Computing (2024) 106:1051–1080
https://doi.org/10.1007/s00607-022-01141-x

SPECIAL ISSUE ARTICLE

An intelligent resource management method in SDN based fog computing using reinforcement learning

Milad Anoushee[1] · **Mehdi Fartash**[1] · **Javad Akbari Torkestani**[1]

Received: 27 October 2021 / Accepted: 19 December 2022 / Published online: 3 January 2023

Abstract

Nowadays, cloud computing faces growing challenges, furthermore, responding to time-sensitive requests in the traditional cloud computing model is one of the major challenges, considering the growth of the Internet of Things. Recently, the powerful fog computing paradigm has been considered for answering these challenges. But a major challenge in fog computing is managing limited FN resources for correctly responding to IoT requests in environments with heterogeneous latency requirements. This study presented a new resource management framework utilizing a software defined network (SDN) architecture and enhanced reinforcement learning methods. This new framework aims to make optimal use of limited FN resources while satisfying the low-latency requirements of IoT applications. Since SDN is the proper choice to support this intelligent distributed structure, an SDN-based fog architecture was proposed. Moreover, FN must allocate its limited and valuable resources intelligently in a heterogeneous IoT environment with different latency requirements. Consequently, this study formulated the problem of resource allocation in fog computing in the form of a Markov decision process (MDP). The study used different reinforcement learning (RL) techniques to solve the MDP problem. Simulation results in different IoT environments with heterogeneous latency requirements corroborate that RL methods achieve the best possible performance regardless of the IoT environment.

Keywords Fog computing · Resource allocation · Delay-sensitive · IoT · Reinforcement learning · Software defined network

Mathematics Subject Classification 90C40

✉ Milad Anoushee
m.anooshee@BSI.ir

1 Department of Computer Engineering, Islamic Azad University, Arak Branch, 3rd km of Khomein road, Imam Khomeini Square, Arak, Markazi Province 38361-1-9131, Iran

1 Introduction

Over the past few decades, cloud computing has been developed as an important trend for transferring computing resources, controlling and storing information in geographically distributed data centers. However, cloud computing is facing growing challenges such as high unpredictable communication latency, privacy gaps, network load connected to the end devices, and costs related to head-on connections [1, 2]. Most of these problems are due to the high physical distance between cloud service provider's data centers and the end-users. This geographical distant results in high latency and reduced quality of service (QoS) problems. These problems act as barriers for time-sensitive system requests, creating challenges in supporting real-time processing and providing fast response times for end-users [1].

Using middleware between end nodes and the cloud can help solve some of these challenges. Consequently, different studies proposed and studied middlewares such as Cloudlets,[1] MEC,[2] Micro Data Centers,[3] and other similar solutions such as Nano-data centers, which can be under the umbrella of fog computing or edge computing [4–6].[4]

Cisco introduced fog computing in 2014 to expand cloud computing to the network edge. It is an entirely virtual platform that presents computing, storage, and network services between end IoT users and traditional cloud computing data centers [5]. Fog computing hopes to decentralize computing and integrate cloud data centers and heterogeneous edge devices to perform ubiquitous distributed computing [7]. Moreover, light-weight or time-sensitive IoT environment tasks can be processed in the fog base station. In contrast, heavyweight tasks or any other thing that is not possible in the fog layer for any reason is performed in cloud computing platforms.

Although fog computing is a promising method for solving the problems of cloud computing and current networks, some problems remain unsolved. Most importantly, there is a need for an intelligent distributed platform at the edge to manage the computational, network, and storage resources at the fog layer. Currently, uncertainties caused by the relation between future requests and limited fog node resources cause multiple obstacles to proper decision-making in resource allocation.

Moreover, if all requests are forwarded from the IoT layer to the cloud, the possibility of responding to requests with low latency requirements will be challenged. On the other hand, as an alternative solution, fog computing faces resource scarcity problems in responding to all requests. Therefore, intelligent resource management at the fog layer can be a solution to this problem.

---

[1] The concept of cloudlet, initiated by Carnegie Mellon University (CMU) is similar to MDC, since both are a small scaled virtualized data center to serve users near the edge in a distributed fashion [3].

[2] Mobile Edge Computing.

[3] MDCs, initiated by Microsoft are small scaled versions of data centers to extend the hyperspace cloud data centers. MDCs aim to provide small size data centers extending the offered services of the cloud near to the end users.

[4] The term Edge computing and Fog computing seem interchangeable, and for a fact, they do share some key similarities. Both Edge and Fog computing systems shift processing of data closer to the source of data generation. The main focus of this shift is to reduce the amount of data sent to the cloud. This shift helps in decreasing latency and thereby improving system response time, especially in remote mission-critical applications.

Resolving the problem is critical for IoT applications that cannot tolerate such latencies. There are tasks with different latency requirements in IoT applications, some of which are more sensitive to latency while others can handle more [8, 9]. Therefore, FN must allocate its limited and valuable resources in heterogeneous IoT environments with different latency requirements intelligently.

1.1 Contributions

The current study presented a new framework for resource allocation in SDN-based fog using reinforcement learning methods. The framework aimed to effectively use limited FN resources while providing the low-latency requirements of IoT applications. Moreover, an SDN-based architecture was used to turn the resource allocation problem into a decision-making process based on reinforcement learning. Consequently, this decision-making could increase the total usefulness of serviced requests at the fog layer. In addition, the proposed algorithm formulated the resource allocation problem as a Markov decision process, which allows the controller to make the best decision in uncertain situations. Since dynamic changes happen in input requests and resource situations, the system cannot precisely predict the transition probabilities and rewards. Therefore, the model-less RL methods such as SARS, Q-Learning, and Monte Carlo were used to learn optimal policies and make correct decision-making.

The key contributions of this paper are listed as follows:

1. Provide the possibility of using all fog layer resources to respond to time-sensitive requests in the shortest possible time.
2. A special multi-layered architecture was presented (Fig. 1) based on SDN using reinforcement learning algorithms.
3. Using reinforcement learning enabled responsible controller to act as an intelligent agent and choose the optimal method for responing to a request without a required prior knowledge of the environment.

The rest of this paper is organized as follows. The next section considers the related studies. After that, architecture and system modeling method are introduced. The problem formulation proposed for solving the resource allocation problem is presented in the third section, along with introducing the reinforcement learning method and related algorithms. Then the simulation results are presented in the fourth section, and the conclusion is in the fifth section. In addition, a list of notation and abbreviations used throughout the paper is provided in Table 1.

1.2 Fog computing and SDN-based architecture

SDN technology was used in the proposed model of this study to increase fog computing efficiency, network scalability, and programmability. The effects of SDN on fog computing were examined in different studies. For example, Bakhtir et al. [10] survey presented a complete description of how fog computing can use SDN to solve its challenges. Their study thoroughly investigated the problems of fog computing in different environments and analyzed them in a categorized fashion. Bakhtir et al.

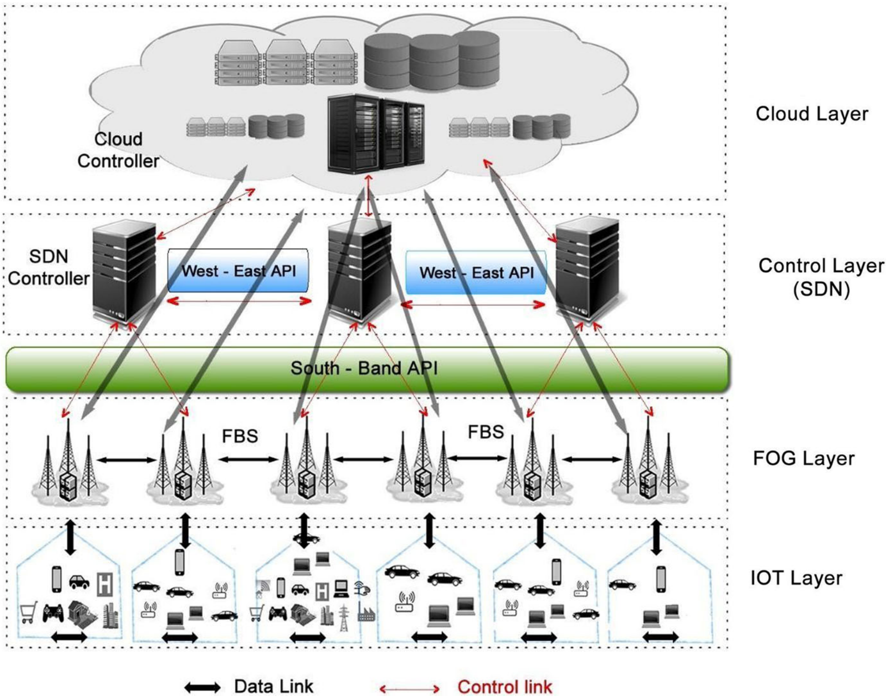

Fig. 1 Multi-Controller Flat SDN-based Fog Architecture: According to the figure, this environment includes multiple controllers. The controller that receives the service requests from one of its FBSs at the moment t is named the responsible controller, which becomes responsible for deciding how to provide the requested resources

[10] found SDN a newfound method of facing these challenges. Moreover, they presented a transparent cooperation model for SDN-based fog computing using practical architecture. In addition, they showed the usage of SDN-related mechanisms in fog computing infrastructure.

