Computer Communications 222 (2024) 188–197

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Reinforcement learning-based dynamic load balancing in edge computing networks

Mohammad Esmaeil Esmaeili [a], Ahmad Khonsari [a,b,*], Vahid Sohrabi [a], Aresh Dadlani [c,d]

[a] School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran
[b] School of Computer, Institute for Research in Fundamental Science (IPM), Tehran, Iran
[c] Department of Computer and Electrical Engineering, Nazarbayev University, Astana, Kazakhstan
[d] Department of Computing Science, University of Alberta, Edmonton, Canada

ARTICLE INFO

Keywords:
Edge computing
Load balancing
Reinforcement learning
Q-learning
Multi-armed bandit
Gradient bandit algorithm

ABSTRACT

Edge computing (EC) has emerged as a paradigm aimed at reducing data transmission latency by bringing computing resources closer to users. However, the limited scale and constrained processing power of EC pose challenges in matching the resource availability of larger cloud networks. Load balancing (LB) algorithms play a crucial role in distributing workload among edge servers and minimizing user latency. This paper presents a novel set of distributed LB algorithms that leverage machine learning techniques to overcome the three limitations of our previous LB algorithm, EVBLB: (i) its reliance on static time intervals for execution, (ii) the need for comprehensive information about all server resources and queued requests for neighbor selection, and (iii) the use of a central coordinator to dispatch incoming user requests over edge servers. To offer increased control, custom configuration, and scalability for LB on edge servers, we propose three efficient algorithms: Q-learning (QL), multi-armed bandit (MAB), and gradient bandit (GB) algorithms. The QL algorithm predicts the subsequent execution time of the EVBLB algorithm by incorporating rewards obtained from previous executions, thereby improving performance across various metrics. The MAB and GB algorithms prioritize near-optimal neighbor node servers while considering dynamic changes in request rate, request size, and edge server resources. Through simulations, we evaluate and compare the algorithms in terms of network throughput, average user response time, and a novel LB metric for workload distribution across edge servers.

1. Introduction

The rapid growth of the Internet of Things (IoT), where billions of physical devices and objects equipped with sensors connect to the Internet to access required resources, has led to the increasing popularity of cloud computing [1]. In traditional cloud architecture, numerous data centers are dispersed worldwide and are geographically located at considerable distances from each other. While these data centers provide abundant processing and storage resources that are crucial for IoT devices, their architecture is unsuitable for time-sensitive applications like environmental sensing [2], healthcare [3,4], and smart cities [5] due to high latency. To mitigate this problem, edge computing (EC) has emerged as a new paradigm in recent years, garnering attention from both industry and academic researchers [6–8].

In this emerging architecture, resources are positioned in close proximity to users' equipment in order to minimize delay and response time. As illustrated in Fig. 1, the general architecture of the EC network consists of three main layers: (i) the *cloud layer*, which includes remote cloud servers to handle resource-intensive requests, (ii) the *edge layer*, which forms a complex network of edge servers to deliver services closer to users and decrease response time, and (iii) the *device layer*, consisting of all devices that are served through base stations (BS). Though edge servers have limited processing and storage capacity compared to cloud servers, their distributed nature allows for improved response time of incoming requests. Therefore, cloud networks serve as the ultimate layer to handle resource-intensive requests in most architectures proposed for edge networks [7–9].

In response to the challenges posed by limited resource availability in edge networks, load balancing (LB) has been investigated as an effective solution for distributing incoming traffic among multiple servers. These algorithms ensure that no single server is overwhelmed with requests, thereby minimizing the overall response time. Moreover, the implementation of LB algorithms enables network operators to optimize server resource utilization and mitigate the risk of server downtime

* Corresponding author at: School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran.
E-mail addresses: me.esmaeili@ut.ac.ir (M.E. Esmaeili), a_khonsari@ut.ac.ir (A. Khonsari), vahidsohrabi.mkh@ut.ac.ir (V. Sohrabi), aresh.dadlani@nu.edu.kz (A. Dadlani).

https://doi.org/10.1016/j.comcom.2024.04.009
Received 13 September 2023; Received in revised form 12 January 2024; Accepted 3 April 2024
Available online 17 April 2024

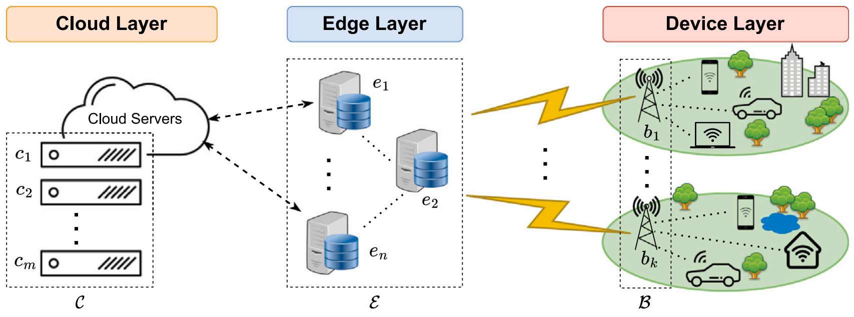


Fig. 1. General edge computing network architecture comprising m cloud servers, n edge servers, and k base stations.

due to excessive loads. It is important to note that there are various LB algorithms available, each offering a trade-off between performance criteria such as server load, response time, latency, and more [10].

In recent years, several LB algorithms for EC networks have emerged, each with its own strengths and shortcomings. These algorithms employ techniques such as optimization, game theory, and machine learning (ML) to address the LB problem with different performance needs [10–12]. In [13], we proposed the efficient Voronoi tessellation-based load balancing (EVBLB) approach in EC networks, which utilizes Voronoi tessellation (VT) [14] to efficiently assign user requests to the edge server with the least load. Simultaneously, it selects edge servers that are closer to the user to ensure low propagation delay. The use of VT allows for quick mapping of users to edge servers based on their geographical proximity and incorporates information about server density in the area, which is needed for load distribution. The EVBLB approach then applies an iterative procedure on the mapping to further distribute the user requests among neighboring edge servers. Nonetheless, the algorithm faces three major limitations that make it impractical for large-scale deployment. Firstly, it employs a static time interval that lacks flexibility and cannot adapt to the network dynamics such as request rates and network bandwidth. This inflexibility reduces the efficiency of the LB algorithm. Secondly, selecting neighboring edge servers in EVBLB requires global information on all server resources and queued requests, which is practically infeasible. Thirdly, the use of a central coordinator in EVBLB to dispatch incoming user requests over edge servers can serve as a bottleneck for its scalability. Building upon these observations, this paper presents a comprehensive extension of EVBLB with the following contributions:

- A Q-learning (QL) [15,16] algorithm is proposed for dynamic time interval selection in EVBLB to enhance its efficiency and adaptability to network changes.
- A novel multi-armed bandit (MAB)-based algorithm [17] is incorporated into EVBLB to enable dynamic edge server selection for efficient request allocation.
- By incorporating gradient information into the dynamic edge server selection process, a gradient bandit (GB) algorithm [16,17] is also proposed, overcoming the need for periodic monitoring of neighboring servers while achieving performance comparable to EVBLB.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the system model and problem formulation, followed by the three ML-based LB algorithms proposed in Section 4. An analysis of the computational complexity of the algorithms is provided in Section 5. Simulation results are discussed in Section 6, and conclusions are drawn in Section 7.

2. Related work

The challenge of LB in edge server networks has recently attracted considerable research attention. While several LB algorithms have been reported, our focus in this paper centers on optimization-based and ML-based algorithms.

In [13], a novel heuristic based on VT was introduced to efficiently solve the LB optimization problem in polynomial time. The algorithm assumes edge servers with limited task queues and employs a first-in-first-out (FIFO) scheduling discipline. Initially, it computes the VT of edge server locations. For each edge server, a limited number of neighboring edge servers with the maximum available resources are then determined to prevent excessive computational overhead. Finally, the user tasks associated with the Voronoi region of a specific edge server are assigned to a neighboring server with the most available resources. The EVBLB algorithm operates on a fixed time interval (ΔT) for allocating incoming requests, which emerges as a practical limitation. Specifically, this rigid timing structure can lead to suboptimal request distribution under dynamic network conditions. Subsequently, EVBLB adopts a neighbor selection algorithm that selects an appropriate neighboring edge server for each incoming request based on the remaining resource capacity and the spatial distribution defined by the Voronoi diagram. However, this approach introduces a significant drawback. The continuous monitoring of neighboring servers' resource availability, necessary for each execution cycle, results in substantial time overhead. This process not only delays the request handling but also centralizes the execution of the algorithm.

