Task Placement in Fog Computing Considering User Mobility and Overload

S. Ansari Moghaddam, S. Noferesti*, M. Rajaei

Information Technology Department, Faculty of Electrical and Computer Engineering, University of Sistan and Baluchestan, Zahedan, Iran.
samira.ansary96@gmail.com, snoferesti@ece.usb.ac.ir, rajayi@ece.usb.ac.ir
*Corresponding author

Received: 08/03/2023, Revised: 20/05/2023, Accepted: 14/10/2023.

Abstract
Efficient distribution of service requests between fog and cloud nodes considering user mobility and fog nodes' overload is an important issue of fog computing. This paper proposes a heuristic method for task placement considering the mobility of users, aiming to serve a higher number of requested services and minimize their response time. This method introduces a formula to overload prediction based on the entry-exit ratio of users and the estimated time required to perform current requests that are waiting in the queue of a fog node. Then, it provides a solution to avoid the predicted overloading of fog nodes by sending all delay-tolerant requests in the overloaded fog node's queue to the cloud to reduce the time required for servicing delay-sensitive requests and to increase their acceptance rate. In addition, to prevent requests from being rejected when the mobile user leaves the coverage area of the current fog node, the requests in the current fog node's queue will be transferred to the destination fog node. Simulation results indicate that the proposed method is effective in avoiding the overloading of the fog nodes and outperforms the existing methods in terms of response time and acceptance rate.

Keywords
Fog computing, Task placement, User mobility, Overload prediction.

1. Introduction

We have witnessed much research and significant technological advances since Kevin Ashton used the term Internet of Things (IoT) in 1999 [1]. One of the main challenges of IoT is to provide an appropriate way to store and process the huge amount of data generated by IoT devices. Cloud computing, thanks to providing an almost infinite amount of storage and processing space, can be very useful in addressing such challenges [2]. However, cloud computing is unable to satisfy the requirements for time-critical applications such as smart healthcare due to its unreliable latency and lack of mobility support [3].

Fog computing, which provides computing resources and services at the edges of the network, has been proposed to address the mentioned problems. Fog computing is a distributed computing paradigm that acts as an intermediate layer between the cloud and IoT devices. Data generation and processing can be handled through intelligent networking devices called fog nodes at the fog layer which leads to low-latency access and faster response to application requests when compared to the cloud. Furthermore, fog computing supports mobile applications with different latency requirements.

Fog computing has many research challenges [4]. One of these is task placement (i.e., selection of the appropriate computing node for executing the incoming task) while considering different QoS requirements of different applications. Among these requirements, the latency of deadline-constrained applications is critical. It is expected that this kind of applications can be executed within their respective deadlines. This raises a new challenge for task placement and scheduling: deciding when and where (fog/cloud) to process the application requests to satisfy their requirements.

Task placement and scheduling in cloud computing are well-studied [5, 6]. However, the rapid increase in the number of IoT devices and application requests, limited resources of the fog nodes, and the mobility of users make task placement in fog computing more challenging. With the growing use of mobile devices, the fog computing framework is required to be adapted to the specific needs of mobile users. Mobile users frequently move from the area covered by one fog node to another and their access points to communicate with the fog nodes change. This may affect the latency QoS of IoT services. Due to the limited service coverage of fog nodes, the mobility of users when leaving the coverage area of the serving fog node will cause service loss or excess delay (because of an increase in the hop counts). To maintain the service continuity and to ensure timely service delivery, the requests are preferred to relinquish to a fog node that is closer to the access point to which the user device is connected. However, when this fog node does not have sufficient available resources to support new service requests, these requests should wait in the queue for execution, which increases the time delay and affects the execution deadline constraints of latency-sensitive applications. On the other hand, the movement of users towards a single resource-constrained fog node

and consequently an increase in the number of arriving requests may result in overload on that node. The performance of the overloaded fog node decreases significantly and it will not be able to meet the demands of latency-sensitive applications [7].

Hence, mobility is a significant issue of task placement in fog computing. Previous works to task placement in fog computing have rarely considered user mobility issues. This paper proposes a new method for task placement adapted for mobile users, aiming at maximizing the number of completed requests with respect to their deadline constraints. Simulation results show that considering user mobility can significantly reduce the probability of overloading on fog nodes and improve the acceptance rate of service requests while taking into account their requirements in terms of response time. In summary, the main contributions of this paper are as follows:

- Proposing a new method for distributing the incoming user requests to the computing devices in the network (fog or cloud nodes) while meeting their deadline and minimizing their response time.
- Introducing a method for predicting overload in the fog node, and accordingly deciding to process the delay-tolerant requests in the fog or send them to the cloud in order to increase the acceptance rate, especially for high-priority requests (delay-sensitive ones)
- Considering the mobility of users and providing a solution to prevent requests from being rejected when the mobile user leaves the coverage area of the current fog node, by moving the user requests from the current fog node's queue to the queue of the neighboring fog node.

The organization of the paper is as follows: Section 2 reviews the research for task placement in fog computing. Section 3 describes the system model. Section 4 details the proposed method for task placement in fog computing. Section 5 presents the results of experiments performed to evaluate the proposed method. Finally, Section 6 draws some conclusions.

2. Literature review

In recent years, the issue of task placement in fog computing has been of great interest to researchers. In [8], a comprehensive overview of the existing approaches for task placement was provided. Generally, existing works for task placement can be divided into four exact, heuristic, metaheuristic, and hybrid methods. In this section, we will introduce some of the related works done in each class.

Many works, such as [9], formulated the task placement problem with Integer Linear Programming (ILP). ILP expresses a problem with mathematical constraints and an objective function that can be solved by generic ILP-solvers, which guarantee to return the optimal result. However, exact algorithms are hardly scalable and suffer from high execution time that is exponential with respect to the problem size (number of services and fog nodes).

Tran et al. [10] introduced a systematic fog computing framework consisting of multiple intelligent tiers for the IoT, and provided a context-aware task placement approach to optimize service decentralization on fog computing. This approach that leverages context-aware information such as location, compute and storage capacities of fog devices, and expected deadline of an application, aims at utilizing fog devices available at the network edges, improving the performance of IoT and minimizing the latency, energy consumption, and cost.

Some of the existing works used heuristics for task placement. Xia et al. [11] proposed two backtrack search algorithms (Exhaustive and Naïve) and two heuristics to efficiently make task placement decisions. The Exhaustive search tried to visit all existing solutions for task placement and returned the best solution that minimizes the average response time. The Naive search returned the first found solution. The first heuristic (Anchor-based fog nodes ordering) aimed at minimizing the response time returned by the Naïve search and the second one (Dynamic components ordering) accelerated the search process.

Khosroabadi et al. [12] presented a heuristic algorithm to solve the service placement problem in fog computing for the smart home applications. The main idea of this algorithm was to place latency-sensitive applications as much as possible closer to the IoT devices. To this end, they introduced a hierarchical fog-cloud architecture for optimization of service placement problem, in which the fog nodes were clustered and the horizontal connections between fog nodes were considered.