In another study, Gupta et al. [11] presented an SDN-based fog computing system named SD-Fog. The proposed system performed the intelligent QoS by managing and controlling the flow between services and orchestration. They implemented network function virtualization (NFV) and SDN technologies to show the effectiveness of this proposed system in their case study on smart homes.

In addition, Sun et al. [12] presented edge IoT with a hierarchical fog and cloud computing structure to effectively manage IoT device data at the network edge. In this structure, the SDN-based cellular core was located above the fog servers to transfer information between these servers. The cooperation between fog computing and SDN created efficiency in IoT data stream collection, categorization, and analysis.

As one of the earliest studies, Truong et al. [13] presented an integrated fog computing and SDN architecture for vehicular ad hoc networks. In their studty, the SDN controller was placed between the fog and cloud layers for fog orchestration and

Table 1 A Summary of notation and abbreviations

Notation	Description
IoT	Internet of Things
SDN	Software Defined Network
MDP	Markov Decision Process
RL	Reinforcement Learning
FN	Fog Node
MEC	Mobile Edge Computing
QL	Q-learning
MC	Monte Carlo
DDQL	Double Deep Q-Learning
FBS	Fog-based Base Station
RB	Resource Block
SDNC	Software Defined Network Controller
EN	End Node
QoS	Quality of Service
u_t	User utility at time t
δ, β, σ	Parameters for utility computation
a_t	Action taken at time t
$a^{'}$	Action from successor state
$a*$	Optimal action
S_t	State at time t
$S^{'}_{Local}$	Successor state when service is provided with FBSs controlled by the responsible SDNC
$S^{'}_{Fog}$	Successor state when service is provided with FBSs controlled by neighboring SDNCs
$S^{'}_{Cloud}$	Successor state when service is provided with cloud resource
R_t	Reward received at time t
R_{Local}	Reward received in case of receiving service in the fog with local sources
R_{Fog}	Reward received in case of receiving service in the fog with non-local sources
R_{Cloud}	Reward received in case of receiving service in the cloud
FRB_{Local}	The total number of Free RBs associated with FBSs controlled by the responsible SDNC at time t
FRB_{Fog}	The total number of Free RBs associated with FBSs controlled by neighboring SDNCs at time t
NRB_{Local}	The total number of RBs associated with FBSs controlled by the responsible SDNC
NRB_{Fog}	The total number of RBs associated with FBSs controlled by neighboring SDNCs
n_{Local}	Occupied RBs associated with FBSs controlled by the responsible SDNC
n_{Fog}	Occupied RBs associated with FBSs controlled by neighboring SDNCs
A	Set of actions

Table 1 continued

Notation	Description
R	Set of rewards
u_h	Threshold for defining high utility
u_l	Threshold for defining low utility
$v_\pi(s)$	State value function of state s following policy π
N_i	Total number of neighbor controls
$q_\pi(s, a)$	Action value function of state s with action a following policy π
$v_*(s)$	Optimal state value function π
$q_*(s, a)$	Optimal action value function π
G_t	Cumulative reward from time t to terminal state
γ	Discount rate
π	Policy for taking actions
$\mathbb{E}_u$	Expectation with respect to the utilities u in the IoT environment
ϵ	Probability of random action
α	Learning rate
θ	Penalty of idle time

resource management. They also showed the benefits of this integrated system by presenting two scenarios.

In addition, Yong et al. [14] employed SDN technology to operationalize the connection between the fog and the cloud, and improve service quality. Lin et al. [15] introduced a distributed network architecture in another study. They benefited SDN in vehicular networks with scalable network management and support of intelligent data computing policies in fog computing. Besides, Liang et al. [16] presented an integrated architecture for software defined and virtualized radio access networks with fog computing. They suggestinged using a software as a Service service (SaaS) named Open Pipe, which enables network layer virtualization.

1.3 Resource management at the fog

Task offloading and resource management is another subject in fog computing that has received significant attention. For example, resource allocation was studied based on cooperative edge computing to achieve ultra-low latency in fog radio access networks (FRAN) [17–19]. Sahni et al. [17] proposed a meshing algorithm for edge computing to distribute decision-making tasks among the edge devices instead of the cloud server.

In another study, heterogeneous F-RAN structures such as small cells and macro base stations were considered to present an F-RAN node selection algorithm for proper heterogeneous resource allocation [18, 19]. Moreover, Mukherjee et al. [20] studied computational offloading for multiple tasks with different latency needs related to users. In the scenario of this paper, the end-user offloaded the task data onto its responsible fog node.

Zhou et al. [21] proposed a solution for coping with different challenges of computational task offloading onto resources related to surrounding vehicles, including the lack of effective motivations and assignment mechanisms. Besides, Zhang et al. [22] considered a particular fog computing network consisting of a set of data service operators (DSO). Each DSO controlled a set of fog nodes in their study for presenting the required data services to its subscribers.

In another study, Gu et al. [23] analyzed the radio and computational resource distribution problem for optimizing system performance and improving user satisfaction. Their proposed method aimed to show a distributed approach to the shared resource allocation problem while using the effective SPA- (S, P) algorithm to find a stable solution to the SPA problem. Their method used a matching game framework, especially a student project allocation (SPA) game, instead of continuous centralized optimization for this aim.

1.4 Fog computing and reinforcement learning

Reinforcement learning methods have been widely used in fog computing due to their benefits in solving resource management and load balancing problems. Other machine learning methods, such as deep learning, are also used to solve the challenges of fog computing [24–26]. However, reinforcement learning techniques generally have broader applications in the literature in this field. This wider use is model-free reinforcement learning doesn't need environmental cognition.

For example, Dutreilh et al. [27] were one of the first attempts to use reinforcement learning algorithms. They used the reinforcement learning technique for automatic resource allocation in the cloud, shaping a proper and dynamic model for resource allocation, which is essential in cloud computing. Furthermore, Le and Tham [28] proposed an offloading application based on deep reinforcement learning for ad-hoc mobile cloud users.

In addition, Gao et al. [29] considered a multi-user mobile edge computing (MEC) system. This system allowed users to perform their computational offloading using wireless channels to a MEC server. Moreover, the total costs of latency and energy consumption for all users were considered the optimization objective. Additionally, the two decisions, including offloading and computing resource allocation, were optimized to minimize the overall system cost. For this purpose, an optimization framework based on reinforcement learning (Q-learning and deep reinforcement learning) was proposed for resource allocation in wireless MEC. The simulation results showed that the proposed method significantly reduced the total cost compared to other base methods.

In another study, Parent et al. [30] suggested reinforcement learning for distributed static load balancing in data-centered applications in a heterogeneous fog computing environment. Moreover, Gazori et al. [31] focused on the timing of fog-based IoT application tasks to minimize long-term service delay, computational costs with limited resources, and execution time. The reinforcement learning method and the double deep Q-learning (DDQL) algorithm were also used to solve these problems. Their evaluation showed that the proposed method performed better than other base algorithms in service latency, computational cost, energy consumption, and task performance while managing the single breaking point and load management challenges.

Besides, Liu et al. [32] presented a trade-off between consumed energy and service latency in mobile vehicular networks considering network traffic and the computational load caused by this traffic. Their study created a cost-minimization problem using the Markov decision process (MDP). In addition, they proposed dynamic reinforcement learning and deep dynamic programming algorithms to solve the offloading decision problem.

In the current study, Table 2 compares some reviewed methods from the point of view of using or not using SDN, the use of artificial intelligence methods and the algorithm used, and their goals. This study intended to present a novel resource management method to improve service quality by introducing SDN-based fog computing architecture with multiple flat-shaped controllers. Furthermore, this study used reinforcement learning algorithms for optimized decision-making regarding precious and limited fog layer resources.

2 Architecture and system model

This study presented SDN-based fog computing architecture (as shown in Fig. 1) to solve the previously mentioned problems. In this architecture, which is a multi-controller flat architecture,[5] the environment is divided into multiple partitions. Each partition includes at least one local Fog-based Base Station (FBS) managed and controlled by the responsible SDN controller.

Each IoT device in a singular partition with different access points (Wi-Fi, WiMAX, Cellular) can direct connection and service requests to the partition's local FBS. The service request information is sent using the local FBS to the responsible controller of that partition for analysis and determines the serving method. Therefore, end-nodes indirectly connect to the controller.

Moreover, the leaving and joining of IoT devices in environments involving mobile IoT devices that move to different partitions over time are recorded by their local FBS at the responsible controller to show the user density. Therefore, every controller has a partition view of their local network situation [37].