With regard to optimization techniques, [18] proposed CooLoad, an LB algorithm for edge networks based on finite quasi-birth-and-death Markov process. In this approach, each edge server acts as a neighbor to other edge servers and maintains a queue with a predefined threshold. When the queue size exceeds the threshold, incoming user requests are transferred to the neighboring server. The main contribution of this article lies in the dynamic threshold update algorithm employed for each server, which significantly reduces the blocking rate. The work in [19] introduces a task deployment approach aimed at LB in edge networks. The article presents a clustering method to identify heavy tasks, which are then sent to remote cloud servers for processing. The remaining tasks are distributed among the edge servers using a heuristic algorithm based on the glowworm swarm optimization (GSO) [20]. The algorithm incorporates a dynamic step size to address the issue of premature convergence to local optima observed in the basic GSO, thus resulting in more accurate solutions with increased iterations.

The LB algorithm analyzed in [21] distributes the load across edge servers based on the data volume generated by sensors within the server's designated area and the number of user queries for the data. Nonetheless, the authors do not explicitly formulate the LB problem. In [22], the authors employ a distributed constraint satisfaction (DisCSP) [23] technique to formulate their proposed LB algorithm. The two algorithms, namely the distributed constraint-based diffusion (CDIFF) and the distributed routing-based diffusion (RDIFF) algorithms select edge servers that meet all the requirements of incoming requests while considering the capacity of the edge servers. In [24], the authors formulate the LB problem in a mobile EC network as a graph coloring problem, wherein the vertices, edges, and colors of the graph correspond to mobile devices, nearby discoverable mobile devices, and edge servers, respectively. They propose a genetic algorithm for solving the graph coloring, thereby enabling the distribution of tasks to the nearest edge server.

ML-based algorithms for LB in EC networks are relatively scarce. A QL algorithm for LB within an LTE network is proposed in [25], where the authors focus on balancing user tasks between a serving cell (S-cell) and its neighboring cells by taking into account the cell individual offset (CIO) parameter. By utilizing QL, the algorithm dynamically adjusts the load distribution to enhance network performance. Similarly, the authors of [26] used deep reinforcement learning (DRL) algorithms for task offloading in EC networks. In particular, they used a graph of tasks and utilized a deep neural network (DNN) to learn an optimized offloading policy. Building upon the EVBLB algorithm, this paper extends its underlying principles and presents three novel ML-based LB algorithms to overcome its shortcomings.

3. System model

We consider an EC network consisting of a set of n edge servers, denoted as $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$, along with a collection of m cloud servers, denoted as $C = \{c_1, c_2, \dots, c_m\}$, and a group of BSs, denoted as $\mathcal{B} = \{b_1, b_2, \dots, b_k\}$. Users submit their network service requests through the nearest BS. Following the research effort in [20], the system operates in episodic intervals, where the user requests within each interval of length ΔT are aggregated at the receiving BS. At the end of each interval, the service requests are distributed to appropriate computing servers in $\mathcal{E}$ or C. Each edge server $e_i \in \mathcal{E}$ is characterized by three types of resources, namely R_i^{mem} that represents the total capacity of primary memory in gigabytes (GB), R_i^{dsk} that denotes the disk capacity in GB, and R_i^{cpu} that indicates the processing speed of e_i in gigahertz (GHz). This mode of service provision enables the system orchestrator to achieve improved LB and service quality. Let the set $\mathcal{R}_{\Delta T} = \{r_1, r_2, \dots, r_w\}$ denote the ensemble of user requests that arrive at the edge servers within the time interval, ΔT. Each request $r_j \in \mathcal{R}_{\Delta T}$ has service demands that require primary memory (T_j^{mem}), disk storage (T_j^{dsk}), and processing capacity (T_j^{cpu}). Moreover, let $\mathcal{R}^i_{\Delta T}$ be the set of all user requests received by edge server $e_i \in \mathcal{E}$ in the interval ΔT. Each edge server e_i is equipped with a queue $Q_i = \{q_1, q_2, \dots, q_l\}$, where $l < w$, to accumulate incoming user requests during ΔT and assign them to appropriate edge servers based on the LB approach. The number of requests residing in queue Q_i is denoted as $|Q_i|$. It is worth noting that to compute the remaining CPU resources, we subtract the total number of cycles required for executing all tasks in Q_i from the total number of cycles that the server can perform over the ΔT time interval. Table 1 summarizes the main notations used in this paper.

Table 1
List of main notations.

Notation	Description
e_i	Edge server i from set $\mathcal{E}$, where $\lvert\mathcal{E}\rvert = n$
c_i	Cloud server i from set C, where $\lvert C\rvert = m$
b_i	Base station i from set $\mathcal{B}$, where $\lvert\mathcal{B}\rvert = k$
v_i	Voronoi region i from VT set $\mathcal{V}$
r_i	User request i from set $\mathcal{R}_{\Delta T}$, where $\lvert\mathcal{R}_{\Delta T}\rvert = w$
$\mathcal{R}^i_{\Delta T}$	Set of user requests at e_i during ΔT
ΔT	Time interval between LB algorithm executions
$\Delta T_{\max}$	Maximum feasible value of ΔT
R_i^{mem}	Memory capacity of edge server $e_i \in \mathcal{E}$
R_i^{dsk}	Disk capacity of edge server $e_i \in \mathcal{E}$
R_i^{cpu}	Processing capacity of edge server $e_i \in \mathcal{E}$
T_i^{mem}	Memory required for user task $r_i \in \mathcal{R}_{\Delta T}$
T_i^{dsk}	Disk required for user task $r_i \in \mathcal{R}_{\Delta T}$
T_i^{cpu}	Processor required for user task $r_i \in \mathcal{R}_{\Delta T}$
Q_i	Task queue of edge server $e_i \in \mathcal{E}$, where $\lvert Q_i\rvert = l$
u_i	CPU utilization of edge server $e_i \in \mathcal{E}$
$\mathcal{N}_i$	Neighboring server set of edge server $e_i \in \mathcal{E}$

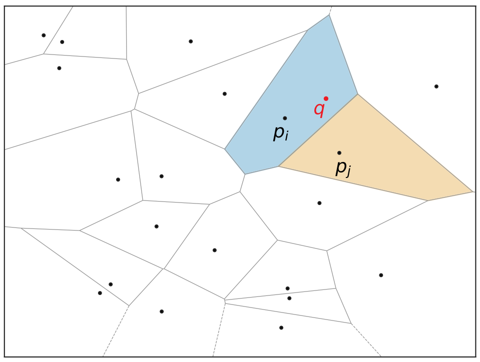


Fig. 2. Schematic of a simple Voronoi tessellation.

3.1. Definition of voronoi tessellation

Voronoi diagram (VT) is a fundamental data structure in computational geometry that divides a plane based on the nearest neighbor rule [14]. Given a set $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$ of points, the VT represents a partitioning of the Euclidean plane into regions $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$, where a point q belongs to the region corresponding to a point p_i if $d(p_i, q) < d(p_j, q)$ for any $j \neq i$. Fig. 2 depicts the line segments that define the regions in a VT, which are equidistant to the two nearest points. As the distance between the center points decreases, the associated regions become smaller, thus resulting in a partition that aligns with the density of the centering points. More details on linear-time algorithms for VT computation can be found in [14,27].

3.2. Problem formulation

We aim to address the LB problem in EC networks by efficiently allocating edge and cloud resources to user requests accumulated within a specified time period ΔT (denoted as $\mathcal{R}_{\Delta T}$), while ensuring a balanced load distribution among edge servers. Without loss of generality, we assume that each user task is explicitly assigned to either an edge server or a remote cloud server. Following the definition in [13], we adopt a performance metric, termed the *load balancing factor* (LBF), which quantifies the degree of LB across the edge servers. The LBF metric given below specifically captures the variance of the load (i.e., utilization of CPU as the main resource of servers) among all the edge servers:

$$\text{LBF} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(u_i - \bar{u}\right)^2}, \tag{1}$$

where u_i denotes the CPU utilization of edge server e_i and is defined as the minimum of 1 and the ratio of total processing resource requirements of queued requests to the total processing power of e_i over time interval ΔT. Formally, $u_i \triangleq \min\{1, \sum_{j=1}^{|Q_i|} T_j^{\text{cpu}}/(R_i^{\text{cpu}} \cdot \Delta T)\}$. This ensures that u_i reflects full capacity utilization (value of 1) when the requested resources exceed or match the server's capacity within ΔT. The mean CPU utilization across all servers is represented as $\bar{u}$. To minimize the LBF metric, LB in our system setting can be formulated as the linear programming (LP) optimization problem given below:

$$\text{Minimize} \quad \text{LBF} \tag{2}$$

subject to:

$$\sum_{i=1}^{n}\left(R_i^{\text{mem}} - T_j^{\text{mem}}\right)X_{i,j} \geq 0; \quad X_{i,j} \in \{0, 1\}, \tag{3}$$