Natesha and Guddeti [13] introduced two metaheuristics for service placement in fog computing to minimize the service cost and ensure the QoS of Industrial IoT (IIoT) applications: MGAPSO, which was developed by combining the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), and EGAPSO that was developed by combining the Elitism-based GA and the PSO.

Goudarzi et al. [14] proposed an optimal version of the Memetic Algorithm for task placement of multiple IoT devices on appropriate fog/cloud servers to minimize their execution time and energy consumption. The proposed method had three phases: pre-scheduling, batch application placement, and failure recovery. In the pre-scheduling phase, the authors proposed an algorithm to organize the concurrent IoT devices' workflows. In the second phase, they proposed a batch application placement based on the Memetic Algorithm to minimize the weighted cost of each IoT device. In the third phase, they embedded a fast failure recovery method in their method to assign failed tasks to appropriate servers.

The metaheuristic algorithms may fall into the local optimal trap, and they do not guarantee the optimal solutions. In addition, the decision time of these algorithms makes them inefficient for latency-sensitive and large-sized problems.

Some previous works have focused on hybrid methods. Kopras et al. [15] proposed a method for task placement in fog computing aiming at minimizing network energy consumption while meeting delay constraints specific for each offloaded task. They formulated a universal non-convex Mixed-Integer Nonlinear Programming (MINLP) problem to minimize task transmission- and processing-related energy and provided an optimal solution by using the primal and dual decomposition techniques and the

Hungarian algorithm, while the values of the allocation variables were obtained in a heuristic manner.

Sarkar et al. [16] proposed a deadline-aware dynamic task placement (DDTP) strategy that selected the appropriate computing node for each incoming task while meeting the deadline. At first, the DDTP strategy classified the incoming tasks based on some task utilization factors and assigned them to a priority queue based on their deadlines. Then, a new policy was proposed to find an appropriate computing device for meeting the tasks' deadlines. This policy adopted the concept of augmenting path to ensure that the maximum number of tasks was assigned to the corresponding computing device. Finally, a heuristic dispatch-constraint offloading strategy was employed to migrate the failed tasks to another suitable fog node in the same region.

In [17], the joint container placement and task-provisioning problem for a dynamic fog computing environment was formulated as an optimization problem using ILP. Initially, this problem was solved using CPLEX in an optimal way, and then, metaheuristic PSO-based and Greedy heuristic algorithms were designed to solve it. The optimization objective included the maximization of the number of successful users' requests within a predefined time.

According to our studies, previous works for task placement in fog computing mostly ignore the mobility of users. In most of the methods introduced in this section, addressing the issue of mobility in task placement is considered as future work [12,14,16].

In [17], the effect of fog node mobility on task placement was studied. To this end, the authors adopted three scenarios with different mobility patterns: 1) static (5 RSUs), 2) slow mobility (80 cellphones), and 3) fast mobility (40 vehicles). Sets of real-world vehicle and pedestrian traces in Manhattan and Rome were used to simulate the mobility of fog nodes.

In [18], a mobility-aware task offloading and migration model was proposed. In this model, the mobility of users was characterized by the sojourn time in each coverage of fog nodes, which followed the exponential distribution, and task offloading was formulated as a MINLP problem. This problem was divided into two sub-problems: 1) A Gini coefficient-based fog node selection algorithm was proposed to optimize the offloading decisions, and 2) A distributed resource optimization algorithm based on the Genetic algorithm was provided to solve the resource allocation problem. These algorithms could manage user mobility in fog computing by reducing the probability of migration.

The above-mentioned references [17, 18], did not consider the overload of fog nodes due to user mobility, and also the deadline constraints of delay-sensitive applications (users' requests have the same priority).

The closest work to ours is by Peixoto et al. [19]. They offered a simple scenario of user mobility, in which mobile devices move towards a particular cloudlet one by one. They simulated overloading resulting from user mobility in a simplified way by increasing the number of requests in a single fog node. They also proposed the Delay-priority algorithm in which the delay-tolerant requests were forwarded to the cloud to satisfy the requirements of delay-sensitive requests.

Table I presents the overview of the existing works to task placement in fog computing. As mentioned before, mobility support is an inseparable feature of the fog computing architecture. Thus, more research is needed in this area.

3. System model

We used the three-layer architecture of cloud, fog and, IoT devices for fog computing [20]. According to Fig. 1, the lowest layer consists of all the IoT devices that interact with the end users and are responsible for sensing the environment and sending the data/requests to the fog layer. This layer communicates with the fog computing layer through the access points, gateways, etc.

The fog layer is the middle layer that contains several heterogeneous and distributed low-power intelligent devices, referred to as fog nodes, which provide computing, storage, and networking capabilities. Fog nodes are clustered into domains. Each fog node has a limited area of coverage where the desired fog services are provided. All the fog nodes can communicate with neighboring fog nodes of the same domain and with the cloud. Furthermore, fog nodes can handle the mobility issues of the mobile nodes. Each Fog node can control and coordinate the mobile users located within its coverage area.

The cloud layer consists of many high-performance servers and data centers that are capable of storing and processing the huge amount of data.

Table I. Summary of the related works for task placement in fog computing.

Reference	method	Performance metric				User mobility	Deadline-aware	Overload due to the mobility
		Response time	Acceptance/ rejection rate	Energy consumption	Network usage /cost			
[9]	Exact	✓	-	-	-	-	✓	-
[10]	Exact	✓	-	✓	✓	-	✓	-
[11]	Heuristic	✓	-	-	-	-	-	-
[12]	Heuristic	✓	-	✓	✓	-	✓	-
[13]	Metaheuristic	✓	-	✓	✓	-	✓	-
[14]	Metaheuristic	✓	-	✓	✓	-	-	-
[15]	Hybrid	-	✓	✓	-	-	-	-
[16]	Hybrid	✓	-	-	-	-	✓	-
[17]	Hybrid	✓	✓	-	✓	✓	-	-
[18]	Hybrid	-	-	✓	✓	✓	-	-
[19]	Heuristic	✓	-	-	✓	✓	-	-
Ours	Heuristic	✓	✓	-	-	✓	✓	✓

