to MAN failure.

(e) Failed Task duo to WLAN failure.

Figure 4

The Reasons for Task Failure.