In addition, the partition views include the request traffic, the local resource status, and available resources of the neighboring controllers that can be updated and shared between neighbor controllers in certain periods based on network traffic. Fog base stations, in addition to network facilities, are equipped with caching, computing, and signal processing capabilities. This equipping is to optimize communication bandwidth consumption between FBS and the cloud and overcome the challenge of increasing IoT devices and low latency requirements. However, their resources are limited, so efficient use is essential.

[5] The main point in using multiple controllers is the multi-controller architecture. After reviewing the literature, we came to the conclusion that fundamental multi-controller architecture can be categorized into flat designs and hierarchical designs. In flat designs, the network is organized into multiple domains that are controlled by controllers situated in their own local view. The controllers connect with each other using their east-west bound interfaces to gain a global view [38].

Table 2 Comparison between some solutions presented in fog computing

Row	Article nam	Use SDN	Architecture presented	Use AI methods/algorithm	Objectives
1	Managing Fog Networks using Reinforcement Learning Based Load Balancing Algorithm [2]	Yes	No	Yes/ Reinforcement learning (Q-learning algorithm)	Load Balancing with minimize the processing time and the overall overloading probability
2	Integrated architecture for software defined and virtualized radio access networks with fog computing [16]	Yes	Yes	No	Proposed an integrated architecture for software defined and virtualized RANs with fog computing
3	Reinforcement Learning for Adaptive Resource Allocation in Fog RAN [41]	Yes	No	Yes/ Reinforcement learning (Q-learning, Expected SARSA, SARSA and Monte Carlo)	Resource allocation problem in Fog RAN for IoT services with heterogeneous latency requirements
4	Deep Reinforcement Learning based Computation Offloading and Resource Allocation for MEC [29]	No	No	Yes/Reinforcement Learning (Deep Reinforcement Learning, Q-learning)	MEC computation resource allocation
5	Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach [31]	No	No	Yes/Reinforcement Learning (Double Deep Q-Learning)	Minimizing long-term service delay and computation cost under the resource and deadline constraints
6	A Distributed Mobile Fog Computing Scheme for Mobile Delay-Sensitive Applications in SDN-Enabled Vehicular Networks [15]	Yes	Yes	No	Proposed a distributed mobile fog computing scheme for scheduling the mobile delay-sensitive applications in vehicular networks
7	Enabling intelligence in fog computing to achieve energy and latency reduction [26]	Yes	No	Yes/Machine Learning	Reduce energy consumption and latency
8	Traffic and Computation Co-Offloading with Reinforcement Learning in Fog Computing for Industrial Applications [33]	No	Yes	Yes/Reinforcement Learning, Deep Learning	Reduce energy consumption and service delay

Table 2 continued

Row	Article nam	Use SDN	Architecture presented	Use AI methods/algorithm	Objectives
9	Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing [34]	No	Yes	Yes/Reinforcement Learning (PDS)	Minimize the long- term system cost (including both service delay and operational cost)
10	Q-placement: Reinforcement-Learning-Based Service Placement in Software Defined Networks [35]	Yes	No	Yes/Reinforcement Learning (Q-Learning)	Minimizing the average accumulated service costs for end users
11	A Deep Reinforcement Learning based Offloading Scheme in Ad-hoc Mobile Clouds [36]	No	No	Yes/Deep Reinforcement Learning	Making an optimal offloading action decision at each system state such that the utility obtained by the task execution is maximized while minimizing the energy consumption, processing delay, required payment and task loss probability
12	An Intelligent Resource Management Method in SDN-based Fog Computing Using Reinforcement Learning (this article)	Yes	Yes	Yes/Reinforcement Learning (Q-learning, Expected SARSA, SARSA and Monte Carlo)	Optimum use of resources in the fog layer and flexibility in the management of this resources

To simplify the modeling, the current study assumed that FBS caching, computing, and processing capacity were quantitatively comprehensive indicators presented as resource blocks (RBs) limited to N [2]. Furthermore, all FBSs were presumed to have constant resources; thus, their RBs would not change over time.

Additionally, we supposed that the end node could not supply its own needs over time, which is why it was trying to access the network resources by sending requests to the local FBS in its domain. Each EN must calculate its latency requirements (maximum acceptable delay), computational cost, and amount of data that must be transferred from the EN before sending the request [2]. FBS can calculate some or all of the intended parameters based on the intended IoT environment. After receiving the request from the IoT environment, FBS calculated the request utility by the method presented later, then forwarded the request to the responsible controller.

The method of calculating the request utility was based on the maximum acceptable delay for requesting EN, the amount of data that must be transferred from EN, and the computational cost required by EN. Moreover, the responsible controller must decide how to respond to the received request based on the status of the resources in the fog layer and the request utility. Thus, the decision included one of the following:

1. Sending the request to one of its own FBSs,
2. Sending the request to the neighbor controller (to receive service on one of the FBSs under its control),
3. Sending the request to the cloud.

The SDN-based architecture (introduced in Fig. 1) allowed the controller to use the resources of other neighbors, in addition to the cloud resources and own resources, for responding to requests. In other words, the controller must make the optimized decision between these three options, which increases the necessity of optimized decision-making.

The SDN controller implements computing and network traffic-related policies using south-bound API (such as Open Flow [38]) and West-East API (such as BGP and OSPF[39]). As mentioned, the FN's computing and processing capacity were limited to N resource blocks (RBs). Furthermore, when user requests arrived sequentially and decisions were taken quickly, no queuing occurred [2].

IoT applications have different levels of latency requirements. Therefore, SDNC gives higher priorities to the servicing low-latency applications. Besides, computational costs and data transfer are taken into account in determining utility to differentiate between requests with similar latency requirements and considering negative points for requests with higher computing costs and data transfers.

Thus, we considered the utility of an IoT end-node request proportional to the reverse of acceptable delay D (in milliseconds), the reverse of process time P (in milliseconds), and the reverse of data transfer time T (in milliseconds). Consequently, we would have: $U \approx \frac{1}{(D \times P \times T)}$ Considering the equal data transfer times: $T = \frac{S}{A}$; where T is the transfer time, A the amount of data that must be transferred from EN (data size of a task), and S the speed or rate of transfer. Then, we would have: $U \approx \frac{S}{D \times P \times A}$. Notably, the items involved in determining the utility of any request (other than the latency requirements) are highly dependent on the environment. In addition, the priority of each item can be changed based on the environment. Therefore, more flexibility can be provided for calculating the utility by using coefficients or power for each item.

$$U = \delta\left(\left(\frac{1}{D}\right)^{\beta} \times \left(\frac{S}{P \times A}\right)^{\sigma}\right) \tag{1}$$

where δ, β and $\sigma > 0$ are mapping parameters.

We obtained the desired range of U and importance level for latency, computational cost, and data transfer time by selecting the δ, β and σ. Usually, latency can get higher levels of importance by selecting larger β values. Considering the choices that SDNCs have (three options), the responsible controllers must act intelligently to make the correct decision for each IoT request. Their intelligent decision helps to achieve the conflicting objectives of maximizing the average total utility of EN requests that responded at the fog layer over time and minimizing their resource idle time. Therefore, the system objective can be described as a limited optimization problem as follows:

$$\begin{aligned} &Max \sum_{t=0}^{T}(u_t \mid a_t = Local\ Service\ Or\ Service\,in\ Fog) \\ &\qquad\qquad\qquad\qquad\qquad\qquad And \\ &Min \sum_{t=0}^{T}(FRB_{Local} + FRB_{Fog} \mid a_t = Service\,in\,Cloud) \end{aligned} \tag{2}$$

where a_t is the action performed at time t and T is the end time when all RBs at the fog layer, including the local responsible SDNC resource blocks and neighbor SDNC resource blocks, are occupied. In addition, FRB_{Fog} is the number of unoccupied RBs related to FBSs under the management of neighbor SDNCs, and FRB_{Local} is the number of unoccupied RBs related to the FBSs under the management of the responsible SDNC.

3 Problem formulation

Reinforcement learning can be applied as a mathematical framework for autonomous learning through interacting with the environment. In the standard reinforcement learning model, an independent learner agent interacts with the environment through a sequence of observations, actions, and rewards. At each time step t, the agent initially observes a state, s_t, from its environment. Then, performs an action a_t and receives a number r_t as the reward feedback. After conducting the action a_t in the environment, the environment is transformed to a new state s_{t+1} (Fig. 2). The process continues, and the agent aims to learn the policy, which is an action selection strategy for maximizing the desired reward in the long run. Moreover reinforcement learning primarily focuses on learning without being aware of the environmental model [4]. Accordingly, this

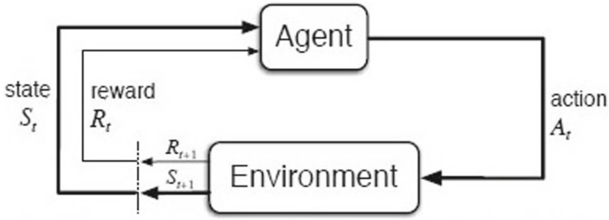

Fig. 2 Agent environment interaction in a MD [40]

is appropriate for use in our desired fog computing, given what was addressed in the previous section.