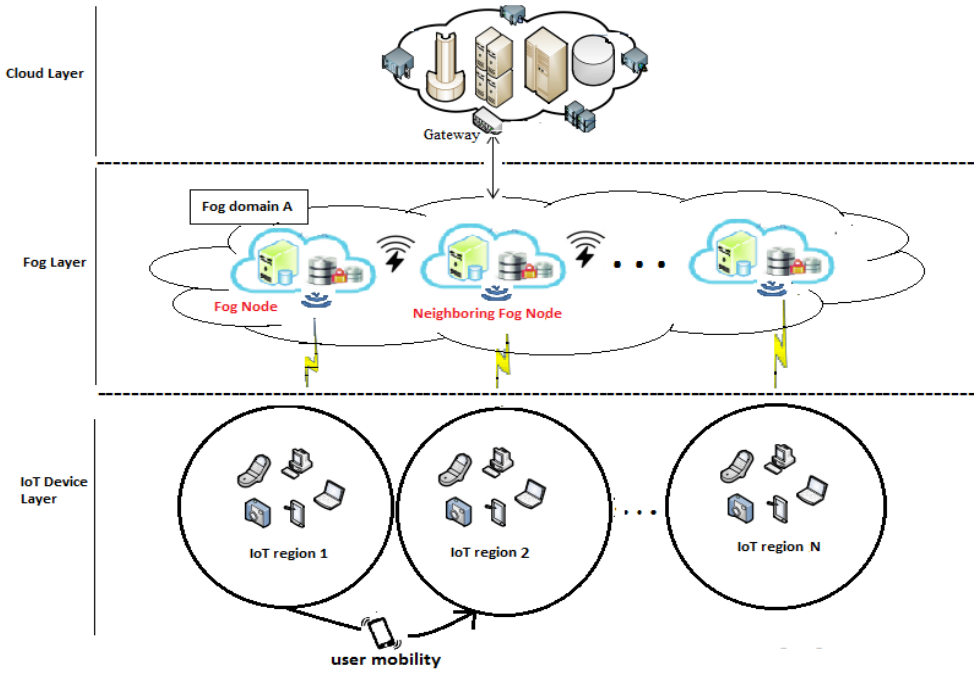

Fig. 1. Three-layer architecture of cloud, fog and IoT devices.

The standardized approach for performing an application request in the IoT systems is as follows: An IoT device sends the service request to the fog layer for processing. In the fog layer, the fog node which has received the request from the IoT device can serve it or may cooperate with other fog nodes in the same domain to execute the request. If not enough resources are available at the fog layer, the fog node can forward the request to the cloud layer.

IoT devices perform several types of applications with different requirements. To show how the quality of task placement can be affected by considering different application classes with different latency requirements, we classify applications based on their ability to tolerate delay into two classes: delay-sensitive and delay-tolerant. The delay-sensitive requests are preferred to be processed in a nearby fog node due to the low latency constraints, while the delay-tolerant ones can be executed in either the fog or the cloud. In fact, the priorities of the requests are determined based on their deadlines.

The three-layer architecture has many benefits. However, task placement and scheduling in this architecture face the following challenges:

- Which node (fog/cloud) and when should process the service requests, according to the type of application (delay-sensitive/delay-tolerant)? How can this decision be made dynamically and quickly?
- How to predict and avoid fog node overloading to maximize the number of satisfied requests and minimize their response time?
- How can task placement be accomplished concerning the mobility of users?

In this paper, we propose an effective method for task placement in fog computing considering the above-mentioned issues. Our system model is composed of one fog domain, hosting N fog nodes, that is defined as: $FD = \{FN_1, FN_2, \ldots, FN_N\}$. Each FN_i is characterized by the total processing capacity P_i and its coverage area R_i. We use the notation C_i to denote the square circumscribed by the coverage area R_i (the diagonal of the square is equal to the diameter of the circle). Each FN_i has a list of neighboring fog nodes $NL_i = \{nl | nl \in FD \text{ and } nl \text{ adjacent to } FN_i\}$ that can forward the user requests to them and a waiting queue (Q_i) for storing the requests arriving from the users or neighboring fog nodes.

We consider a set of mobile users $U = \{U_1, U_2, \ldots, U_M\}$ where each user $U_j = (Vel_j, Dir_j, Loc_j)$ is characterized by its velocity Vel_j, movement direction Dir_j, and geographical location Loc_j. The geographic location of the mobile node is determined using latitude and longitude. The velocity, movement direction, and location of mobile users are updated periodically over the specified time slots with an equal length of Δ_1, denoted by $S = \{S_1, S_2, \ldots, S_z\}$. Each user U_j can generate delay-sensitive and/or delay-tolerant application requests during the time.

Similar to many previous works, for tractability and enabling manageable analysis, we consider a time-discrete system model in which the proposed task placement algorithm starts at the beginning of the predefined time intervals with an equal length of Δ_2, denoted by $T = \{t_1, t_2, \ldots, t_W\}$ ($\Delta_1 \ll \Delta_2$).

Each user U_j periodically generates new application requests. $A_{(id,\ U_j, t_k)} = (L, K, D, W)$ denotes a request with identifier id owned by the user U_j $j \in \{1,2,...,M\}$ at time interval t_k, where L is the number of instructions, K is the class of the requested application (delay-sensitive or delay-tolerant), D is the deadline constraint, and W

determines whether the request received from an IoT device or a neighboring fog node. We assumed that each delay-sensitive request has a hard deadline. Thus, a delay-sensitive request waiting in the queue is rejected by the system after missing the deadline. Delay-tolerant requests do not have deadline constraints but they may fail if the mobile device leaves the coverage area of the serving fog node during the execution of the request.

Each FN_i at time interval S_k serves a set of mobile users located in its coverage area C_i, represented by $H_{(i,s_k)}=\{U_{(f1)},...,U_{(fk)}\}$ where $\{f_1,f_2,\ldots,f_k\} \in \{1,2,...,M\}$ and $\forall i,j\in\{1,2,...,N\}: H_{(i,s_k)}\cap H_{(j,s_k)}=\emptyset$. In other words, each U_j is served by only one fog node in time slot S_k. When the mobile node U_j sends the request $A_{(id,\,U_j,t_k)}$, it is initially assigned to a nearby fog node at its access point $FN_i: U_j \in H_{(i,t_k)}$, hereinafter referred to as the primary fog node. The notation used in this paper is presented in Table II.

Using this system model, we aim to determine the suitable node to process a user request. In the proposed approach, service requests, especially delay-sensitive ones, as much as possible are handled in the fog nodes. If the primary fog node (the first fog node that receives the request from the IoT device) does not have enough resources to fulfill the request or it had a high load of requests, the request is forwarded to the cloud. Moreover, if it is predicted that the location of the user submitting the request falls outside the coverage area of the serving fog node, the request will be forwarded to the neighboring fog nodes in the same domain.

4. The proposed method

In this section, at first the proposed approach for managing the mobility of users is presented. Then, the proposed algorithm for task placement in fog computing is described in detail.

4.1. The proposed approach to manage the mobility of users

Mobile devices can change their locations dynamically. To forecast the future location of a mobile device, the two characteristics of direction and velocity are considered.

According to Fig. 2, in the mobile device, the Mobility Manager module updates the users' direction, velocity, and geographical location periodically over the specified time intervals S_k. In our discrete-time model, the set of mobile users $H_{(i,s_k)}$ served by FN_i remains unchanged during time interval S_k; while it may change across different time intervals.

Table II. Notations used in this paper.