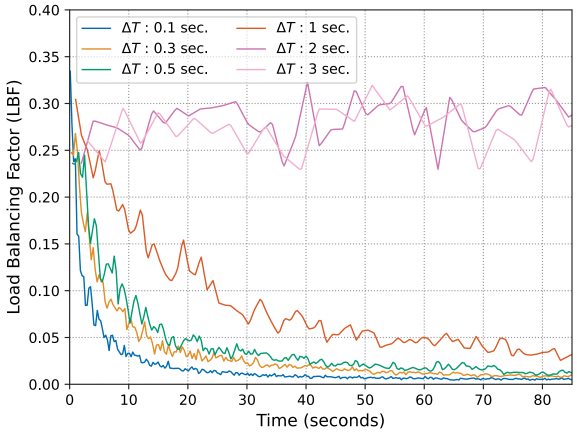


Fig. 3. Impact of ΔT values on LBF in the EVBLB algorithm.

$$\sum_{i=1}^{n}\left(R_i^{\text{dsk}} - T_j^{\text{dsk}}\right) X_{i,j} \geq 0; \quad X_{i,j} \in \{0,1\}, \tag{4}$$

$$\sum_{i=1}^{n} X_{i,j} = 1, \tag{5}$$

for all $r_j \in \mathcal{R}_{\Delta T}$, $j = \{1,2,\dots,w\}$, and for all $e_i \in \mathcal{E}$, $i = \{1,2,\dots,n\}$. Here, the binary random variable $X_{i,j}$ is equal to 1 only if user request r_j is assigned to server e_i, and is equal to 0 if otherwise. This variable will only apply to all tasks within their corresponding regions and the selected servers from the neighbors to which these tasks are assigned; not the other way around. In fact, (3) specifies that when any request r_j is allocated to server e_i (i.e., $X_{i,j} = 1$), the memory capacity of e_i must exceed the memory demand of r_j. A similar requirement for disk resources is given in (4). Additionally, (5) guarantees that each request is assigned to exactly one server. By only accounting for the selected servers, the CPU utilization of server e_i can be rewritten as:

$$u_i \triangleq \min\left\{1, \frac{\sum_{j=1}^{w} T_j^{\text{cpu}} \cdot X_{i,j}}{R_i^{\text{cpu}} \cdot \Delta T}\right\}. \tag{6}$$

The main reasons that make EVBLB impractical for real-world applications are its reliance on a fixed time interval, the requirement for comprehensive information about server resources and queued user requests for selecting neighboring servers, and the need for a central dispatcher. In what follows, we introduce our three RL-based algorithms that operate independently on edge servers within the distributed EC network. Unlike EVBLB, there is no need for a central coordinator to dispatch user requests, thereby enhancing scalability and minimizing overhead. Moreover, each edge server can be individually configured based on its specific resource capacity and needs. This level of customization allows for independent LB on each edge server, thus providing greater control and flexibility in the workload distribution process.

4. Proposed load balancing algorithms

In this section, we first present our QL-based algorithm for predicting the optimal time interval (ΔT) for the EVBLB algorithm. Subsequently, we introduce our two ML-based LB algorithms and discuss their distinctive characteristics.

4.1. EVBLB execution time interval

The EVBLB algorithm, like many other time slot-based LB algorithms [28], utilizes static time intervals for its execution. However, in practical scenarios, achieving an effective load balance requires adapting the time interval dynamically to account for changes in the network such as task rate, task size, and the number and power of edge servers.

To showcase this issue, Fig. 3 plots the LBF metric for various time intervals ranging from 0.1 to 3 s, based on the simulation configuration in [13]. The figure illustrates that for intervals of 2 and 3 s, the LBF metric reaches its maximum value, indicating the poor performance of the algorithm. However, for the remaining time intervals, there is minimal variation in the LBF values, thus suggesting that $\Delta T = 1$ s would be a more effective choice due to the overhead associated with executing the algorithm. To determine the optimal interval, we propose a QL-based algorithm that selects the best ΔT from a given set of intervals, thereby dynamically adjusting the execution period of EVBLB in response to changes in user requests over time.

Algorithm 1 Proposed QL-Based Time Interval Selection

```
Input: $\mathcal{S} = \{f_i\}_{i=1}^{p}$ such that $\forall f_i,\ 0 \leq f_i \leq 1$, and $\mathcal{A} = \{\Delta T_j\}_{j=1}^{q}$ such that
    $\Delta T_q = \Delta T_{\max}$.
Output: Optimal $\Delta T_j \in \mathcal{A}$ in each round.
1: Initialize the $p \times q$ Q-table to zero.
2: $a \leftarrow$ Randomly select $\Delta T_j$ from set $\mathcal{A}$.
3: while true do
4:     Execute EVBLB with time interval $a$ to find the corresponding LBF value.
5:     $s \leftarrow$ Apply $\mathcal{F}$(LBF) defined in (10).
6:     $a \leftarrow$ Apply $\epsilon$-greedy policy to find next interval.
7:     Update the Q-table using (8) and (9).
8:     $s \leftarrow$ Assign new state $s' \in \mathcal{S}$.
9:     $a \leftarrow$ Assign new optimal time interval $a' \in \mathcal{A}$.
10: end while
```

4.2. QL-based time interval selection

QL is a well-known model-free reinforcement learning (RL) algorithm used to maximize the cumulative rewards gained by taking actions in given states. It employs a Q-table, a table that stores the values of state–action pairs, initialized with default values based on the problem at hand. During the training process, the table is updated iteratively until the algorithm achieves the desired level of accuracy compared to the optimal value. Formally, the Q-function used to update the Q-table is defined as the following mapping, where $\mathcal{S}$ and $\mathcal{A}$ denote the sets of states and actions, respectively:

$$Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}. \tag{7}$$

The function used to update the Q-table is the Bellman optimality equation, which is defined as follows:

$$Q'(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot \left(\hat{r} + \gamma \cdot \max_{a'} Q(s',a')\right), \tag{8}$$

where $\hat{r}$ is the reward received when transitioning from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ under action $a \in \mathcal{A}$, α is the learning rate ($0 < \alpha \leq 1$), and discount factor γ determines the weight given to future rewards, influencing the importance of long-term versus immediate rewards.

Algorithm 1 outlines our proposed distributed QL-based algorithm which is executed independently on each $e_i \in \mathcal{E}$ to find the optimal time interval. The algorithm takes a range of LBF values, given by the state set $\mathcal{S} = \{f_1, f_2, \dots, f_p\}$, such that $\forall f_i \in \mathcal{S},\ 0 \leq f_i \leq 1$. It also considers a set of actions $\mathcal{A} = \{\Delta T_1, \dots, \Delta T_{\max}\}$, the step size of which can be customized based on the LBF value. For $j = 1,2,\dots,q$, $\Delta T_j \in \mathcal{A}$ represents a positive time interval, and $\Delta T_{\max}$ $(= \Delta T_q)$ denotes the maximum feasible time interval. The Q-table is updated using the following reward function:

$$\hat{\mathcal{R}}\left(f_i, \Delta T_j\right) = \theta(1-f_i) + (1-\theta)\left(\frac{\Delta T_j}{\Delta T_{\max}}\right); \quad 0 \leq \theta \leq 1, \tag{9}$$

where the coefficient θ represents the weighting of the LBF metric compared to ΔT_j. This simple reward function allows us to make trade-offs between our desired performance metrics effectively. The first term in (9) addresses the LBF values (f_i), where a lower f_i indicates better load distribution. This term assigns higher rewards for smaller f_i values, incentivizing more balanced loads. The second term, $(1-\theta)\left(\frac{\Delta T_j}{\Delta T_{\max}}\right)$, considers the time interval ΔT_j relative to the maximum permissible interval $\Delta T_{\max}$. Selecting a longer time interval ΔT_j, which implies prolonged algorithm execution, yields a higher reward.

After initializing all entries of the $p \times q$ Q-table with zeros, the algorithm randomly selects an action $a = \Delta T_j$ from the set $\mathcal{A}$ as the initial state. In line 4, the EVBLB algorithm is executed with the time period of ΔT_j, and the resulting LBF is computed. The following best-fit function $\mathcal{F}(\cdot)$ is then utilized to determine the corresponding state $s = f_i$ for the obtained LBF. Essentially, the function selects the $f_i \in \mathcal{S}$ that is closest to LBF as follows:

$$\mathcal{F}(\mathrm{LBF}) = \underset{f_i \in \mathcal{S}}{\arg\min}\, |\mathrm{LBF} - f_i|. \tag{10}$$

In line 6, we adopt the ϵ-greedy policy [16, pg.100] to select the optimal action in each iteration. Lines 7 to 9 are responsible for updating the Q-table with the current action a and state s to generate the action and state for the next round. The algorithm runs continuously, updating the appropriate time interval according to the reward function in order to achieve the maximum reward.