Formally, reinforcement learning can be described as a Markov decision-making process (MDP), an ideal mathematical form of the reinforcement learning problem. That is because a detailed theoretical description can be expressed in which the environment's response to the subsequent state S_{t+1} only depends on the state S_t where the action A_t is taken. Furthermore, MDP and agent create a sequence that starts in the following order: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$. In a finite MDP, the set of states, actions, and rewards (S, A, and R) all have a limited number of factors. In addition, the the resource allocation problem in the fog layer can be defined as a finite quadruple MDP as follows.

3.1 Set of states

S is a set of feasible states,for example, $s_t \in S$. In addition, the utility values are quantized to model the environment; therefore, we have $u_t \in 0, 1, 2, 3, \ldots, U$. Accordingly, the state S_t in any time t is defined as follows:

$$S_t = (10^{m+n} \times FRB_{Fog}) + (10^n \times FRB_{Local}) + U_{t+1} \tag{3}$$

which allows representing any state simply by a single number. Here,FRB_{Local} is the total number of free RBs associated with FBSs controlled by the responsible SDNC at time t. FRB_{Fog} is the total number of free RBs associated with FBSs controlled by neighboring SDNCs at time t. Besides, m and n are positive integers defined as follows: m is the smallest integer such that $U < 10^n$; and n is the smallest integer such that $RB_{Local} < 10^m$. Regarding this description, Eq. (2), and assuming that in case of reference to fog, the request is always referred to a neighboring SDN with the largest number of free resources;[6] the number of feasible states is:

$$Numberofstates = (NRB_{Local} + 1) \times (NRB_{Fog} + 1) \times U \tag{4}$$

[6] It is certainly possible to use other methods, such as choosing the controller that has received the least requests so far; but this will make the model more complex.

3.2 Set of actions

A set of possible actions was defined in our model as follows. For a user request with utility U_t at time t, the controller must select the optimal action from the following three actions (assuming not all resources are occupied in the fog layer):

- **Local Service:**[7] One of the SDNC FBSs will be assigned to respond to the request. As a result, one RB of resources managed by SDNC will be occupied, and the immediate reward rt is received. (a_t = Local Service).
- **Service in Fog Layer:** The request is forwarded to one of the neighbor controllers. For action a_t = Fog Layer Service, one RB of resources under the control of the neighbor SDNC is occupied , RBs of local FBS are retained, and reward, r_t is received.
- **Service in Cloud Layer:** The request is forwarded to the cloud (a_t = Cloud Layer Service), all existing resources blocks of FBS in fog layer are retained, and a reward r_t is received.

Based on the descriptions mentioned above, the set of actions can be defined as follows:

$$A = \{ServiceinFogLayer, ServiceinCloudLayer, LocalService\}$$

3.3 Probability function

$P^a_{SS'}$ is the transition probability from state S to $S^{'}$ by choosing the action a, which means $P^a_{SS'} = P(S^{'} \mid s, a)$, where $S^{'}$ represents the successor state. This function is unknown concerning the lack of information from the IoT environment and how requests are sent from it. In addition, and some methods are presented in the following sections to overcome this problem.

3.4 Set of rewards

In reinforcement learning, the purpose or goal is recognized as a specific signal, named reward, and transferred from the environment to the agent. In each time step, a reward is a real number and $r_t \in R$. $R^a_{SS'}$ is the immediate reward received when the action a is adopted at state S and ends at state $S^{'}$. The reward mechanism of $R^a_{SS'}$ is usually selected based on the goal of the system designer concerning the system's unknown nature.

In our model, rewards were determined in a way that led to our intended policies. Moreover in our environment, the goal was to give priority to requests with higher U_t at the fog layer. In addition, the goal was to use the local resources of the responsible SDNC. If the responsible SDNC was some unoccupied resources more than the average number of resources in the fog layer, it used its resources. To achieve the stated goal (Eq. 2), r_t was defined based on the received utility U_t and system state S_t as shown

[7] This name is negligently selected because service is provided at the fog layer while using the local resources of the responsible controller.

in Table 3. According to the following table:

$$r_t \in \{R_{LH1}, R_{LH2}, R_{LL1}, R_{LL2}, R_{LM1}, R_{LM2}, R_{FH1}, R_{FH2}, R_{FH1}, R_{FH2}, R_{FM1}, R_{FM2}, R_{CH1}, R_{CH2}, R_{CL1}, R_{CL2}, R_{CM1}, R_{CM2}\}$$

U_h and U_l of a request are among the design parameters and depend on the utility distribution in the IoT environment. For example, U_h and U_l can be a certain percentile of utilities in the environment.

Example: Consider a hypothetical environment with two controllers. Each one has a FBS with four resource blocks ($NRB_{local} = NRB_{Fog} = 4$). Nine levels of request utility are considered in the environment ($U = 9$), and upper and lower thresholds of utility are $U_h = 7$ and $U_t = 3$, respectively. The hypothetical scenario of IoT requests with u_t and random actions can be observed in Table 4. Furthermore, the state changing graph is expressed as follows:

$$4491 \rightarrow 3461 \rightarrow 3321 \rightarrow 3221 \rightarrow 3241 \rightarrow 2271 \rightarrow 2231 \rightarrow 1251 \rightarrow 0281 \rightarrow 0211 \rightarrow 0181 \rightarrow 061$$

3.5 Cumulative reward

According to the definition presented for the reward, the agent's goal can be considered the maximization of the total received reward. Thus, the optimal action in each state is defined as an action that maximizes the cumulative reward as follows [40]:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{5}$$

where T is the terminal state. The terminal state is reached in this problem when all local and fog layer resource blocks are occupied. To discriminate between immediate and future rewards, $\gamma \in [0, 1]$ is defined as the discount rate, which "determines the present value of future rewards"[40]. Moreover $\gamma = 0$[8] means no significance for the future rewards and $\gamma = 1$ means the similar significance of future and immediate rewards. Therefore, the MDP problems aim to maximize the cumulative discounted rewards from the start point (G_0), which can be expressed as follows [40]:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{6}$$

[8] If $\gamma = 0$, the agent is myopic in being concerned only with maximizing immediate rewards [40].

Table 3 The Rewarding method based on selected action, received request utility (U_t) and state condition (S_t) including local free blocks (FRB_{local}), free blocks in the fog layer (FRB_{fog}), the total number of existing resources in the environment (total RB), total number of neighbor controls (N_i) and thresholds U_h and U_l

	U_t	RBs status	r_t	
Local Service	$U_t \geq U_h$	$FRB_{Local} > FRB_{Fog}/N_i$	R_{LH1}	
	$U_t \geq U_h$	$FRB_{Local} \leq FRB_{Fog}/N_i$	R_{LH2}	
	$U_t \leq U_l$	$FRB_{Local} > FRB_{Fog}/N_i$	R_{LL1}	
	$U_t \leq U_l$	$FRB_{Local} \leq FRB_{Fog}/N_i$	R_{LL2}	R_{Local}
	$U_l < U_t < U_h$	$FRB_{Local} > FRB_{Fog}/N_i$	R_{LM1}	
	$U_l < U_t < U_h$	$FRB_{Local} \leq FRB_{Fog}/N_i$	R_{LM2}	
Service in Fog	$U_t \geq U_h$	$FRB_{Local} < FRB_{Fog}/N_i$	R_{FH1}	
	$U_t \geq U_h$	$FRB_{Local} \geq FRB_{Fog}/N_i$	R_{FH2}	
	$U_t \leq U_l$	$FRB_{Local} < RB_{Fog}/N_i$	R_{FL1}	
	$U_t \leq U_l$	$FRB_{Local} \geq FRB_{Fog}/N_i$	R_{FL2}	R_{Fog}
	$U_l < U_t < U_h$	$FRB_{Local} < FRB_{Fog}/N_i$	R_{FM1}	
	$U_l < U_t < U_h$	$FRB_{Local} \geq FRB_{Fog}/N_i$	R_{FM2}	
Service in Cloud	$U_t \geq U_h$	$FRB_{Local} + FRB_{Fog} < TotalRB/2$	R_{CH1}	
	$U_t \geq U_h$	$FRB_{Local} + FRB_{Fog} \geq TotalRB/2$	R_{CH2}	
	$U_t \leq U_l$	$FRB_{Local} + FRB_{Fog} < TotalRB/2$	R_{CL1}	
	$U_t \leq U_l$	$FRB_{Local} + FRB_{Fog} \geq TotalRB/2$	R_{CL2}	R_{Fog}
	$U_l < U_t < U_h$	$FRB_{Local} + FRB_{Fog} < TotalRB/2$	R_{CM1}	
	$U_l < U_t < U_h$	$FRB_{Local} + FRB_{Fog} \geq TotalRB/2$	R_{CM2}	

Table 4 Actions and states for the hypothetical environment

t	U_t	FRB_{Local}	FRB_{Fog}	S_t	A_t	R_t	S_{t+1}
0	9	4	4	449	Local	R_{LH2}	346
1	6	3	4	346	Fog	R_{FM1}	332
2	2	3	3	332	Fog	R_{FL2}	322
3	4	3	2	322	Cloud	R_{CM2}	324
4	7	3	2	324	Local	R_{LH2}	227
5	3	2	2	227	Cloud	R_{CM2}	223
6	5	2	2	223	Local	R_{LM2}	125
7	8	1	2	125	Local	R_{LH2}	028
8	1	0	2	028	Cloud	R_{CL1}	021
9	8	0	2	021	Fog	R_{FH1}	018
10	6	0	1	018	Fog	R_{FM1}	006
11	2	0	02	006	Cloud	R_{CL1}	–

The received reward at k time step in the future is γ^{k-1} times as much as the reward received immediately. For episodic tasks,[9] Eq. (5) can be defined as follows:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{7}$$

Returns are interrelated at consecutive time steps, which is more critical for RL theory and algorithms and can be expressed as follows:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{8}$$

3.6 Policies and value functions

Almost all RL algorithms involve estimating the state value function $V(s)$ and action value function $Q(s, a)$, which are estimations of how good the agent is in a certain state (how good a given action is in a certain state) [40]. The concept "how good" is specified regarding the expected future rewards and the expected return, based on Eq. (6) stated above.