Notation	Description
FD	Set of all available fog nodes in the fog domain
N	The number of fog nodes
$FN_i \ (\forall_i = 1,\ldots,N \quad FN_i \in FD)$	*i*th fog node
Q_i	FN_i's Queue
NL_i	List of neighboring fog nodes of FN_i
R_i	FN_i's coverage area
C_i	A square circumscribed by the FN_i's coverage area R_i
M	Total number of mobile users
$U=\{U_1,U_2,\ldots,U_M\}$	Set of mobile users
Vel_j	Velocity of the *j*th user
Dir_j	Movement direction of the *j*th user
Loc_j	Location of the *j*th user
Z	Total number of time slots for updating users' location
$S=\{s_1,s_2,\ldots,s_Z\}$	Set of all discrete time intervals for updating users' location
Δ_1	Length of time slot $S_k, k \in S = \{S_1,S_2,...,S_z\}$
w	Total number of discrete time intervals for performing the task placement algorithm
$T=\{t_1,t_2,\ldots,t_W\}$	Set of all discrete time intervals for performing the task placement algorithm
Δ_2	Length of time interval $t_k, k \in \{1,2,\ldots,W\}$
$A_{(id,\,U_j,t_k)}$	Application request with identifier *id* owned by the user U_j at time interval t_k.
$L_{A(id,\,U_j,t_k)}$	Number of instructions of $A_{(id,\,U_j,t_k)}$
$K_{A(id,\,U_j,t_k)}$	Class of $A_{(id,\,U_j,t_k)}$
$D_{A(id,\,U_j,t_k)}$	Deadline constraint of $A_{(id,\,U_j,t_k)}$
$W_{A(id,\,U_j,t_k)}$	Is $A_{(id,\,U_j,t_k)}$ from an IoT device or a neighboring fog node?

$H_{(i,s_k)} = \{U_{(f_1)},,\ldots,U_{(f_K)}\}$ The set of IoT devices in R_i at time interval s_k

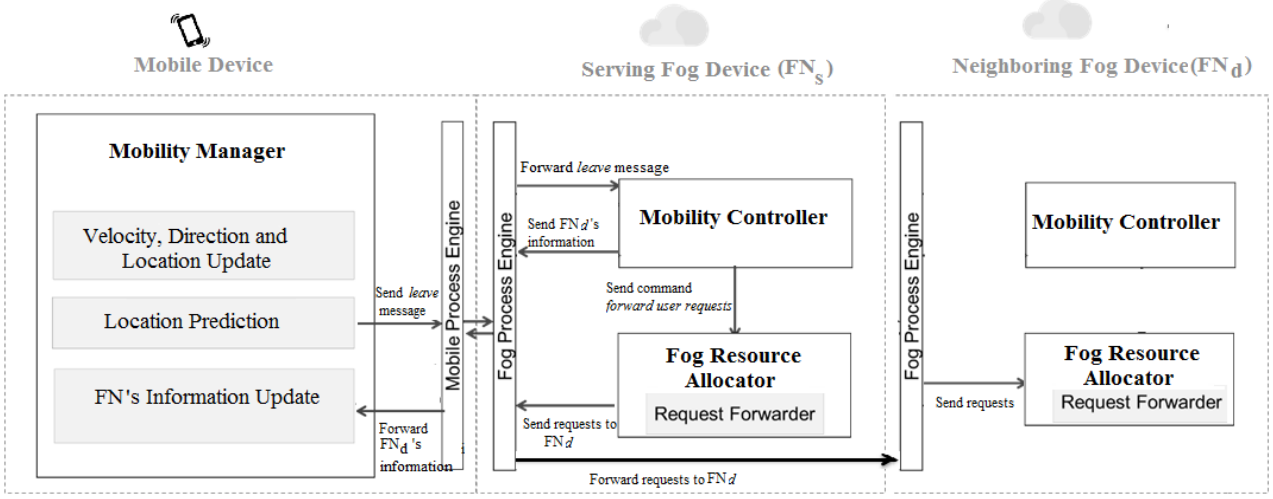

Fig. 2. The proposed approach for user mobility management

In Fig. 2, the Location Prediction function in the mobile device is responsible to forecast the user's location in the future time interval S_{k+1}. This prediction is performed by assuming that the direction and velocity of the user will not change during the current time interval S_k. This assumption often works when the length of the time intervals is considered small enough.

The first time a mobile node U_j requests a service, the primary fog node sends it information about its coverage area. Whenever the forecasted location of the U_j falls outside the sub-coverage area C_s of the serving fog node FN_s, the "leave" message is sent by the U_j to FN_s. After the "leave" message received by the Mobility Controller module in FN_s, it sends the information of the neighboring fog node that the mobile user is moving towards it, referred to as FN_d, to the U_j as a response. Moreover, the Request Forwarder in the Fog Resource Allocator module notifies FN_d of the arrival of U_j and also sends theU_j's non-executed requests to it. From now on, U_j sends its requests to the FN_d. The Process Engine Module in the mobile and fog devices plays the role of the interface and is responsible for sending requests and responding to other nodes.

4.2. The proposed algorithm for task placement

The proposed task placement algorithm, called *TICC*(Θ', $\Delta_1, \Delta_2, \gamma$), has two parts. The first part decides where the requests in the queue of a fog node should be scheduled so that deadline constraints are met, and the second part manages the user requests arriving from the neighboring fog nodes.

The first part of the proposed algorithm is run in each FN_i at the beginning of each time interval t_k with length Δ_2, to decide where to process the requests in Q_i (serving fog node, neighboring fog node, or cloud). This decision depends on some factors including request priority (application class), user mobility, and load of requests in the fog node.

In the proposed algorithm, the decision on where to serve requests is made based on criterion θ which is proportional to the current load of fog nodes. If FN_i is overloaded, some of its requests can be forwarded to the cloud. Fog node overloading refers to the situation in which a substantial number of requests in the queue are not serviced because of the high rate of incoming users into the fog node's coverage area compared to their leave rate. This situation results in the increase of waiting time of requests in the queue and deadline expiration of delay-sensitive requests. Criterion θ for FN_i at the time interval t_k is calculated based on the result of multiplying the entry-exit ratio of users $E_{(i,t_{k-1})}$ by the estimated time required to perform current requests that are waiting in the queue $TQ_{(i,t_k)}$.

$$\Theta_{(i,t_k)} = E_{(i,t_{k-1})} \times TQ_{(i,t_k)}. \quad (1)$$

Equation (2) is used to calculate the entry-exit ratio of users at time interval t_k inFN_i:

$$E_{(i,t_k)} = \begin{cases} \epsilon & O_{(i,t_k)} = 0 \; And \; I_{(i,t_k)} = 0 \\ I_{(i,t_k)} & O_{(i,t_k)} = 0 \; And \; I_{(i,t_k)} > 0 \\ \left(\frac{I_{(i,t_k)}}{O_{(i,t_k)}}\right) + \epsilon & otherwise \end{cases}. \quad (2)$$

where, $I_{(i,t_k)} = |H_{(i,s_j)} - H_{(i,s_{j\prime})}|$ is the number of users entering into the FN_i's sub-coverage area (C_i), while $O_{(i,t_k)} = |H_{(i,s_j)} - H_{(i,s_{j''})}|$ stands for users left out of C_i in the time interval t_k, $start(t_k) \in sj$, $start(t_{k-1}) \in sj'$ and $start(t_{k+1}) \in sj''$. The higher the entry-exit ratio, the greater the number of users in C_i, which in turn increases the likelihood of fog node overloading. ϵ represents a small positive value adopted to prevent $E_{(i,t_k)}$ from becoming zero. The value of zero for $E_{(i,t_{k-1})}$ ignores the effect of $TQ_{(i,t_k)}$ on calculation of $\Theta_{(i,t_k)}$.