Next, we present the MAB and the GB algorithms to enhance the neighbor selection process in the EVBLB algorithm. The algorithms strive to select the most optimal neighboring servers based on their performance history. As a result, the best neighbor set of edge server e_i, which we denote by $\mathcal{N}_i$, is chosen to handle incoming tasks in each round. The EVBLB algorithm executes tasks using either a single or double-hop routing strategy, wherein the tasks are processed on either the server itself or one of its neighboring servers. This helps minimize the time spent transmitting user requests to distant servers.

4.3. MAB-based neighbor selection

In this section, we integrate efficient neighbor selection into the EVBLB framework using the well-established MAB algorithm. By employing a reward function to improve the performance metrics, our proposed MAB algorithm allows each edge server to select its neighbor with the maximum reward. This distributed approach introduces a novel LB method in EC networks, where the algorithm's execution time is distributed among the edge servers, resulting in reduced overall execution time for the LB process in each time period. To prioritize the selection of servers from the most recent time intervals, we introduce a selection window, W, in our MAB algorithm. This window plays a crucial role in enhancing the efficiency of our algorithm by disregarding time periods that significantly differ from the current network behavior. Additionally, it allows for assigning higher importance to the selection of servers in recent times, thereby providing a more accurate reflection of the network dynamics.

To formulate the bandit problem, we first define the binary set $\mathcal{Y}_i = \{y_{i,1}, \dots, y_{i,t}, \dots, y_{i,|W|}\}$ for each edge server e_i and window size $|W|$, to indicate whether or not e_i is selected in the tth round of the algorithm. That is to say, $y_{i,t} = 1$ if e_i is selected in the tth round and 0, if otherwise. Similarly, we also define $\mathcal{Z}_i$ as a list of rewards received by server e_i during the execution of the algorithm. The tth element of $\mathcal{Z}_i = \{\hat{r}_{i,t}\}$ can take on values between $\beta_- \le \hat{r}_{i,t} \le \beta_+$, corresponding to the reward obtained in round t, where $1 \le t \le |W|$, and reward $\hat{r}_{i,t}$ is given as:

$$\hat{r}_{i,t} = \begin{cases} \beta_-; & \text{if } e_i \text{ blocks input requests,} \\ \beta_+ \left(\frac{\mathcal{T}^i_{\mathrm{thr}} - \mathcal{T}^i_{\mathrm{rsp}}}{\mathcal{T}^i_{\mathrm{thr}}}\right)^+; & \text{if otherwise.} \end{cases} \tag{11}$$

where the threshold $\mathcal{T}^i_{\mathrm{thr}}$ determines the maximum tolerable time for executing incoming user requests on edge server e_i and $\mathcal{T}^i_{\mathrm{rsp}}$ denotes the time required by e_i to process all input user requests, taking into account its own request queue. If the request queue of e_i (i.e., $\mathcal{Q}_i$) is full or would become full upon accepting new requests, it is unable to process all the input requests, and thus, the algorithm imposes a penalty, denoted by β_-. Conversely, if the requests can be executed, the reward is calculated based on the difference between the maximum threshold $\mathcal{T}^i_{\mathrm{thr}}$ and the actual processing time $\mathcal{T}^i_{\mathrm{rsp}}$, scaled by a positive coefficient β_+. The expression $(\cdot)^+$ ensures that the reward is non-negative. Let $\hat{Y}_i$ be the number of times e_i was selected within the W time frame and $\hat{Z}_i$ be the sum of rewards obtained as a result of choosing e_i within window W. Thus, we have:

$$\begin{cases} \hat{Y}_i = \sum_{t=1}^{|W|} \mathcal{Y}_i(t), \\ \hat{Z}_i = \sum_{t=1}^{|W|} \mathcal{Z}_i(t). \end{cases} \tag{12}$$

Our MAB algorithm adopts the Kullback Leibler upper confidence bound (KL-UCB) policy [16,29] to calculate an upper confidence bound on the expected reward for each arm and select the arm yielding the highest bound in each round. The function below computes this bound for edge server e_i by identifying the smallest probability distribution λ that satisfies the KL divergence constraint [30]:

$$\text{KL-UCB}_{\hat{Y}_i, \hat{Z}_i, t} = \max\Big\{\lambda \in \Theta : \mathcal{D}_{\mathrm{KL}}\Big(\frac{\hat{Z}_i}{\hat{Y}_i}, \lambda\Big) \cdot \hat{Y}_i \le \log(t) + \bar{c}\log\big(\log(t)\big)\Big\}, \tag{13}$$

where Θ denotes the set of probability distributions, the parameter $\bar{c} > 0$ governs the exploration–exploitation trade-off, and $\mathcal{D}_{\mathrm{KL}}(\cdot,\cdot)$ is the KL divergence between two distributions.

Algorithm 2 presents our proposed MAB algorithm for selecting efficient neighbors in EVBLB. Line 4 loops over all edge servers to find their efficient neighbors. Lines 5–9 handle the case where none of the neighbors of an edge server have been selected within the time window W of size $|W|$. In such cases, a random neighbor is chosen, thus prioritizing selected servers within the window. Subsequently, lines 11–17 select the neighboring server with the maximum reward based on the KL-UCB policy for user request assignment. This is followed by updating the sets $\mathcal{Y}_i$ and $\mathcal{Z}_i$ in lines 18–19. Once an efficient neighbor is found, lines 21–24 initialize the values of $\mathcal{Y}_i$ and $\mathcal{Z}_i$ for the other neighbors, ensuring the selection of another neighboring edge servers in the subsequent round. Finally, line 26 ensures that the counter t does not exceed the window size by using the modulo operation.

4.4. GB-based neighbor set selection

To effectively employ EVBLB, it is essential to continuously monitor the disk, memory, and processing capabilities of the servers. Recognizing this requirement, we develop another novel LB algorithm that prioritizes server selection based on past performance. Among the various solutions, one effective approach is the GB [17] algorithm that employs gradient ascent to determine the optimal arm to select. In essence, we maintain a mean reward variable that tracks the average reward obtained over time. If a selected bandit provides rewards higher than the average, its likelihood of being chosen increases, and vice versa. The fundamental GB algorithm is well-suited for non-stationary scenarios, where the reward distribution of each edge server varies over time.

Our GB algorithm addresses two key challenges in the EVBLB framework. First, we encounter the decision-making task of selecting the best edge server for each Voronoi region. Secondly, we aim to avoid direct information exchange with the neighboring servers of each edge server. To tackle these challenges, our proposed GBA algorithm distributes the user requests across edge servers and utilizes their feedback to make informed decisions for subsequent server selections. We consider each

Algorithm 2 Proposed MAB-Based Neighbor Selection

```
Input: Sets of edge servers $\mathcal{E}$, user requests $\mathcal{R}_{\Delta T}$, neighbor set $\mathcal{N}_i, \forall e_i \in \mathcal{E}$, and VT.
Output: Optimal neighbor $e_j \in \mathcal{N}_i, \forall e_i \in \mathcal{E}$, in each round.
 1: Initialize window $W$, $\mathbf{Y} \triangleq \{\mathcal{Y}_i\}_{i=1}^{n}$, and $\mathbf{Z} \triangleq \{\mathcal{Z}_i\}_{i=1}^{n}$ such that
    $\forall e_i \in \mathcal{E}, |\mathcal{Y}_i| = |\mathcal{Z}_i| = |W|$.
 2: while true do
 3:   $t \leftarrow 1$
 4:   for all $e_i \in \mathcal{E}$ do
 5:     if $\sum_{k=1}^{|W|} \mathcal{Y}_i(k) = 0$ then
 6:       $e_{i'} \leftarrow$ Randomly select a server from set $\mathcal{N}_i$.
 7:       $\hat{r}_{i',t} \leftarrow$ Compute reward by assigning all tasks in the Voronoi
          region of $e_i$ to $e_{i'}$.
 8:       $\mathcal{Y}_i(t) \leftarrow 1$
 9:       $\mathcal{Z}_i(t) \leftarrow \hat{r}_{i',t}$
10:     else
11:       for all $e_j \in_{j \neq i} \mathcal{N}_i$ do
12:         $\hat{Y}_j \leftarrow \sum_{k=1}^{|W|} \mathcal{Y}_j(k)$
13:         $\hat{Z}_j \leftarrow \sum_{k=1}^{|W|} \mathcal{Z}_j(k)$
14:         $\pi_j \leftarrow$ Apply KL-UCB$_{\hat{Y}_j,\hat{Z}_j,t}$ defined in (13).
15:       end for
16:       $e_{i'} \leftarrow \arg\min_{e_k \in \mathcal{N}_i} \pi_k$
17:       $\hat{r}_{i',t} \leftarrow$ Compute reward by all tasks in the corresponding
          Voronoi region to $e_{i'}$.
18:       $\mathcal{Y}_i(t) \leftarrow 1$
19:       $\mathcal{Z}_i(t) \leftarrow \hat{r}_{i',t}$
20:     end if
21:     for all $e_j \in_{j \neq i} \mathcal{N}_i \backslash \{e_{i'}\}$ do
22:       $\mathcal{Y}_i(t) \leftarrow 0$
23:       $\mathcal{Z}_i(t) \leftarrow 0$
24:     end for
25:   end for
26:   $t \leftarrow (t+1) \mod |W|$
27: end while
```