The expected rewards the agent receives in the future depend on the actions it will do. To this end, the state value function is defined concerning specific practical methods, named policies. The policy of mapping states to probabilities of selecting an action is formally feasible.

9 Episodic tasks are the tasks that have a terminal state.

The state value function of state S following policy π $(v_\pi(s))$ is the expected return at the beginning of state S and following policy π after that. For MDPs, v_π can be formally defined as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], forall \quad s \in S \tag{9}$$

where E_π is the expected value of a random variable given that the agent follows policy π, and t is the time step. It should be noted that the expected value of the terminal state, if it exists, is always zero. The present study named function v_π as the state value function for policy π. Similarly, This study determined the value of choosing action in state S, following policy π, shown as $q_\pi(s, a)$, as the expected return which starts from S, performs action a, and then follows policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \tag{10}$$

This study named q_π as the action value function following policy π.

The fundamental feature of state value function used during reinforcement learning is that they satisfy recursive relationships similar to that created for return (Eq. 8). For each policy π and any state S, the following consistency condition holds between the value of s and the value of its possible successor states [40]:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \quad (By \quad 8 : G_t = R_{t+1} + \gamma G_{t+1}) \\ &= \sum_a \pi(a \mid s) \sum_{S}^{'} \sum_r p(S^{'}, r \mid s, a)[r + \gamma E_\pi[G_{t+1} \mid S_{t+1} = S^{'}]] \\ &= \sum_a \pi(a \mid s) \sum_{S^{'}, r} p(S^{'}, r \mid s, a)[r + \gamma v_\pi(S^{'})] \quad forall \quad s \in S \end{aligned} \tag{11}$$

where action a is taken from A, successor state $S^{'}$ from S, and rewards r from R. Eq. (11) is the Bellman equation for v_π [40]. It is a widely-used and crucial formula indicating a relationship between a state's value and its successor states' values. Moreover a reinforcement learning task means finding a policy that leads to the maximum reward in the long term, which is named optimal policy. For limited MDPs, an optimal policy can precisely be defined as follows:

$$v_*(s) = max v_\pi(s) \quad forall \quad s \in S \tag{12}$$

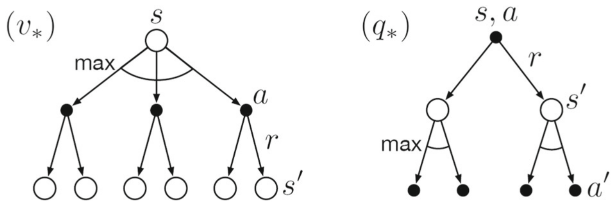

Fig. 3 Backup diagrams for v_* and q_* [40]

Optimal policies also have an optimal action value function, which is represented by q_* and defined as follows:

$$q_*(s) = maxq_\pi(s) \quad forall \quad s \in S \tag{13}$$

We can rewrite q_* in terms of v_*:

$$q_*(s) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid s_t = s, A_t = a] \tag{14}$$

Since v_* is the state value function of an optimal policy, we have:

$$\begin{aligned} v_*(s) &= max_{a \in A} q_{\pi *}(s, a) \\ &= max_a \mathbb{E}[R_{t+1} + \gamma v_*(s_{t+1}) \mid S_t = s, A_t = a] \quad (By14) \end{aligned} \tag{15}$$

Equation (15) is the Bellman optimality equation for v_*. In addition, the Bellman optimality equation for q_* is expressed as follows:

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma max_{a'} q_*(S_{t+1}, a^{'}) \mid S_t = s, A_t = a] \tag{16}$$

The Bellman optimality equation states that the value of a state under an optimal policy should be equal to the expected return for the best action in this state. In other words, the optimal policy results from choosing the best actions in every state. The expression $Q_*(s, a)$ is even more convenient for choosing the optimal actions because the best action in each state is selected based on the maximum value of this expression. Based on Eq. (14), the following equation can be applied to determine the optimal actions:

$$a_* = max_{a \in A} q_*(s, a) = max_{a \in A} \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \tag{17}$$

The following backup diagrams show the domain of future states and actions considered in Bellman optimality equations for v_* and q_* (Fig. 3).

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. However, this solution is rarely directly useful. It is akin to an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and desirabilities in terms of expected rewards. This solution relies on at least three assumptions that are rarely true in practice: (1) we accurately know the dynamics of the environment, (2) we have enough computational resources to complete the computation of the solution and (3) the Markov property [40].

A technique for solving Bellman equations and calculating optimal policies is dynamic programming (DP). DP algorithms update the state values based on estimating the successor state values; that is, other estimations update estimations. This idea is named bootstrapping. Many reinforcement learning techniques conduct bootstrapping.

The present study assumed the lack of precise information being from the IoT environment where the agent is located in this problem. Thus, considering the limited number of states, this study applied model-free RL techniques, known as approximate DP techniques, to solve the problem. This study resorted to model-free RL techniques instead of exact DP and estimated the optimal value functions. The related algorithms are presented in the following sections.

In this study's problem, the request was first sent from the IoT layer to FBS, and FBS passed it for decision-making about execution to the responsible SDNC. Then, SDNC decided to choose the optimal action among the allowable ones based on the current state and utility of the received request. Therefore, according to Eq. (17), the optimal action in the MDP problem is defined as follows:

$$\begin{aligned} a_* = max\{R_{Local} + \gamma \mathbb{E}_u[V^*(S^{'}_{Local}), R_{Fog}] \\ +\gamma \mathbb{E}_u[V^*(S^{'}_{Fog})], R_{Cloud} + \gamma \mathbb{E}_u[V^*(S^{'}_{Cloud}])\} \end{aligned} \tag{18}$$

According to Eq. (3), successor states are defined as follows:

$$\begin{aligned} S^{'}_{Local} &= (10^{m+n} \times FRB_{Fog}) + (10^{n} \times FRB_{Local} - 1) + u_{t+1} \\ S^{'}_{Fog} &= (10^{m+n} \times FRB_{Fog} - 1) + (10^{n} \times FRB_{Local}) + u_{t+1} \\ S^{'}_{Cloud} &= (10^{m+n} \times FRB_{Fog}) + (10^{n} \times FRB_{Local}) + u_{t+1} \end{aligned}$$

where $S^{'}_{Local}$ is the successor state when a= Local Service, $S^{'}_{Fog}$ is the successor state when a=Service in Fog Layer, and $S^{'}_{Cloud}$ is the successor when that a=Service in Cloud Layer and $\mathbb{E}_u$ is the expectation concerning the utilities u in the IoT environment.

3.7 Monte Carlo method

This section presents the first learning method for estimating value functions and discovering optimal policies. This method helps to solve the resource allocation problem of fog computation. As mentioned, the lack of thorough environmental cognition is one of the assumptions here.

A popular method to compute the optimal state values is the value iteration by Monte Carlo (MC) calculations, as presented in Eq. (13). MC methods only require the experiences, which is the sequence of samples of states, actions, and received reward resulting from the actual or simulated interaction with an environment. These methods are used for reinforcement learning based on averaging sample returns.

The value of a state is the expected return, which is the cumulative discounted future rewards beginning from that state. Therefore, a straightforward method for estimating the state value is to use experiences. The experiences are used by calculating the mean value of averaging returns after each state visit and observing many returns. It is worth noting that the average must converge to the expected value. This idea is the basis of all MC methods.

For example, assuming that we want to estimate the value of $V_\pi(s)$, based on the MC method, we would have: $V_\pi(S) = \mathbb{E}_\pi[G_t \mid S_t = s]$. Figure 4A indicates the backup diagram for the MC method. Moreover, Algorithm1 presents how to learn the optimal policy for the MDP problem based on the MC method. Each occurrence of state S in an episode is named the visit of S. In an episode, S may be visited many times. The first-visit MC method estimates $v(s)$ as the average of the returns after the first visit of state S. However, the every-visit MC estimates the average of returns after all visits of state S. These two MC methods are so similar to each other, with only some differences in the theoretical properties. We considered the first-visit MC method in this study.