Precise calculation of the execution time of an application request is a challenging issue and depends on many factors. We estimate the time needed to process the requests of Q_i in FN_i over the time interval t_k by Equation (3).

$$TQ_{(i,t_k)} = \frac{\sum_{\forall A(id,U_j,t_k) \; in \; Q_i} L_{A(id,U_j,t_k)}}{P_i} / 1000 \quad (3)$$

where, $L_{A(id,U_j,t_k)}(MI)$ is the number of instructions of request $A_{(id,U_j,t_k)}$ in Q_i, and P_i(MIPS) stands for the total processing capacity of FN_i. The risk of overloading increases with the increased time required to process users' requests. $TQ_{(i,t_k)}$ is in milliseconds.

If the value $\Theta_{(i,t_k)}$ obtained by Equation (1) is greater than the predetermined threshold Θ', the FN_i is expected to become overloaded in the next time interval (t_{k+1}); hence, the delay-tolerant requests in its queue would be sent to the cloud. The threshold value is determined according to the number of failed requests in each fog node as described in section 5.

Based on the results of overload prediction, placement and scheduling of requests in the fog node proceed in the following steps:

- Fog node: At any time interval t_k and in the absence of overload probability in the fog node, the node will handle requests itself with free processing capacity.
- Cloud node: If it is predicted that the fog node would be overloaded in the next time interval t_{k+1}, all delay-tolerant requests in the queue will be sent to the cloud. The purpose of this transfer is to reduce the time required for servicing delay-sensitive requests and to increase their acceptance rate.

The second part of the proposed algorithm is run in each FN_i at the beginning of each time slot S_k with time length Δ_1, to decide where to process the arrived requests from neighboring fog nodes (fog node or cloud). As stated, if it is predicted that a user will move from the serving fog node (FN_s)'s sub-coverage area (C_s) to a neighboring fog node (FN_d)'s sub-coverage area (C_d) within its domain, its requests in the FN_s's queue will be transferred to the FN_d. The FN_d can accept these requests or send them to the cloud. If the time demanded to process the requests of FN_d's queue ($TQ_{(d,t_k)}$) is bigger than the threshold γ, the delay-tolerant requests arrived from the neighboring fog nodes will be sent to the cloud and the delay-sensitive ones will remain in Q_d. The threshold value γ is determined according to the number of failed requests in each fog node, as described in section 5.

5. Simulation and results

In this section, we first present the simulation setup, and then, we evaluate the efficiency of the proposed method for task placement in fog computing.

5.1. Simulation setup

The ifogSim simulator was employed to simulate the proposed method for task placement [21]. The fog nodes were modeled according to the architecture mentioned in Fig. 1. A proxy server established the connection between the fog nodes and the cloud. Table III presents the configuration parameters of cloud, fog and mobile nodes including the processing capacity (MIPS), RAM (MB), uplink bandwidth, and downlink bandwidth.

Table III. Characteristics of the computational devices.

Device type	CPU length (MIPS)	RAM (GB)	Uplink bandwidth (MB)	Downlink bandwidth (MB)
Cloud	44800	40	100	10000
Proxy server	2800	4	10000	10000
Fog node	2800	4	10000	10000
Mobile device	1000	1	10000	270

Fog nodes were connected to a proxy (service function) through a network link with 4ms of latency [19]. The link between the gateway and the cloud had 100 *ms* of latency. Furthermore, the communication latency among fog nodes and between fog and mobile nodes were considered 20 and 2 *ms*, respectively.

We considered two types of applications: delay-sensitive and delay-tolerant. For the former, similar to [19], we used the electroencephalography (EEGPT) online game. The objective of EEGPT is to gather target objects by concentrating on them. To experience a true online game, fast processing and low response time of user requests are critical. This game consists of EER, client, concentration calculator, coordinator, and display modules. EER and display modules should be placed in mobile nodes while other modules are placed in both fog and cloud nodes [19]. For the delay-tolerant type, we considered a video surveillance/object tracking (VSOT) application which can tolerate datacenter distance latencies. The VSOT application has some distributed intelligent cameras that can track movement. This application has six modules as follows: camera, motion detector, object detector, object tracker, user interface, and zoom (PTZ) control. Camera and motion detector modules must be placed in mobile nodes, while user interface modules are placed in the cloud. Other modules can be placed both in fog and cloud nodes [19].

Real-time IoT applications are considered to work based on the Sense-Process-Actuate model [22]. Each sensor periodically transmits tasks at a specified rate. These tasks are forwarded to different modules of the IoT applications, and processed based on the dependency model among constituent modules. Each module receives tasks from predecessor modules as input and executes them, and produces respective tasks as its output to be forwarded to the next modules. The final results will be forwarded to the actuator as the last module.

To simulate user mobility, we assumed a geographical area of 1440*1440 m^2 (a segment of a city) for a fog domain that was divided into nine sub-areas $C_i: i \in \{1,2,...,9\}$ with the same size (Fig. 3). We created one fog node for each sub-area. As can be seen in Fig. 3, FN_i covers the area R_i of a radius of 340 *m*.

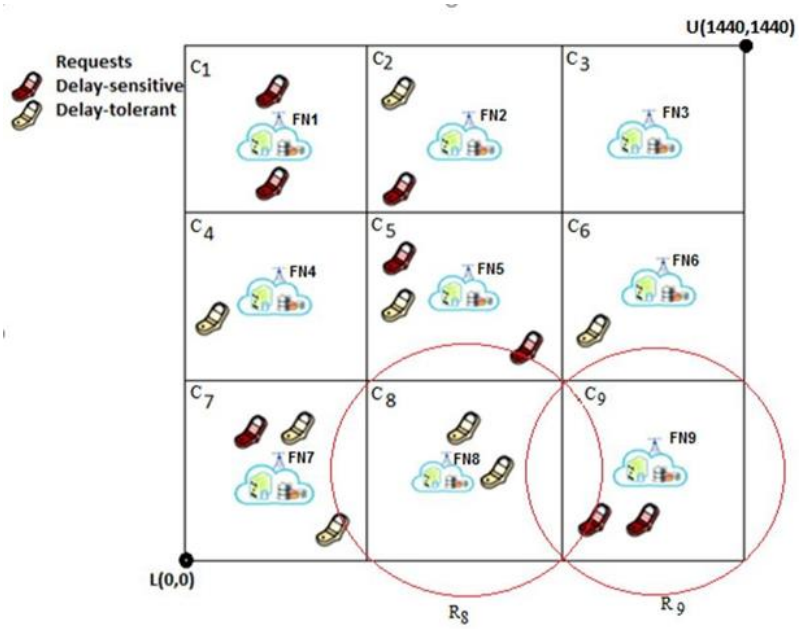

Fig. 3. Geographical area intended for the simulation.