edge server $e_i \in \mathcal{E}$ to have an action set $\mathcal{A}_i = \{a_1, \dots, a_j, \dots, a_{|\mathcal{N}_i|}\}$, which represents the selection of a neighboring edge server, for which it is correspondingly rewarded. Assuming that action $a_j \in \mathcal{A}_i$ is chosen in round t, the reward for this action is obtained similar to (11), as:

$$\hat{\mathcal{R}}_t^i(j) = \begin{cases} \beta_-; & \text{if } e_j \in \mathcal{N}_i \text{ blocks request from } e_i, \\ \beta_+ \left(\frac{\mathcal{T}_{\text{thr}}^{i,j} - \mathcal{T}_{\text{rsp}}^{i,j}}{\mathcal{T}_{\text{thr}}^{i,j}} \right)^+; & \text{if otherwise.} \end{cases} \tag{14}$$

where the threshold $\mathcal{T}_{\text{thr}}^{i,j}$ determines the maximum tolerable time for executing incoming user requests from edge server e_i on server $e_j \in \mathcal{N}_i$, and $\mathcal{T}_{\text{rsp}}^{i,j}$ denotes the time required by e_j to process all input user requests, taking into consideration its own request queue. If the request queue of e_j (i.e., Q_j) is full or would become full upon accepting new requests, it is unable to process all the input requests, and thus, the algorithm imposes a penalty, denoted by β_-. Conversely, if the requests can be executed, the reward is calculated based on the difference between the maximum threshold $\mathcal{T}_{\text{thr}}^{i,j}$ and the actual processing time $\mathcal{T}_{\text{rsp}}^{i,j}$, scaled by a positive coefficient β_+. The notation $(\cdot)^+$ represents the positive part of the value.

Now, let the average of all the rewards accumulated up to and including round t for $e_j \in \mathcal{N}_i$ be denoted by $\bar{\mathscr{R}}_t^i(j)$. The main objective is to maximize the average rewards obtained from selecting neighbors for each edge server in every execution round of the EVBLB algorithm. In each iteration, our algorithm selects the neighbor with the highest probability for any given edge server e_i, where the probability of selecting an arbitrary neighboring edge server $e_j \in \mathcal{N}_i$ at time t is calculated as follows:

Algorithm 3 Proposed GB-Based Neighbor Selection

```
Input: Sets of edge servers $\mathcal{E}$, user requests $\mathcal{R}_{\Delta T}^i, \forall e_i \in \mathcal{E}$, neighbor set $\mathcal{N}_i, \forall e_i \in \mathcal{E}$, and VT.
Output: Optimal neighbor $e_j \in \mathcal{N}_i, \forall e_i \in \mathcal{C}$, in each round.
 1: $\forall e_i \in \mathcal{E}$ and $e_j \in \mathcal{N}_i$, initialize $\hat{R}_0^i(j)$ and $H_0^i(j)$ to zero.
 2: while true do
 3:   $t \leftarrow 0$
 4:   for all $e_i \in \mathcal{E}$ do
 5:     for all $e_j \in \mathcal{N}_i$ do
 6:       Compute reward $\hat{R}_t^i(j)$ using (14).
 7:       Compute probability $\pi_t^i(j)$ using (15).
 8:     end for
 9:     $e_{i'} \leftarrow \text{argmax}_{e_k \in \mathcal{N}_i} \pi_t^i(k)$
10:     Assign all user requests in the corresponding Voronoi region to
        $e_{i'}$.
11:     Compute $\hat{\mathcal{R}}_{t+1}^i(i')$ using (14).
12:     Compute $H_{t+1}^i(i')$ using (16).
13:     for all $e_j \in_{j \neq i} \mathcal{N}_i \backslash \{e_{i'}\}$ do
14:       $\hat{R}_{t+1}^i(j) \leftarrow \bar{\mathscr{R}}_t^i(j)$
15:       Compute $H_{t+1}^i(j)$ using (16).
16:     end for
17:   end for
18:   $t \leftarrow t+1$
19: end while
```

$$\pi_t^i(j) = \frac{\exp\left(H_t^i(j)\right)}{\sum_{e_k \in \mathcal{N}_i} \exp\left(H_t^i(k)\right)}, \tag{15}$$

where $H_t^i(j)$ is a learned preference for taking action $a_j \in \mathcal{A}_i$ to determine the optimal action in each iteration (as shown in line 7 of Algorithm 3). To update $H_t^i(j)$ in each iteration of the GB algorithm, we employ the stochastic gradient ascent (SGA) algorithm [16,31] as follows, where $\delta > 0$ denotes the step-size parameter, which governs the magnitude of the update made to the estimated action-value function at each time step and it controls the learning rate in the GB algorithm, thus influencing the convergence speed of the algorithm towards the optimal action:

$$H_{t+1}^i(j) = \begin{cases} H_t^i(j) + \delta\left(\hat{\mathcal{R}}_t^i(j) - \bar{\mathscr{R}}_t^i(j)\right)\pi_t^i(j); & \text{if } e_j \in \mathcal{N}_i \text{ is selected,} \\ H_t^i(j) - \delta\left(\hat{\mathcal{R}}_t^i(j) - \bar{\mathscr{R}}_t^i(j)\right); & \text{if otherwise.} \end{cases} \tag{16}$$

Algorithm 3 outlines the steps taken by our GB algorithm to determine the optimal neighboring servers for each edge server. In lines 5–8, the algorithm computes the probability of selecting each neighbor for a given edge server using (14) and (15). It then selects the neighbor with the highest probability and processes input requests from that particular edge server in lines 9–12. The reward for this selected neighbor is calculated using (14). In lines 13 through 16, the rewards of non-selected neighbors are set to $\bar{\mathscr{R}}_t^i(\cdot)$, representing their average rewards. Additionally, the function $H_t^i(\cdot)$, which determines the preference of each neighbor, is updated using (16).

5. Complexity analysis

In this section, we investigate the complexity of the proposed algorithms in terms of the number of states ($|\mathcal{S}|$) and actions ($|\mathcal{A}|$), the convergence rate of the Q-values, and the number of iterations needed to achieve a near-optimal solution. Note that the time complexity of EVBLB in each round is approximated as $O\left(|\mathcal{E}| \cdot \log(|\mathcal{E}|) + |\mathcal{R}_{\Delta T}|\right)$ [13].

The most time-consuming step in the QL-based algorithm is updating the Q-values, which has a time complexity proportional to the number of state–action pairs explored while learning. Formally, this complexity can be expressed as $O(|\mathcal{S}|\cdot|\mathcal{A}|)$, where $|\mathcal{S}|$ and $|\mathcal{A}|$ denote the number of states and actions, respectively. The convergence rate of the Q-values depends on the learning rate (α) and discount factor (γ). Smaller α and larger γ lead to slower convergence but more accurate estimation of optimal Q-values. As a result, the number of iterations needed to achieve near-optimality varies depending on the values of α, γ, and problem complexity [16,32]. It is worth noting that the time complexity of the ϵ-greedy policy used for action selection is negligible compared to updating the Q-value. Therefore, the overall time complexity of the QL-based algorithm can be approximated as $O(|\mathcal{S}|\cdot|\mathcal{A}|\cdot\hat{n})$, where $\hat{n}$ is the number of iterations required to achieve near-optimality.