Algorithm 1 Learning optimum policy using Monte Carlo

Require: $\{\epsilon, \gamma\} \in [0; 1]$, $\{U_l, U_h\} \in \mathbb{R}$, $\{NRB_{Local}, NRB_{Fog}, U\} \in \mathbb{N}$, $\{R_{LH1}, R_{LH2}, R_{LL1}, \ldots, R_{Cl2}, R_{CM1}, R_{CM2}\} \in \mathbb{R}$
$V(s) \leftarrow 0$ (for all s ∈ S) ;
$Returns(s, iteration) \leftarrow$ an empty list (a Two-dimensional array to save states' returns in all iterations);
$Temporary(s) \leftarrow$ an empty list ;
for iteration = 0; 1; 2; ... **do**
$n_{Local} \leftarrow 0$;
$n_{Fog} \leftarrow 0$;
Generate an episode (Generate random sequences of U_s in the range of 1 to U as received requests);
  while all the Fog resource blocks (NRB_{Local} and NRB_{Fog} are occupied **do**
    Choosing the optimal action a_t based on Formula (24) and π (e.g., ϵ-greedy);
  end while
Temporary(s) ← sum of discounted rewards from s till terminal state for all states appearing in the episode;
Append Temporary(s) to Returns(s, iteration);
$V(s) \leftarrow average(Returns(s, iteration))$;
  if V (s) converges for all s **then**
$V^*(S) \leftarrow V(S), \forall s$;
break;
  end if
end for

3.8 Temporal difference learning methods

Temporal difference (TD) learning combines MC and DP ideas. TD methods combine the sampling of MC with bootstrapping of DP. While Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S_t)$ (only then is Gt known) [40], TD methods need to wait only until the next time step. In addition, TD methods immediately perform an update at time t+1 using observed reward Rt+1 and estimating $V(S_{t+1})$. The simplest TD method leads to the following update:

$$V(S_t) \leftarrow V(S_t) + \alpha[\, R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{19}$$

The aim of updating in the MC method is Gt, while in the TD method, it is $R_{t+1} + \gamma V(S_{t+1})$. This is named TD(0) or the one-step TD methods. Since TD(0), like DP, builds its updating based on existing estimates, it is called a bootstrapping method. Figure 4B shows the backup diagram for the TD(0) method. Furthermore, TD methods have an advantage over DP methods as they do not require knowledge of the next case's environment model, reward, and probability distribution. Moreover, the most sensible advantage of TD methods over MC methods is that they are implemented online, fully incremental, and only wait for one time-step. Similar to the MC methods, these approaches are divided into on-policy and off-policy. The current study presented the optimal policy learning method from an IoT environment using the on-policy SARSA and off-policy Q-learning methods.

3.8.1 On-policy temporal difference learning method

In the TD(0) method, the present study considered the transfer from one state to another and learned about state values. Here, this study considered the transfer from a state-action pair to another state-action pair and learned about the values of the state-action pair. To this end, Eq. (19) can be rewritten based on the state-action pair:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\, R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{20}$$

This update was carried out after each transfer from a non-terminal state. If S_{t+1} is the terminal state, $Q(S_t, A_t)$ would be defined as zero.

This formula uses all five elements of events $(S_t, A_t, R_{t+1}, S_{t+1}, S_{t+1})$ that form the transfer from a state-action pair to another. These five elements are the reason for naming this algorithm SARSA. The backup diagram for SARSA is presented in Fig. 4C.

3.8.2 Q-learning (off-policy temporal difference learning method)

One of the successful reinforcements of the learning methods is the off-policy TD control algorithm, also known as Q-learning, which is defined as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\, R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{21}$$

In this case, the action value function Q estimates the optimal action value function directly and independently of the following policy. This estimation significantly simplifies the algorithm's analysis and makes it possible to prove the initial convergence. The backup diagram can be seen in Fig. 4D.

In addition, the integrated application of SARSA and Q-learning methods for solving the optimal resource allocation problem are indicated in Algorithm 2.

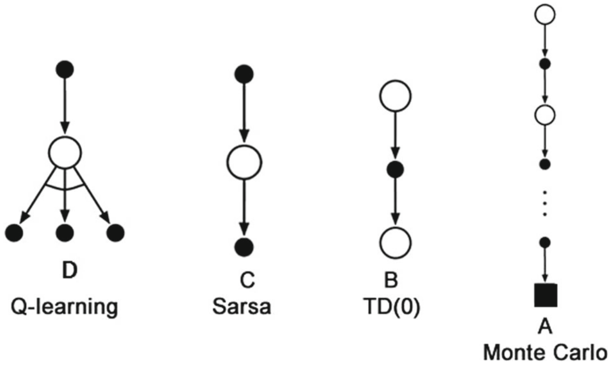

Fig. 4 Backup diagrams for TD, SARSA, Q-learning, and Monte Carlo methods

Algorithm 2 Learning optimum policy using QL and SARSA

Require: $\{\epsilon, \gamma\} \in [0; 1], \alpha \in (0; 1]$ $\{U_l, U_h\} \in \mathbb{R}$, $\{NRB_{Local}, NRB_{Fog}, U, n\} \in \mathbb{N}$, $\{R_{LH1}, R_{LH2}, R_{LL1}, \ldots, R_{Cl2}, R_{CM1}, R_{CM2}\} \in \mathbb{R}$
$Q(s, a) \leftarrow 0$ for all (s,a) ;
for iteration = 0; 1; 2; . . . **do**
$n_{Local} \leftarrow 0$;
$n_{Fog} \leftarrow 0$;
Generate an episode (Generate random sequences of U_s in the range of 1 to U as received requests);
while all the Fog resource blocks (NRB_{Local} and NRB_{Fog} are occupied **do**
$t \leftarrow 0$;
Choosing the optimal action a_t based on Formula (24) and π (e.g., ϵ-greedy) and store r_t and S_{t+1};.
if $t \geq n - 1$ **then**
$\tau \leftarrow t + 1 - n$;
QL:$G \leftarrow \sum_{j=\tau}^{t+1} \gamma^{(j-\tau)} r_j + \gamma^n max Q(s_{t+1}, a)$;
SARSA:$G \leftarrow \sum_{j=\tau}^{t+1} \gamma^{(j-\tau)} r_j + \gamma^n Q(s_{t+1}, a_{t+1})$;
$Q(s_\tau, a_\tau) \leftarrow Q(s_\tau, a_\tau) + \alpha[G - Q(s_\tau, a_\tau)]$;
end if
$t = t + 1$;
end while
if Q (s,a) converges for all (s,a) **then**
$Q^*(s, a) \leftarrow Q(s, a), \forall(s, a)$;
break;
end if
end for

4 Evaluation and simulation

The goal of providing a solution is to maximize the total utility of the service requests and minimize the unoccupied time of local resources and fog layer resources (Eq. 2). Therefore, a performance metric named R was defined to evaluate the performance of the proposed algorithms and compare them with the performance of a constant threshold algorithm.

The total utility of requests serviced in fog is the essential factor in increasing the metric. However, it is defined in a way that the larger utility of served requests in the cloud and the greater number of unoccupied resource blocks available for the responsible controller when servicing the request in the cloud leads to the reduction in the R-value.

$$R = E\left[\sum_{t=0}^{T}(u_t \mid a_t = LocalOrFogService) \right. \\ \left. -\theta\sum_{t=0}^{T}(u_t + (NRB_{Local} + NRB_{Fog}/AllResource) \mid a_t = Cloud)\right] \tag{22}$$

where u_t is the utility of requests and NRB_{Fog} is the number of unoccupied RBs related to FBSs under the management of neighboring SDNCs. Moreover, NRB_{Local} is the number of unoccupied RBs related to FBSs under the management of responsible SDNC, T is the time period of the episode, and θ is the discount factor as a penalty for idle time ($\theta \in [0, 1]$).

In the next step, the simulation results were presented to evaluate the performance of the SDN controller during the execution of RL methods. The methods were SARSA, Q-learning, and MC (as presented in Algorithms 1 and 2) and in the introduced architecture frame. Then, the performance of the RL-based SDN controller was compared with that of the SDN controller with a network slicing approach with various slicing thresholds (2–8 slices) in different IoT environments with different compositions of IoT latency requirements.