For illustration purpose, the mentioned area can be assumed as a large department store, which includes several buildings. Users are allowed to navigate through various sections of the store that may cause a user to leave the coverage area of its serving fog node. Selection of a suitable fog node for the user that leaves the coverage area of its present fog node when several fog nodes are available is challenging. Hence, for reducing complexity, we assumed that the requests of the user located in the specific sub-area C_i were just served by the corresponding fog node of that area (FN_i). The two fog nodes FN_i and $\mathrm{FN_j}\ \forall \mathrm{i,j} \in \{1,2,...,N\}$ were considered as neighbors if C_i and C_j had a common border. For example, in Fig. 3, the neighboring fog nodes for FN_2 are FN_1, FN_3, and FN_5.

We also assumed that half of the users play EERPT with their smartphones and others run the VSOT application. During special hours of the day (e.g. lunch time), users move to a particular region (e.g. the restaurant) that may result in the overloading of the associate fog node.

At the beginning of the simulation process, six mobile nodes were connected to each fog node ($|H_{(i,s_1)}| = 6\ \forall i \in \{1,2,\dots,N\}, N = 9$). Mobile devices $H_{(i,s_1)}$ were randomly placed in the coverage area C_i of the corresponding fog node (FN_i). To update the location of a mobile node, the two characteristics of direction and velocity are considered. The direction of the mobile node can be one of the eight cardinal or ordinal directions. The user's velocity is also selected from [0-1.5] km/h. The velocity range was defined according to the average speed of pedestrians when walking into a department store. Initially, the velocity and direction of the mobile nodes were selected at random.

The users' movement direction was set towards the coverage area of a particular fog node (FN_5 in this paper) with predefined probability over the simulation process to create overloading at that node in order to facilitate the evaluation of the performance of the proposed method under the overloading situation.

Each mobile user submits 500 requests during the simulation process (20000 *ms*). As a result, a total number of 27,000 requests are submitted by the mobile nodes. It was assumed that each mobile node has a sensor and an actor. Each sensor generates a request and sends it to a higher layer, i.e. the mobile node. The time interval between submitting two consequent requests by a sensor is randomly selected from [10-15] *ms*. The user's location in the mobile node is updated every 1 second ($\Delta_1 = 1s$). Also, the length of the time interval to perform the task placement algorithm in each fog node is taken as 15 milliseconds ($\Delta_2 = 15\ ms$).

The execution deadline of delay-sensitive requests, i.e. the maximum amount of time for the difference between the time the request was sent in the sensor and the time the response was received in the actor, was randomly selected from [250-750] *ms* for each request. The simulation parameters are presented in Table IV.

Table IV. Simulation parameters.

Parameter	Value
N	9
M	54
$\#A_{(id,\ U_j,t_k)}$ for each U_j	500
Time interval of request submission	[10-15] *ms*
Δ_1	1000 *ms*
Δ_2	15 *ms*
$Vel_j\ j \in \{1,2,\dots,M\}$	[0-1.5] *km/h*
$Dir_j\ j \in \{1,2,\dots,M\}$	{north, east, south, west, northeast, southeast, southwest, northwest}
$Loc_j\ j \in \{1,2,\dots,M\}$	[0-1440]*[0-1440]m^2
$D_{A(id,\ U_j,t_k)}$	[250-750] *ms*
Z	20
W	80

5.2. Adjusting the threshold

As noted in section 4, the number of failed requests during the simulation was used to calculate the thresholds Θ' and γ. Fig. 4 shows the value of Θ and the number of failed requests in the fog nodes during a simulation run. To determine the Θ' and γ. thresholds, the values of Θ and $TQ_{(i,t_k)}$ were recorded when the number of failed requests in the fog node is between 100 and 150, respectively. Then, in each run the median of recorded values of Θ was calculated. Finally, the average of the medians was chosen as the threshold Θ' with a value of 5.4 after the simulation process was run 10 times. Also, the average median of recorded values of $TQ_{(i,t_k)}$ was considered as the threshold γ whose value equals 9.49. Also, the value of ϵ in Equation (2) is considered to be 0.001.

5.3. Evaluation of the proposed task placement method

The proposed method was compared with FCFS, Cloud-only, and the Delay-priority algorithms [19]. In the FCFS algorithm, requests in the fog node's queue are serviced sequentially without any priority. Depending on the processing power of a fog node (in this paper only CPU capacity is considered) at any given time interval, multiple requests can be executed simultaneously. In the traditional Cloud-only algorithm, all requests are sent to the cloud and no requests are processed in fog nodes. This algorithm is based on the traditional cloud-based implementation of applications. In the Delay-priority algorithm requests are executed like FCFS and in the absence of available capacity of fog nodes, delay-tolerant requests are scheduled in the cloud. In FCFS and Delay-priority algorithms, the closest fog node is selected to process the user request.

In the rest of this section, the results are averaged over 10 different runs. Table V presents the mean, best, worst, and standard deviation percentage of failed requests, as well as the percentage of delay-sensitive requests that failed for different algorithms during 10 simulated runs. As can be seen in Table V, $TICC(\Theta', \Delta_1, \Delta_2, \gamma)$ algorithm obtained better results than the other three methods. The reason for this is the transfer of delay-tolerant requests considering the entry-exit ratio of requests and the queue load in fog nodes which leads to the reduction of the number of requests in the fog node's queue, so the delay-sensitive requests are serviced in a shorter time. The role of predicting users leaving the current fog node areas and sending their non-executed requests to neighboring fog nodes cannot be ignored. This eliminates the need to fail requests due to the mobility of users so that as many requests would be executed as possible.

Table V. The mean, best, worst, and standard deviation of the results obtained by each algorithm during 10 runs.

Algorithm	Failed requests				Delay-sensitive failed requests			
	Mean	Best	Worst	Std	Mean	Best	Worst	Std
FCFS	65.57	9.31	75.88	19.87	90.09	95.06	84.16	3.4
Cloud-only	39.86	40	39.69	0.11	79.71	80.01	79.39	0.22
Delay-priority	23.72	26.85	18.87	2.42	45.27	51.59	35.84	4.7
TICC	12.07	13.7	8.09	1.67	23.93	27.01	15.98	3.32