In the MAB algorithm, the time complexity should be analyzed for each round on every edge server. In each round, the algorithm calculates the KL-UCB index for every neighboring server. In turn, the time complexity of the KL-UCB algorithm per round can be divided into two steps: computing the KL divergence and minimizing the function over Θ. Typically, the KL divergence computation takes $O\left(\log(n_b)\right)$ time, where n_b is the number of bins used to discretize the distribution. Minimizing the function over λ can be done efficiently using binary search, which incurs a time complexity of $O\left(\log(n_r)\right)$, where n_r denotes the total number of rounds. Hence, the overall complexity of the KL-UCB algorithm per round is of the order $O\left(\log(n_b)\log(n_r)\right)$ [30]. However, other factors such as the size of the state space or v (defined in [13] as the maximum number of neighbors for any edge server) may contribute to the overall time complexity. As a result of this, the time complexity of the proposed MAB-based algorithm with KL-UCB can be approximated as $O\left(|\mathcal{E}|\left(|W|+v\left(|W|+\log(n_b)\log(n_r)\right)\right)\right)$.

The time complexity analysis for each round of the proposed GB algorithm can be divided into two parts. The first part involves calculating the probabilities $\pi_t^i(j)$ for any neighboring edge server $e_j\in\mathcal{N}_i$. This, in turn, requires computing the exponential functions $\exp\left(H_t^i(\cdot)\right)$ in (15). The time complexity of this part is upper bounded by $O(v^2)$. The second part deals with selecting the best edge server $e_{i'}$ based on the calculated probabilities and assigning all tasks in the Voronoi region to $e_{i'}$, which takes time $O(v)$. Eventually, updating the value of $\hat{\mathcal{R}}_t^i(j)$ based on the received rewards for any $e_j\in\mathcal{N}_i$ incurs a time complexity of $O(v^2)$. Therefore, the time complexity of each round is $O(v^2)$, and the total complexity of the algorithm will be in the order of $O(|\mathcal{E}|\cdot v^2)$.

The EVBLB algorithm selects neighbors by calculating the remaining resources of each server, which incurs a high computational overhead. It examines the entire state space to choose the best solution, which is selecting servers with the most remaining resources. However, this approach requires obtaining information about all requests in the queue of each neighbor server $e_j\in\mathcal{N}_i$, resulting in a time complexity of $O(|\mathcal{Q}_i|)$ for every server $e_i\in\mathcal{E}$. Therefore, selecting the best neighbor for any arbitrary edge server in EVBLB has a time complexity of $O(\sum_{e_j\in\mathcal{N}_i}|\mathcal{Q}_j|)$, meaning that it can be significantly time-consuming if neighboring servers have long request queues. On the other hand, the MAB and GB-based algorithms proposed in this work do not require storing information about the edge servers, including server resources or request queues. Instead, these algorithms aim to find the optimal neighbor based on the rewards received from the edge servers. As a result, the optimal neighbor can be selected within an acceptable execution time that is comparable with EVBLB.

6. Simulation results and discussions

In this section, we benchmark the performance of the proposed LB algorithms against the EVBLB algorithm in [13]. To perform the evaluation, the edge network is emulated using Mininet [33], and the algorithms are implemented in Python to run on the edge servers. The set size $|\mathcal{E}|$ is within the range of $[10,100]$ and ΔT is taken to be 2 s. The request generation rate is uniformly distributed between 100 and 200 requests per second. Each edge server is allocated $\{4,8,16,32\}$ GB of memory, and processor speeds ranging from 1.0 to 4.0 GHz.

Table 2
Simulation parameter settings.

Parameters	Value
Edge server set size ($\vert\mathcal{E}\vert$)	$[10,100]$
R_i^{mem}	$\{4,8,16,32\}$ GB
R_i^{dsk}	$\{500,1000,2000\}$ GB
R_i^{cpu}	$\{1,1.1,\ldots,4\}$ GHz
$\mathcal{R}_{\Delta T}$	$[100,200]$
T_i^{mem}	$[10,100]$ MB
T_i^{dsk}	$[100,1000]$ MB
T_i^{cpu}	$[100,1000]$ MHz
$\Delta T/\Delta T_{\max}$	2/10 s
$\mathcal{S}$	$\{0.01,0.02,\ldots,1\}$
$\mathcal{A}$	$\{0.1,0.2,\ldots,10\}$ s
Max. no. iterations to find $\mathcal{N}_i$ ($\bar{n}$)	50
Max. no. neighboring servers (v)	5
Max. domain radius (Ψ)	5
Step size (δ)	0.01
Max. execution time ($\mathcal{T}_{\text{thr}}^{i,j}$)	30 s
$\alpha/\gamma/\theta$	0.2/0.2/0.5

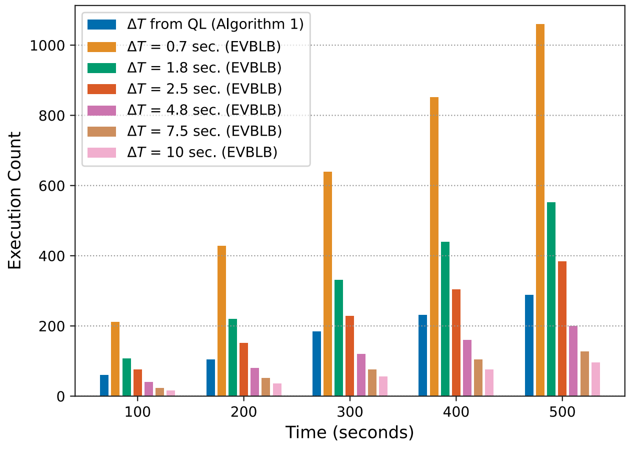


Fig. 4. Temporal comparison of execution count between the proposed QL-based algorithm and the EVBLB algorithm.

The memory usage of user requests is set to be in the range of 10 to 100 MB and processing cycle requirements between 100 and 1000 GHz. Unless explicitly specified, Table 2 summarizes the simulation parameters and their respective values used throughout our simulations. Furthermore, a 95% confidence interval (CI) has been incorporated in Figs. 6, 7, and 8 for better statistical representation.

Fig. 4 illustrates the efficacy of the proposed QL-based algorithm by showcasing its output with respect to different time intervals. Unlike the static intervals employed in EVBLB, our QL-based algorithm leverages the LBF metric to adeptly adjust to dynamic changes in user request rates and thus, predict the optimal future time intervals. Consequently, we can affirm that the QL-based algorithm is capable of predicting intervals without the need for any initial setup. As evident in Fig. 4, the optimal ΔT (=3.3 s) obtained by the QL-based algorithm lies in between the two fixed intervals of 2.5 and 4.8 s. Moreover, we also observe that the QL-based algorithm requires fewer iterations to achieve the optimal interval compared to the next highest fixed interval in the EVBLB algorithm. For instance, after running the two algorithms for 500 s on the edge network, the QL-based algorithm obtains the optimal interval with nearly 28% fewer iterations compared to EVBLB using a fixed interval length of $\Delta T=2.5$ s. Additionally, the QL-based algorithm requires approximately 36% more iterations than using a fixed interval length of $\Delta T=4.8$ s.

Fig. 5 shows how our QL-based algorithm consistently enhances the LBF in response to network dynamics. Initially, it is observed that

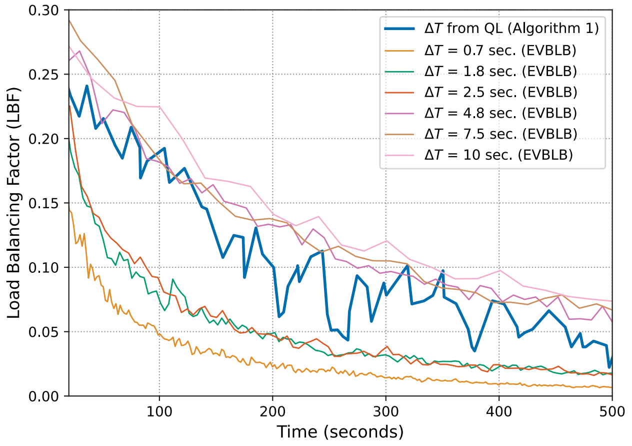
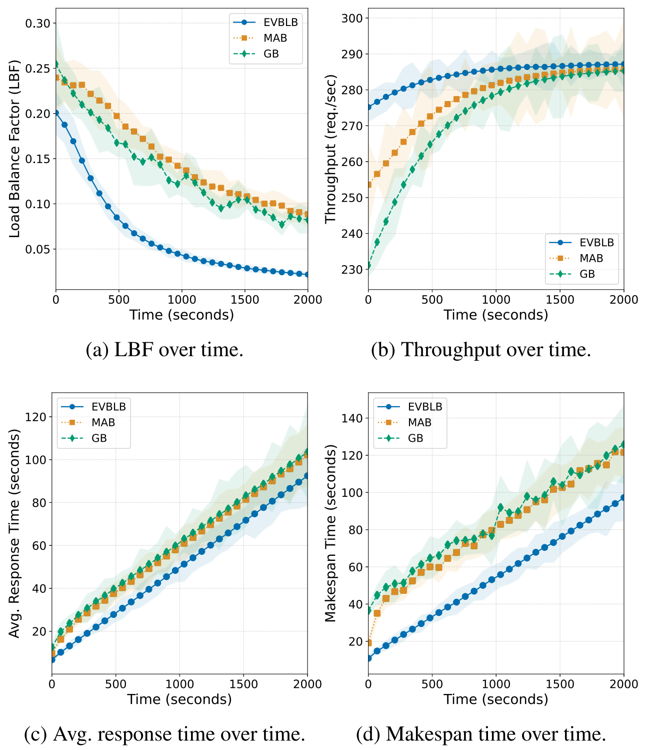


Fig. 5. Temporal comparison of LBF between the proposed QL-based algorithm and the EVBLB algorithm.