In this simulation, consider 10 utility levels with different latency requirements to exemplify a variety of IoT applications. It is assumed that utilities are calculated according to Eq. (1). In the first level of utility (u = 1), the requests were placed with the least importance, similarly, in the last level (u = 10), the requests ware with the most importance. In this way, by changing the combination of utility classes, we produce 10 scenarios of IoT environments, which are presented in Table 5. As it is clear in Table 5, in the first environment, most of the requests are with high utility and have a special sensitivity (this environment is similar to a hospital or military IoT environment) and respectively, in the following environments, the number of requests with high utility decreases and the E10 environment has the lowest number of requests with high utility. (This environment is similar to an IoT entertainment or smart home environment)

The simulation parameters demonstrated in Table 6 were used in this section. Furthermore, the rewards defined in Table 3 were valued by Table 7. The greedy policy was selected as the optimal policy to facilitate and accelerate the learning process.

A local SDN controller equipped with computational, signal processing, and storage resources, including 5 RBs, was employed, meaning that: NRB_{Local}=5. Besides, our intended environment included three neighboring SDN controllers with a total of nine resources: NRB_{Fog}=9. The locally serving threshold (U_h) and serving threshold in the cloud layer (U_l) were expressed by the utility levels and the ration of local and fog layer resources to the total resources as:

Table 5 Utility distribution for different IoT environments with various latency requirements

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
P(u = 1)	0/008	0/035	0/065	0/063	0/1	0/123	0/109	0/133	0/114	0/216
P(u = 2)	0/009	0/067	0/048	0/093	0/1	0/127	0/146	0/139	0/185	0/209
P(u = 3)	0/011	0/072	0/107	0/125	0/1	0/063	0/121	0/167	0/094	0/185
P(u = 4)	0/016	0/029	0/073	0/116	0/1	0/092	0/126	0/174	0/208	0/175
P(u = 5)	0/006	0/047	0/107	0/103	0/1	0/095	0/098	0/137	0/199	0/165
P(u = 6)	0/265	0/097	0/098	0/087	0/1	0/107	0/107	0/047	0/031	0/006
P(u = 7)	0/175	0/073	0/126	0/107	0/1	0/079	0/073	0/029	0/035	0/016
P(u = 8)	0/075	0/129	0/121	0/086	0/1	0/115	0/107	0/072	0/054	0/011
P(u = 9)	0/219	0/277	0/146	0/096	0/1	0/102	0/048	0/067	0/043	0/009
P(u = 10)	0/216	0/174	0/109	0/124	0/1	0/097	0/065	0/035	0/037	0/008
P(u > 5)	%95	%75	%60	%50	%50	%50	%40	%25	%20	%5

Table 6 A summary of simulation parameters and their value

Parameter	Description	Value
γ	Discount factor	0.7
α	Learning rate	0.01
ϵ	Probability of random action	0
θ	Penalty of idle time	0.5
n	Batch/step size	1
NRB_{Local}	The total number of RBs associated with FBSs controlled by the responsible SDNC	5
NRB_{Fog}	The total number of RBs associated with FBSs controlled by neighboring SDNCs	9

Table 7 Rewards considered in the simulations

R_{Local}	R_{Fog}	R_{Cloud}
$R_{LH1} = 9$	$R_{FH1} = 8$	$R_{CH1} = 4$
$R_{LH2} = 7$	$R_{FH2} = 6$	$R_{CH2} = -4$
$R_{LL1} = -10$	$R_{FL1} = -2$	$R_{CL1} = 8$
$R_{LL2} = -8$	$R_{FL2} = -4$	$R_{CL2} = 6$
$R_{LM1} = 6$	$R_{FM1} = 10$	$R_{CM1} = 6$
$R_{LM2} = 2$	$R_{FM2} = 8$	$R_{CM2} = -2$

$$
\begin{aligned}
U_l &= \frac{U_{max}}{3} + (\mathbb{E}(u) - \frac{U_{max} + U_{min}}{2}) \\
U_h &= U_l + \left(U_{max} \times \frac{2}{3} \times \frac{NRB_{Fog}}{NRB_{Local} + NRB_{Fog}} \right)
\end{aligned}
\tag{23}
$$

To perform simulation, the software was designed and implemented, the simulation was repeated 300 times for each test environment and the results of each stage of the tests were stored in the database. After that, the analysis was done based on the data

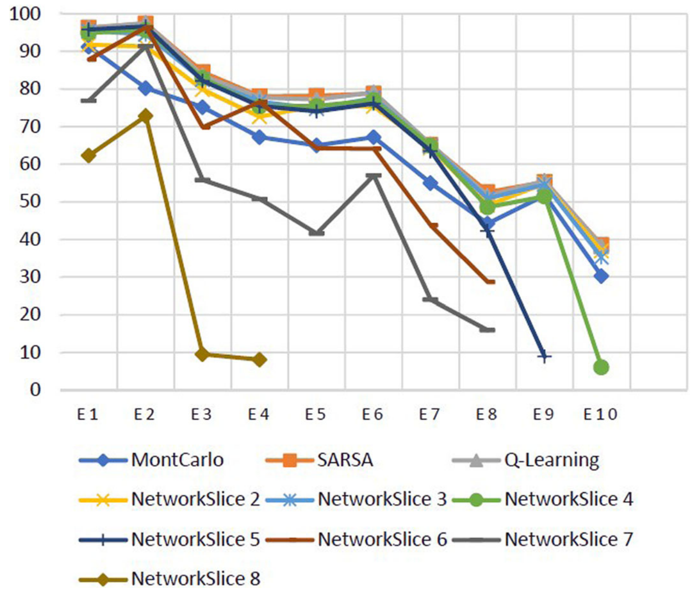

Fig. 5 Performance of RL method in terms of R metric

obtained from the tests. The following table compares the performance of RL methods in terms of R with that of utility filtering-based network slicing methods with various slicing thresholds in 10 different IoT environments. The utility filtering algorithm uses a constant threshold for network slicing, regardless of the environment. For the network slicing-based methods, 2, . . ., 8 possible slices are investigated and ten levels of utility (0–9) were considered in our discretization, as demonstrated in Fig. 5 and Table 8. Furthermore, SARSA and Q-learning methods outperformed other methods. In addition, SARSA and Q-learning methods showed the best performance. The results were based on the average 300 simulations for each environment and each method.

Based on Table 8, the performance of utility filtering algorithms was close to that of SARSA and Q-learning in some environments. As the table shows, in the E2 environment, the NetworkSlice5 and NetworkSlice6 methods had similar performance with Q-learning. The important point here is that determining what threshold should be used in each environment to achieve maximum performance is not simple, especially in complex environments where we don't have precise prior knowledge, and the state of the environment changes over time. Moreover, RL-based methods provide the best performance, compared to utility filtering algorithms, in different environments merely through learning while not requiring background knowledge about the environment.

5 Conclusions

This paper presented a new resource allocation framework using a software-defined network (SDN) architecture and reinforcement learning techniques. The aim was to

Table 8 Performance of RL method in terms of R metric

$\frac{Environment}{Method}$	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
Monte Carlo	91.23	80.21	75.21	67.18	65.01	67.16	54.98	44.15	51.73	30.24
SARSA	96.39	97.38	84.52	78.07	78.18	78.83	65.16	52.54	55.30	38.52
Q-Learning	96.23	97.53	83.52	77.72	77.21	79.12	65.27	51.61	55.58	38.19
Network Slice 2	91.78	91.30	79.84	72.59	75.64	75.34	64.33	49.10	54.83	36.86
Network Slice 3	95.48	94.56	82.51	76.63	74.81	77.56	64.66	50.93	54.61	35.23
Network Slice 4	94.93	95.69	82.90	75.42	75.49	77.05	64.89	48.54	51.45	6.05
Network Slice 5	95.83	96.63	82.20	75.45	74.02	76.14	63.50	42.24	8.90	– 322.12
Network Slice 6	87.86	96.40	69.81	76.52	64.26	64.08	43.76	28.68	– 15.48	– 374.3
Network Slice 7	76.88	91.38	55.77	50.76	41.55	57.04	23.99	15.91	– 56.23	– 653.17
Network Slice 8	62.30	72.81	9.50	8.07	– 3.72	– 10.08	– 89.81	– 78.58	– 153.82	– 1173.82

apply limited resources in the fog layer optimally. In addition, the framework was presented regarding an accurate architecture based on SDN with multiple controllers using reinforcement learning algorithms. Thus, the following achievements were reached:

1. The optimum use of resources in the fog layer and flexibility in managing these resources was employed. It was provided through exploiting the resources of other SDN controllers and using the resources of the SDN subset of the responsible controller. This is considered an innovation and a key advantage in efficiently managing resources.
2. The use of reinforcement learning methods made the responsible SDN controller behave like an intelligent agent without needing background knowledge of the environment. It was only through learning in different environments and choosing the optimal method of serving the received request from the possible options.

This framework was simulated based on three reinforcement learning techniques: SARSA, Q-learning, and Monte Carlo. The framework was examined in different IoT environments with heterogeneous latency demands. The simulation results showed the superiority of Q-learning and SARSA reinforcement learning compared to network slicing approaches with various slicing thresholds. In addition, the study found that Q-learning and SARSA reinforcement learning have consistency with the IoT environment. Furthermore, RL techniques made an appropriate compromise between two opposing objectives: maximizing the average utility of served requests and minimizing the idle time of fog resources. It is worth noting that the maximization indicated the optimal utilization of resources in the fog layer.