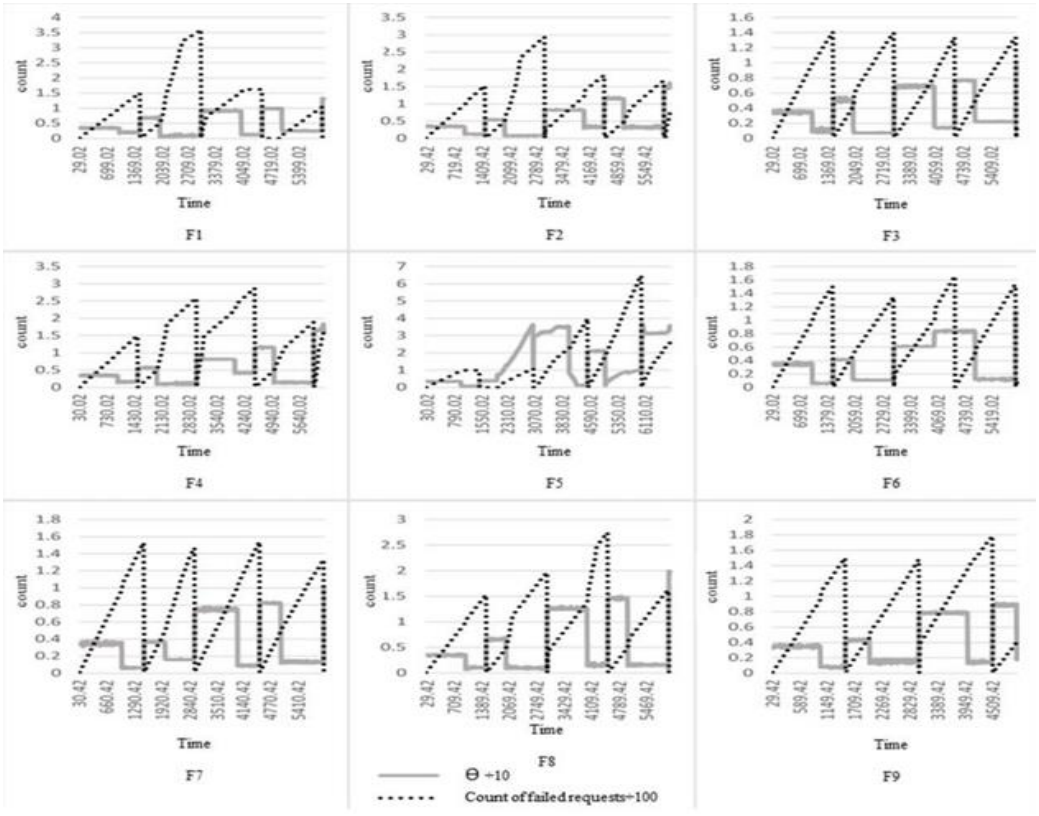

Fig. 4. Normalized values of Θ and the number of failed requests during the simulation in all fog nodes.

Fig. 5 shows the performance of the proposed approach for mobility management. As shown in Fig. 5, sending users' requests to the neighboring fog nodes (before they leave the current fog node's coverage area) increases the acceptance rate of requests and reduces the percentage of failed requests. Some of the requests that fail due to the mobility of users can be executed in neighboring nodes. Forwarding the requests of the mobile user at the boundary of two fog nodes before the user's leaving allows requests to be received and serviced faster at the destination fog node.

To evaluate the performance of the proposed method in the case of fog node overloading, the percentage of failed requests and delay-sensitive failed requests in the overloaded fog node (FN_5) are shown in Fig. 6. As mentioned earlier, for FN_5 to become overloaded during the simulation process, users move to that node with a 70% probability. The results in Fig. 6 indicate that in FN_5, fewer requests were failed due to $TICC(\Theta', \Delta_1, \Delta_2, \gamma)$ algorithm use. Also, according to this figure, a significant percentage of requests (23.17%) in the FCFS algorithm were failed in overloading conditions.

Table VI compares the average, best, worst and standard deviation for response time of requests in the proposed method with the other three algorithms. Response time refers to the time difference between the moment a sensor sends a request and the moment an actor receives the response. As can be seen, the average response time of delay-sensitive requests in $TICC(\Theta', \Delta_1, \Delta_2, \gamma)$ algorithm is shorter than other methods. Although $TICC(\Theta', \Delta_1, \Delta_2, \gamma)$ has a shorter average response time in comparison with other algorithms, this time is obtained for more number of requests. The difference between the lengths of delay-sensitive tasks and the lengths of delay-tolerant ones has caused the average response time of delay-sensitive requests to be higher than the average response time of all requests. As seen in Table VI, this difference is observed in all four evaluated algorithms.

Fig. 7 shows the percentage of failed requests in $TICC(\Theta', \Delta_1, \Delta_2, \gamma)$ algorithm compared to FCFS and Delay-priority algorithms when different values of 10, 15, 20, and 25 (ms) are selected for the time interval of execution of scheduling algorithms.

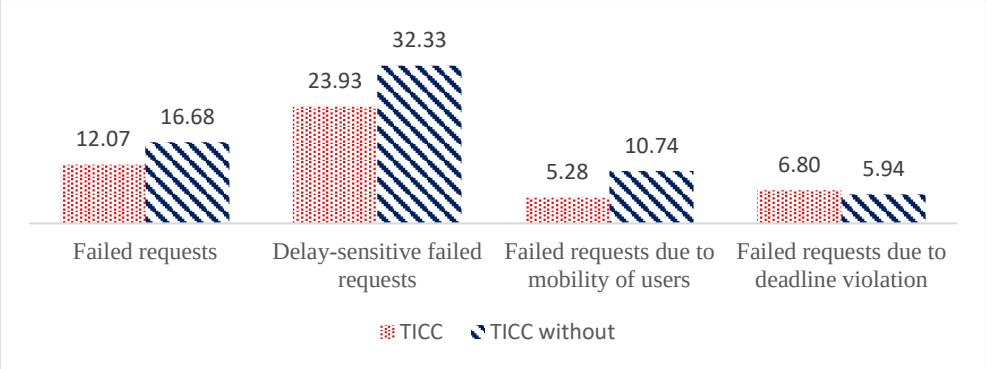

Fig. 5. Performance of the proposed approach for mobility management.

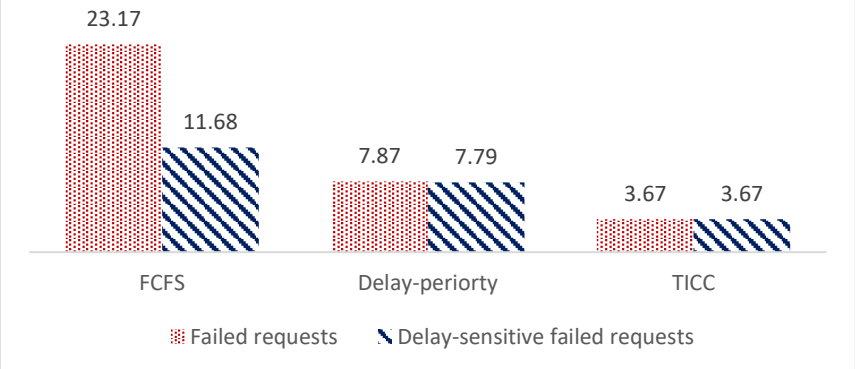

Fig. 6. Percentage of failed requests in fog node 5 using FCFS, Delay- priority and $TICC(\Theta', \Delta_1, \Delta_2, \gamma)$ algorithms.

Table VI. The mean, best, worst, and standard deviation of the response times obtained by each algorithm for 10 runs.

Algorithm	Failed requests				Delay-sensitive failed requests			
	Mean	Best	Worst	Std	Mean	Best	Worst	Std
FCFS	1080.02	976.5	1197.09	71.79	1495.86	13.79.79	1625.45	74.49
Cloud-only	900.05	894.68	904.06	2.78	901.45	896.96	905.69	0.11
Delay-priority	300.33	234.94	373.02	37.36	464.44	323.44	615.08	76.74
TICC	263.84	226.18	346.32	36.38	327.63	241.67	487.44	73.07

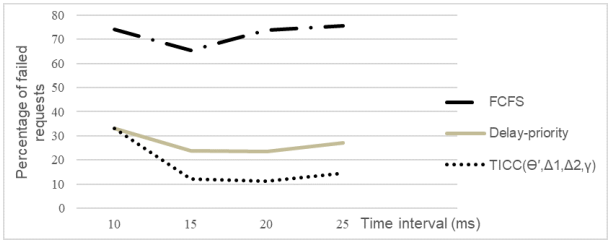

Fig. 7. Percentage of failed requests in FCFS, Delay-priority, and *TICC*($\Theta', \Delta_1, \Delta_2, \gamma$) algorithms for different time intervals.