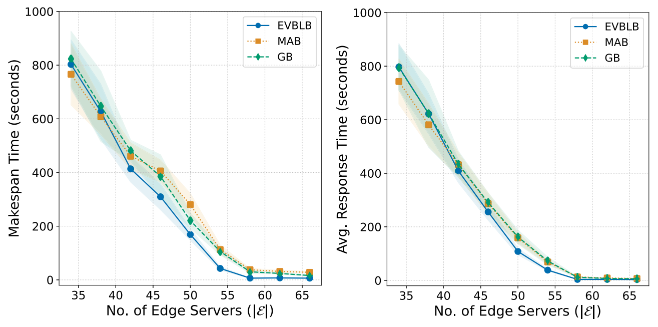
(a) LBF over time. (b) Throughput over time.

(c) Avg. response time over time. (d) Makespan time over time.

Fig. 6. Temporal comparison of the proposed MAB and GB algorithms with EVBLB in terms of various performance metrics.

the LBF is high (close to 0.3), which is attributed to the relatively low number of user requests residing in the majority of server queues. However, as the simulation time progresses and the network load increases, the LBF gradually decreases as larger queues are formed. Such behavior is due to the algorithm's capability to adapt to dynamic changes in the network and determine the optimal ΔT interval using $\theta = 0.5$. While the figure exhibits noticeable fluctuations due to variations in incoming requests, the overall trend of the LBF is downward. Similar to the observations in Fig. 4, we note that the LBF associated with the optimal ΔT obtained through the QL-based algorithm falls is bounded by the LBF values for intervals $\Delta T = 2.5$ and $\Delta T = 4.8$ s.

To compare the performance of the MAB and GB algorithms with the EVBLB algorithm, we measured and analyzed five performance metrics, namely the LBF, throughput (i.e., number of user requests processed per second), the average response time, and the maximum task completion time (also known as makespan time) in Fig. 6. The simulations were conducted over 10 runs using random seeding to obtain the output results. Fig. 6(a) shows the LBF obtained from the three algorithms over time. In the early stages of the simulation (the first 100 s), the LBF remains high despite imperfect LB, primarily due to the distribution of requests among high-capacity servers and underutilized low-capacity edge servers, as well as the basic network settings such as resource capacity of edge servers, user input rate, and execution period ΔT. However, as the high-capacity servers become exhausted, incoming requests are rapidly redirected to low-capacity servers, thus leading to a significant decrease in the LBF and the initiation of LB on the network. According to Fig. 6(a), EVBLB yet performs better than the other two algorithms. This is because it searches comprehensively for the server with the highest available resources, which incurs additional computational overhead. Although there is a slight performance disparity between EVBLB and the other algorithms in the long run, this difference is negligible from a performance perspective since the MAB and GB algorithms demonstrate lower overhead and better interaction with the environment. This marginal difference is due to the fact that a neighbor server may not always possess the maximum remaining resources among all neighboring edge servers. Considering that the MAB and GB algorithms employ similar strategies for selecting the neighboring edge server, their performances are comparable. In Fig. 6(b), we see that EVBLB initially outperforms MAB and GB in terms of throughput. This is due to EVBLB's selection of servers with higher resources, enabling a higher rate of request execution. However, the proposed algorithms gradually improve their performance over time as they learn and select optimal neighboring servers. Consequently, after approximately 1500 s, a marginal difference of 2.4% is observed compared to the EVBLB algorithm. Fig. 6(c) plots the variations in the average response time for the three algorithms over time. The EVBLB algorithm exhibits a lower average response time by distributing requests to servers with higher resources in each round, ensuring they are processed on servers with shorter response times. Conversely, the MAB and GB algorithms, due to their learning behavior over time, may occasionally select servers with lower resources, resulting in longer average execution times for certain input requests. The makespan time of input requests across the three algorithms is shown in Fig. 6(d). The baseline algorithm consistently achieves lower times compared to the other two algorithms due to its preference for servers with higher resources, thus enabling faster execution of tasks in the queue. In contrast, the proposed algorithms exhibit significant fluctuations in makespan due to changes in input requests. However, as these two algorithms identify optimal neighboring servers over time, the fluctuations in maximum execution time diminish. It is worth noting that EVBLB maintains a linear trend with its static server selection policy, while the dynamic behavior of the other two algorithms leads to noticeable fluctuations in the figure.



(a) Makespan time versus $|\mathcal{E}|$. (b) Avg. response time versus $|\mathcal{E}|$.

Fig. 7. Comparison of the makespan time and average response time in terms of the number of edge servers.

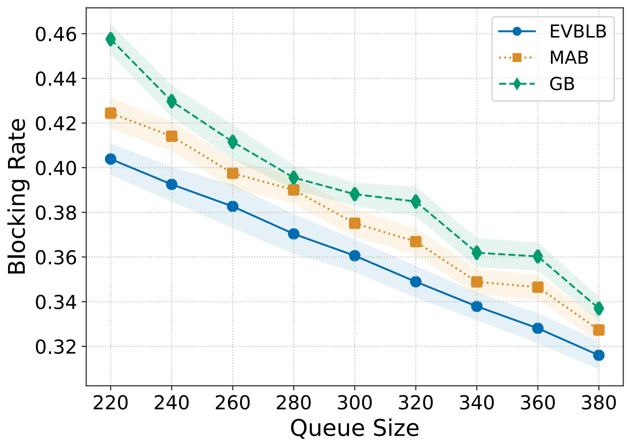


Fig. 8. Comparison of the blocking rate in terms of queue size.

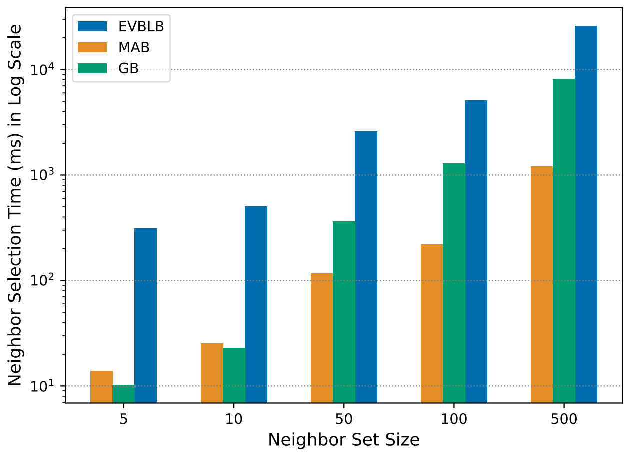


Fig. 9. Neighboring server selection time in terms of neighbor set size.

Fig. 7 depicts the completion time and average response time as functions of the number of resources. In Fig. 7(a), we observe that the makespan time drops at the same rate for all three algorithms. A similar trend is also evident in Fig. 7(b) for the three algorithms. This is a clear indication of their ability to effectively distribute requests to servers and improve performance metrics as the number of servers (and thus, resources) increases. In addition to their effectiveness, these results also demonstrate the scalability of the algorithms in handling high resource levels.

Fig. 8 displays the blocking rate of incoming requests as the capacity of the server queues increases. It is evident that an increase in queue capacity leads to a decrease in the rate of servers being blocked. Specifically, the EVBLB algorithm, which selects servers based on shorter queues (determined by the calculating the remaining resources of the neighboring servers) exhibits a lower blocking rate. As mentioned earlier, due to the static server selection policy, EVBLB demonstrates a linear decrease in the blocking rate. Conversely, MAB and GB show significant fluctuations in their blocking rates. These two ML-based algorithms, which have the potential to select servers with longer queues and limited task execution capabilities, result in higher blocking rates compared to the baseline algorithm. However, as the queue capacity increases, their blocking rates gradually converge towards the blocking rate of the EVBLB algorithm. As depicted in the figure, at a queue capacity of $|Q| = 380$, the blocking rates of MAB and GB algorithms are only negligibly higher, with MAB at 0.33 and GB at 0.339, respectively, compared to that of EVBLB.