For future works, it is recommended to generalize the proposed resource allocation framework to susceptible environments with high request density, leading to queue formation for SDN controllers. In this situation, the subject of priority queues becomes meaningful, and making decisions for SDN controllers is accompanied by more complexities. Moreover, the shared resource allocation for a request, where the required resources are provided partly from resources under the control of RSDNC and the remaining from other SDNs, can be examined as a more complicated approach.

Declarations

Conflict of interest The authors did not receive support from any organization for the submitted work. All authors certify that they have no affiliations with or involvement in any organization or entity with any financial or non-financial interest.

References

1. Mukherjee M, Shu L, Wang D (2018) Survey of fog computing: fundamental network applications and research challenges. IEEE Commun Surv Tutor 20(3):1826–1857. https://doi.org/10.1109/COMST.2018.2814571
2. Baek J, Kaddoum G, Garg S, Kaur K, Gravel V (2019) Managing fog networks using reinforcement learning based load balancing algorithm. In: IEEE wireless communications and networking conference (WCNC), 2019, pp 1–7. https://doi.org/10.1109/WCNC.2019.8885745

3. Ali AMM, Ahmad NM, Amin AHM (2014) Cloudlet-based cyber foraging framework for distributed video surveillance provisioning. In: Proceedings of IEEE 4th world congress on information and communication technologies (WICT), pp 199–204
4. Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the Internet of Things. In: Proceedings of the first edition of the MCC workshop on mobile cloud computing (MCC), pp 13–16
5. Aazam M, Huh E (2016) Fog computing: the cloud-IoT/IoE middleware paradigm. IEEE Potentials 35(3):40–44
6. Vaquero LM, Rodero-Merino L (2014) Finding your way in the fog: towards a comprehensive definition of fog computing. SIGCOMM Comput Commun Rev 44(5):27–32
7. Lin C, Han G, Qi X, Guizani M, Shu L (2020) A distributed mobile fog computing scheme for mobile delay-sensitive applications in SDN-enabled vehicular networks. IEEE Trans Veh Technol 69(5):5481–5493
8. Baktir AC, Ozgovde A, Ersoy C (2017) How can edge computing benefit from software-defined networking: a survey, use cases, and future directions. IEEE Commun Surv Tutor 19(4):2359–2391
9. Mouradian C et al (2017) A comprehensive survey on fog computing: state-of-the-art and research challenges. IEEE Commun Surv Tutor 20(1):416–464
10. Baktir AC, Ozgovde A, Ersoy C (2017) How can edge computing benefit from software-defined networking: a survey, use cases, and future directions. IEEE Commun Surv Tutor 19(4):2359–2391
11. Gupta H, Nath SB, Chakraborty S, Ghosh SK (2017) SDFog: a software defined computing architecture for QoS aware service orchestration over edge devices. ArXiv Preprint, [online]. arXiv:1609.01190
12. Sun X, Ansari N (2016) EdgeIoT: mobile edge computing for the Internet of Things. IEEE Commun Mag 54(12):22–29
13. Truong NB, Lee GM, Ghamri-Doudane Y (2015) Software defined networking-based vehicular adhoc network with fog computing. In: Proceedings of the IFIP/IEEE international symposium on integrated network management (IM), pp 1202–1207
14. Yang P, Zhang N, Bi Y, Yu L, Shen XS (2017) Catalyzing cloud-fog interoperation in 5G wireless networks: an SDN approach. IEEE Netw 31(5):14–20
15. Lin C, Han G, Qi X, Guizani M, Shu L (2020) A distributed mobile fog computing scheme for mobile delay-sensitive applications in SDN-enabled vehicular networks. IEEE Trans Veh Technol 69(5):5481–5493
16. Liang K, Zhao L, Chu X, Chen H (2017) An integrated architecture for software defined and virtualized radio access networks with fog computing. IEEE Netw 31(1):80–87
17. Sahni Y, Cao J, Zhang S, Yang L (2017) Edge mesh: a new paradigm to enable distributed intelligence in Internet of Things. IEEE Access 5:16441–16458
18. Pang A-C, Chung W-H, Chiu T-C, Zhang J (2017) Latency-driven cooperative task computing in multi-user fog-radio access networks. In: Proceedings of the IEEE 37th international conference on distributed computing systems (ICDCS), June, pp 615–624
19. Chiu T-C, Chung W-H, Pang A-C, Yu Y-J, Yen P-H (2016) Ultra-low latency service provision in 5G fog-radio access networks. In: Proceedings of the IEEE 27th annual international symposium on personal, indoor and mobile radio communications (PIMRC), Sept, pp 1–6
20. Mukherjee M et al (2019) Task data offloading and resource allocation in fog computing with multi-task delay guarantee. IEEE Access 7:152911–152918
21. Zhou Z, Liu P, Feng J, Zhang Y, Mumtaz S, Rodriguez J (2019) Computation resource allocation and task assignment optimization in vehicular fog computing: a contract-matching approach. IEEE Trans Veh Technol 68(4):3113–3125
22. Zhang H, Xiao Y, Bu S, Niyato D, Yu FR, Han Z (2017) Computing resource allocation in three-tier IoT fog networks: a joint optimization approach combining Stackelberg game and matching. IEEE IoT J 4(5):1204–1215
23. Gu Y, Chang Z, Pan M, Song L, Han Z (2018) Joint radio and computational resource allocation in IoT fog computing. IEEE Trans Veh Technol 67(8):7475–7484
24. Li H, Ota K, Dong M (2018) Learning IoT in edge: deep learning for the Internet of Things with edge computing. IEEE Netw 32(1):96–101
25. Patel P, Ali MI, Sheth A (2017) On using the intelligent edge for IoT analytics. IEEE Intell Syst 32(5):64–69
26. La QD, Ngo MV, Dinh TQ, Quek TQS, Shin H (2019) Enabling intelligence in fog computing to achieve energy and latency reduction. Digit Commun Netw 5(1):3–9

27. Dutreilh X, Kirgizov S, Melekhova O, Malenfant J, Rivierre N, Truck I (2011) using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In: International conference on autonomic and autonomous systems (ICAS), May, Venice, Italy, pp 67–74
28. Le DV, Tham CK (2018) A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds. In: Proceedings under IEEE conference on computer communications workshops (INFOCOM WKSHPS), 15–19, pp 760–765
29. Li J, Gao H, Lv T, Lu Y (2018) Deep reinforcement learning based computation offloading and resource allocation for MEC. In: IEEE wireless communications and networking conference (WCNC). Barcelona 2018, pp 1–6
30. Parent J, Verbeeck K, Lemeire J (2002) Adaptive load balancing of parallel applications with reinforcement learning on heterogeneous networks. In: International symposium on distributed computing and applications to business & engineering science, pp 16–20
31. Gazori P, Rahbari D, Nickray M (2020) Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach, Future Gener. Comput Syst 110:1098–1115
32. Liu Y, Cheng S, Hsueh Y (2017) eNB selection for machine type communications using reinforcement learning based Markov decision process. IEEE Trans Veh Technol 66(12):11330–11338
33. Wang Y, Wang K, Huang H, Miyazaki T, Guo S (2019) Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. IEEE Trans Ind Inform 15(2):976–986. https://doi.org/10.1109/TII.2018.2883991
34. Xu J, Chen L, Ren S (2017) Online learning for offloading and autoscaling in energy harvesting mobile edge computing. IEEE Trans Cogn Commun Netw 3(3):361–373. https://doi.org/10.1109/TCCN.2017.2725277
35. Zhang Z, Ma L, Leung KK, Tassiulas L, Tucker J (2018) Q-placement: reinforcement-learning-based service placement in software-defined networks. In: 2018 IEEE 38th international conference on distributed computing systems (ICDCS), pp 1527–1532. https://doi.org/10.1109/ICDCS.2018.00159
36. Van Le D, Tham C (2018) A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds. In: IEEE INFOCOM 2018—IEEE conference on computer communications workshops (INFOCOM WKSHPS), pp 760–765. https://doi.org/10.1109/INFCOMW.2018.8406881
37. Wu D et al (2020) Towards distributed SDN: mobility management and flow scheduling in software defined urban IoT. IEEE Trans Parallel Distrib Syst 31(6):1400–1418
38. Hu T, Guo Z, Yi P, Baker T, Lan J (2018) Multi-controller based software-defined networking: a survey. IEEE Access 6:15980–15996
39. Rego A, Sendra S, Jimenez JM, Lloret J (2017) OSPF routing protocol performance in software defined networks. In: Fourth international conference on software defined systems (SDS), 2017, pp 131–136. https://doi.org/10.1109/SDS.2017.7939153
40. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction, 2nd edn. The MIT Press, Cambridge, pp 1–157
41. Nassar A, Yilmaz Y (2019) Reinforcement learning for adaptive resource allocation in fog RAN for IoT with heterogeneous latency requirements. IEEE Access 7:128014–128025