According to Fig. 7, the lowest percentage of failed requests in all algorithms is when the time interval of 15 ms is selected. For this reason, as mentioned in section 5.1, the time interval to calculate $\Theta(\Delta_2)$ in each fog node was taken as 15 ms. In addition, according to Fig. 7, the proposed algorithm is performed better in comparison with FCFS and Delay-priority algorithms for different values of time intervals.

Finally, to determine the statistical differences between *TICC*($\Theta', \Delta_1, \Delta_2, \gamma$) and the compared algorithms, the Friedman test is conducted. To this end, we defined an objective function that multiplies the average percentage of failed requests by the average response time obtained through 10 runs for each algorithm. When the significance level (α) is set to 0.05, the results of the Friedman test are presented in Table VII. As seen in Table VII, *TICC*($\Theta', \Delta_1, \Delta_2, \gamma$) has the smallest mean ranking value which means it obtains the best overall rank. Also, the obtained p-value is smaller than the significance level. When the p-value is greater than the significance level, the difference is not statistically significant. However, since the p-value obtained by the Friedman test is less than 0.05, the compared algorithms are significantly different.

In summary, experimental results show that the proposed method for task placement in fog computing outperforms FCFS, Cloud-only, and Delay-priority algorithms in terms of acceptance rate and average response time of requests.

Table VII. The results of the Friedman test for ($\alpha = 0.05$).

Algorithm	Mean rank	p-value	Diff?
FCFS	3.90	2.33e-6	yes
Cloud-only	3.10		
Delay-priority	2.00		
TICC	1.00		

6. Conclusion

The issue of user mobility, along with the different latency requirements of different applications, necessitates the development of new methods of task placement that can provide the best possible performance and QoS to mobile users. This paper proposed an efficient method for task placement with respect to hierarchical architecture in fog computing. The proposed method supported user mobility and was able to handle different requests with different latency requirements. In the proposed method, the decision on where to process a request was done according to the mobility of users, prediction of overloading in the fog nodes, and the priority of requests. The experimental results showed that the proposed method outperforms the FCFS, Cloud-only, and Delay-priority algorithms in terms of acceptance rate and response time. Also, compared to the other three methods, the proposed method performed better in overload conditions. We aim to improve the proposed formula to overload prediction, use data mining techniques to identify users' mobility patterns, and employ different scheduling algorithms as future work.

7. References

[1] K. Ashton, "That internet of things thing", *RFID Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[2] M. Bahrami, M. Singhal, "The role of cloud computing architecture in big data", *Information granularity, big data, and computational intelligence*, pp. 275–295, 2015.

[3] A. Kumari, S. Tanwar, S. Tyagi, N. Kumar, "Fog computing for healthcare 4.0 environment: Opportunities and challenges", *Computers & Electrical Engineering*, vol. 72, pp. 1–13, 2018.

[4] M. Mukherjee, L. Shu, D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges", *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1826–1857, 2018.

[5] S. Mostafavi, F. Ahmadi, M. Agha Sarram, "Reinforcement-Learning-based Foresighted Task Scheduling in Cloud Computing", *Tabriz Journal of Electrical Engineering*, vol. 50, no. 1, pp. 387-401, 2020 (in persian).

[6] S. Ghasemi-Falavarjani, M.A. Nematbakhsh, B. Shahgholi Ghahfarokhi, "Multi-Objective Task Allocation in Offloading to Mobile Cloud", *Tabriz Journal of Electrical Engineering*, vol. 46, no. 4, pp. 217-232, 2017 (in persian).

[7] B. Nair, M.S.B. Somasundaram, "Overload prediction and avoidance for maintaining optimal working condition in a fog node", *Computers & Electrical Engineering*, vol. 77, pp. 147–162, 2019.

[8] K. Gasmi, K. Dilek, S. Tosun, S. Ozdemir, "A survey on computation offloading and service placement in fog computing-based IoT", *The Journal of Supercomputing*, vol. 78, no. 2, pp. 1983-2014, 2022.

[9] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, "Towards qos-aware fog service placement", In 1st international conference on Fog and Edge Computing (ICFEC), 2017, pp. 89-96.

[10] M.Q. Tran, D.T. Nguyen, V.A. Le, D.H. Nguyen, T.V. Pham, "Task placement on fog computing made efficient for IoT application provision", Wireless Communications and Mobile Computing, pp. 1-17, 2019.

[11] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, F. Desprez, "Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed IoT applications in the fog", In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 2018, pp. 751-760.

[12] F. Khosroabadi, F. Fotouhi-Ghazvini, H Fotouhi, "Scatter: Service placement in real-time fog-assisted iot networks", *Journal of Sensor and Actuator Networks*, vol. 10, no. 2, pp. 26, 2021.
[13] B.V. Natesha, R.M.R. Guddeti, "Meta-heuristic based hybrid service placement strategies for two-level fog computing architecture", *Journal of Network and Systems Management*, vol. 30, no. 3, pp. 47, 2022.
[14] M. Goudarzi, H. Wu, M. Palaniswami, R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments", *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298-1311, 2020.
[15] B. Kopras, B. Bossy, F. Idzikowski, P. Kryszkiewicz, H. Bogucka, "Task Allocation for Energy Optimization in Fog Computing Networks with Latency Constraints", *IEEE Transactions on Communications*, vol. 70, no. 12, pp. 8229-8243, 2022.
[16] I. Sarkar, M. Adhikari, N. Kumar, S. Kumar, "Dynamic task placement for deadline-aware IoT applications in federated fog networks", *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1469-1478, 2021.
[17] A. Mseddi, W. Jaafar, H. Elbiaze, W. Ajib, "Joint container placement and task provisioning in dynamic fog computing", *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10028-10040, 2019.
[18] D. Wang, Z. Liu, X. Wang, Y. Lan, "Mobility-aware task offloading and migration schemes in fog computing networks", *IEEE Access*, vol. 7, pp. 43356-43368, 2019.
[19] M. Peixoto, T. Genez, L.F. Bittencourt, "Hierarchical scheduling mechanisms in multi-level fog computing", *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2824-2837, 2021.
[20] S. Sarkar, S. Misra, "Theoretical modelling of fog computing: a green computing paradigm to support iot applications", *Iet Networks*, vol. 5, no. 2, pp. 23–29, 2016.
[21] H. Gupta, A.V. Dastjerdi, SK. Ghosh, R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments", *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.
[22] M. Goudarzi, M. Palaniswami, R Buyya, "A distributed application placement and migration management techniques for edge and fog computing environments", In 2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS), 2021, pp. 37–56.