Lastly, the performance of the three algorithms in selecting suitable neighbors for each round of execution in a scenario with 500 edge servers and a maximum queue size of 500 is illustrated in Fig. 9. The analysis considers varying numbers of neighbors to assess their impact on time consumption. The experimental results reveal that the two proposed ML-based algorithms exhibit significantly lower time consumption compared to the baseline algorithm. Notably, for scenarios with a small number of neighbors, the GB algorithm outperforms the MAB algorithm. However, as the number of neighbors increases, the time for finding neighboring servers rises in GB due to its inherent time complexity, leading to the superiority of MAB in such cases. These findings clearly highlight the efficiency and scalability of the proposed algorithms in large-scale edge computing environments.

In summary, the results demonstrate that the EVBLB algorithm outperforms the proposed MAB and GB algorithms across all five performance metrics. However, this superior performance comes at the cost of significant overhead, involving the collection of server information, calculating remaining resources, and queue status for server selection in each round. This overhead becomes more pronounced with an increasing number of edge servers and longer queue lengths. On the other hand, the two ML-based algorithms do not require server information collection or resource calculations. They can autonomously learn and adapt to changes in the network and servers over time, making them more distributed and flexible. Moreover, these two algorithms exhibit similar performance in all evaluation metrics. Consequently, the difference in outputs between these two algorithms is negligible, highlighting their efficiency in terms of execution time without the need for extensive server information collection. This makes the proposed dynamic algorithms highly practical for real-world deployment.

7. Conclusion

In this paper, we have focused on efficient load distribution across edge servers with limited resources. In particular, we have proposed three ML-based algorithms to address the shortcomings of the existing EVBLB algorithm for EC environments. The QL-based algorithm has been proposed to predict the optimal time intervals for executing the EVBLB algorithm under varying network conditions, thus striking a balance between execution overhead and LB effectiveness. To enable better neighbor selection based on the resource availability of edge servers, the MAB and GB algorithms have been proposed as further improvements to the EVBLB algorithm. Simulation results have demonstrated significant improvements offered by all three algorithms compared to the base algorithm, thus making them practically feasible. Although the proposed algorithms demonstrate improvements, they are vulnerable to unstable network behavior. To overcome these challenges and further enhance their efficiency, future research endeavors will concentrate on addressing these limitations and developing advanced algorithms. Furthermore, the architecture should consider additional parameters such as bandwidth constraints, energy consumption, and the integration of LB with offloading, among others.

CRediT authorship contribution statement

Mohammad Esmaeil Esmaeili: Conceptualization, Methodology, Software. **Ahmad Khonsari:** Supervision, Writing – review & editing. **Vahid Sohrabi:** Visualization. **Aresh Dadlani:** Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mohammad Esmaeil Esmaeili reports article publishing charges was provided by University of Tehran. Mohammad Esmaeil Esmaeili reports a relationship with University of Tehran that includes: non-financial support.

Data availability

The data that has been used is confidential.

References

[1] T. Dillon, C. Wu, E. Chang, Cloud computing: Issues and challenges, in: 24th IEEE International Conference on Advanced Information Networking and Applications, AINA, 2010, pp. 27–33.

[2] P. Lea, Internet of Things for Architects: Architecting IoT Solutions by Implementing Sensors, Communication Infrastructure, Edge Computing, Analytics, and Security, Packt Publishing Ltd, 2018.

[3] M. Hartmann, U.S. Hashmi, A. Imran, Edge computing in smart health care systems: Review, challenges, and research directions, Trans. Emerg. Telecommun. Technol. 33 (3) (2022) e3710.

[4] S.U. Amin, M.S. Hossain, Edge intelligence and internet of things in healthcare: A survey, IEEE Access 9 (2020) 45–59.

[5] L.U. Khan, I. Yaqoob, N.H. Tran, S.M.A. Kazmi, T.N. Dang, C.S. Hong, Edge-computing-enabled smart cities: A comprehensive survey, IEEE Internet Things J. 7 (10) (2020) 10200–10232.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.

[7] W.Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, A. Ahmed, Edge computing: A survey, Future Gener. Comput. Syst. 97 (2019) 219–235.

[8] W. Yu, F. Liang, X. He, W.G. Hatcher, C. Lu, J. Lin, X. Yang, A survey on the edge computing for the internet of things, IEEE Access 6 (2018) 6900–6919.

[9] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: A survey, IEEE Internet Things J. 5 (1) (2018) 450–465.

[10] H. Pydi, G.N. Iyer, Analytical review and study on load balancing in edge computing platform, in: 4th IEEE International Conference on Computing Methodologies and Communication, ICCMC, 2020, pp. 180–187.

[11] Q. Luo, S. Hu, C. Li, G. Li, W. Shi, Resource scheduling in edge computing: A survey, IEEE Commun. Surv. Tutor. 23 (4) (2021) 2131–2165.

[12] M.H. Kashani, A. Ahmadzadeh, E. Mahdipour, Load balancing algorithms in fog computing, IEEE Trans. Serv. Comput. 16 (02) (2023) 1505–1521.

[13] V. Sohrabi, M.E. Esmaeili, M. Dolati, A. Khonsari, A. Dadlani, EVBLB: Efficient voronoi tessellation-based load balancing in edge computing networks, in: IEEE Global Communications Conference, GLOBECOM, 2021, pp. 1–6.

[14] F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure, ACM Comput. Surv. 23 (3) (1991) 345–405.

[15] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3–4) (1992) 279–292.

[16] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 2018.

[17] T. Lattimore, C. Szepesvári, Bandit Algorithms, Cambridge University Press, 2020.

[18] R. Beraldi, A. Mtibaa, H. Alnuweiri, Cooperative load balancing scheme for edge computing resources, in: 2nd IEEE International Conference on Fog and Mobile Edge Computing, FMEC, 2017, pp. 94–100.

[19] Y. Dong, G. Xu, Y. Ding, X. Meng, J. Zhao, A 'joint-me' task deployment strategy for load balancing in edge computing, IEEE Access 7 (2019) 99658–99669.

[20] K. Krishnanand, D. Ghose, Detection of multiple source locations using a glowworm metaphor with applications to collective robotics, in: IEEE Swarm Intelligence Symposium, SIS, 2005, pp. 84–91.

[21] R. Mogi, T. Nakayama, T. Asaka, Load balancing method for IoT sensor system using multi-access edge computing, in: 6th International Symposium on Computing and Networking Workshops, CANDARW, 2018, pp. 75–78.

[22] K.D. Hoang, C. Wayllace, W. Yeoh, J. Beal, S. Dasgupta, Y. Mo, A. Paulos, J. Schewe, New distributed constraint satisfaction algorithms for load balancing in edge computing: A feasibility study, in: 10th International Workshop on Optimization in Multiagent Systems, OptMAS, 2019, pp. 1–6.

[23] M. Yokoo, T. Ishida, E. Durfee, K. Kuwabara, Distributed constraint satisfaction for formalizing distributed problem solving, in: 12th IEEE International Conference on Distributed Computing Systems, ICDCS, 1992, pp. 614–621.

[24] J. Lim, D. Lee, A load balancing algorithm for mobile devices in edge cloud computing environments, Electronics 9 (4) (2020) 686.

[25] S.S. Mwanje, A. Mitschele-Thiel, A Q-learning strategy for LTE mobility load balancing, in: 24th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications, PIMRC, 2013, pp. 2154–2158.

[26] J. Yan, S. Bi, Y.J.A. Zhang, Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach, IEEE Trans. Wireless 19 (8) (2020) 5404–5419.

[27] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry, Springer Berlin Heidelberg, 2008.

[28] Y. Dai, D. Xu, S. Maharjan, Y. Zhang, Joint load balancing and offloading in vehicular edge computing and networks, IEEE Internet Things J. 6 (3) (2018) 4377–4387.

[29] O. Cappé, A. Garivier, O.-A. Maillard, R. Munos, G. Stoltz, Kullback–Leibler upper confidence bounds for optimal sequential allocation, Ann. Statist. 41 (3) (2013) 1516–1541.

[30] A. Garivier, O. Cappé, The KL–UCB algorithm for bounded stochastic bandits and beyond, in: 24th Annual Conference on Learning Theory, COLT, 2011, pp. 359–376.

[31] D.P. Bertsekas, Nonlinear programming, J. Oper. Res. Soc. 48 (3) (1997) 334–334.

[32] S.D. Whitehead, Complexity and cooperation in Q-learning, in: Machine Learning Proceedings 1991, Elsevier, 1991, pp. 363–367.

[33] MININET - open networking foundation — opennetworking.org, 2023, https://opennetworking.org/mininet/, [Accessed 14-Jun-2023].