Research Article

Fuzzy-Based Mobile Edge Orchestrators in Heterogeneous IoT Environments: An Online Workload Balancing Approach

Tran Trong Khanh, VanDung Nguyen , and Eui-Nam Huh

Department of Computer Science and Engineering, Kyung Hee University, Republic of Korea

Correspondence should be addressed to Eui-Nam Huh; johnhuh@khu.ac.kr

Received 17 February 2021; Revised 8 July 2021; Accepted 31 July 2021; Published 15 August 2021

Academic Editor: Marica Amadeo

Copyright © 2021 Tran Trong Khanh et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Online workload balancing guarantees that the incoming workloads are processed to the appropriate servers in real time without any knowledge of future resource requests. Currently, by matching the characteristics of incoming Internet of Things (IoT) applications to the current state of computing and networking resources, a mobile edge orchestrator (MEO) provides high-quality service while temporally and spatially changing the incoming workload. Moreover, a fuzzy-based MEO is used to handle the multicriteria decision-making process by considering multiple parameters within the same framework in order to make an offloading decision for an incoming task of an IoT application. In a fuzzy-based MEO, the fuzzy-based offloading strategy leads to unbalanced loads among edge servers. Therefore, the fuzzy-based MEO needs to scale its capacity when it comes to a large number of devices in order to avoid task failures and to reduce service times. In this paper, we investigate and propose an online workload balancing algorithm, which we call the fuzzy-based (FuB) algorithm, for a fuzzy-based MEO. By considering user configuration requirements, server geographic locations, and available resource capacities for achieving an online algorithm, our proposal allocates the proximate server for each incoming task in real time at the MEO. A simulation was conducted in augmented reality, healthcare, compute-intensive, and infotainment applications. Compared to two benchmark schemes that use the fuzzy logic approach for an MEO in IoT environments, the simulation results (using EdgeCloudSim) show that our proposal outperforms the existing algorithms in terms of service time, the number of failed tasks, and processing times when the system is overloaded.

1. Introduction

Nowadays, fifth-generation (5G) cellular technology has tremendously changed our daily lives by supporting various new applications, such as video streaming analysis, augmented reality (AR), the Internet of Things (IoT), and autonomous driving [1, 2]. In particular, the 5G-enabled IoT is a key major driver of the growth in IoT applications [3]. Nonetheless, in the era of the IoT, there is a large number of sensors, actuators, and mobile devices that generate a heavy load for networks [2, 4]. These applications require a very short response time and large amounts of computational resources. Moreover, 5G-enabled IoT applications require real-time responses and are dramatically increasing at the same time [5]. As a result, processing massive amounts of data traffic, and the growing demands for high data rates

and computational capabilities, are key features of the future Internet [6].

In conventional cloud computing, mobile devices upload all data to centralized servers to utilize the powerful computing capability of the cloud over a wide area network (WAN) [7, 8]. In heterogeneous IoT environments, cloud computing cannot ensure the requirements of new delay-sensitive applications. Moreover, cloud computing is deployed at some distance from IoT devices; therefore, it cannot support context-aware computing for IoT applications [9].

To address the challenges in the cloud for heterogeneous IoT environments, the edge computing paradigm brings servers closer to the IoT devices and is deployed at the edge of the network [10]. The mobile edge computing (MEC) provides latency-sensitive applications and can reduce traffic bottlenecks in the core and backhaul networks while assisting

in the offloading of heavy computational tasks from mobile devices to the edge [3, 11]. However, the heterogeneous IoT environment opens up potential innovations for efficient resource management that meets the quality requirements of different IoT application types, such as how the edge system responds to the dynamic flow of requests [12].

One such approach is known as the mobile edge orchestrator (MEO), as discussed by the European Telecommunications Standards Institute 1. It is designed to integrate end devices, edge servers, and the cloud to form a hierarchical IoT architecture [6]. We depict this system in Figure 1. The MEO can manage the available computing/storage/network resources and the life cycle of the application [13]. The MEO chooses the target server (i.e., local edge server/neighboring edge server/cloud server) for each requested task.

On the other hand, one of the main characteristics of a heterogeneous IoT environment is rapidly changing uncertain systems. Therefore, it is difficult to provide accurate mathematical models. Recently, machine learning techniques enable edge devices to maximize a predefined objective such as QoS and cache hit ratio (CHR) after they monitor their environment by learning and predicting various network features, such as channel dynamics, traffic patterns, and content requests [14]. However, the wireless edge networks have complex dynamics, such as channel variations, user mobility, user preference, and content popularity. The time complexity of solving optimization problems increases exponentially [14, 15]. Especially, when the frequency of user mobility is high, it leads to increased latency, higher energy consumption, and inefficient resource utilization [16].

Fuzzy logic has recently become a good alternative to controlling real-world imprecision. Because the fuzzy logic-based approach has lower computational complexity than other decision-making algorithms, it is significant for solving online and real-time problems. The decisions based on fuzzy logic are made considering edge and cloud computing characteristics [17]. Considering the parameter link delay and the signal-to-noise ratio, the fuzzy-based offloading ratio is calculated [18]. By using network information and matching it with the requirements received from applications, a fuzzy logic-based orchestrator can execute processes; so, the target mobile edge host can process applications [12]. In a fuzzy-based MEO, the input variables chosen have a significant effect on system task execution. Additionally, the role of a fuzzy-based MEO is to choose a target server, which can be a local edge server, a neighboring edge server, or a cloud server, based on the profile of an incoming application task and mobile edge computing characteristics. In [19], Nguyen et al. propose the collaborative task offloading scheme between mobile device-edge-cloud servers by adding the WLAN delay as a fuzzy input parameter. Meanwhile, Hosain et al. [20] use the neighboring MEC VM utilization as the fuzzy input parameter. However, the existing system leads to unbalanced loads among edge servers when the system is overloaded. Moreover, it also did not study how to find the best neighboring edge server where to offload the task is beneficial and scale its capacity when it comes to a large number of devices in order to avoid task failures and to reduce service time.

In this paper, we investigate the MEO system and the fuzzy-based MEO using different online workload balancing algorithms. We propose a new algorithm, achieved by considering user configuration requirements, server geographic locations, and available resource capacities. Our proposal allocates the proximate server when the user requests arrive at the MEO in real time. The key contributions of this paper are as follows.

- (1) We investigate and compare fuzzy-based MEOs using different online workload balancing algorithms. Simulation results suggest that the fuzzy-based MEO should use a worst fit online workload balancing algorithm
- (2) We propose a new approach to not only find the proximate server but also to balance the resource capacities of edge servers. By keeping track of server geographic locations and available resource capacities, our proposal can find the proximate server when the user requests arrive at the MEO in real time
- (3) Simulation results demonstrate the following. (1) Our proposal has better performance results, compared with the baseline online workload balancing algorithms, in terms of failed task ratio, average service times, and processing times for augmented reality and healthcare applications. (2) Compared to the existing algorithms, when the system load is not high, our proposal has results approximately the same as the fuzzy approach [12] and higher than competitor approach [17]. When the system is overloaded, our proposal outperforms both of these algorithms in terms of the failed task ratio
- (4) Our proposal allows the huge number of requested tasks from IoT applications that are processed when the system is overloaded. By using our proposal, the number of failed tasks is reduced. Moreover, our proposal can process more AR and healthcare application tasks than competitors

The remainder of this paper is organized as follows. Section 2 reviews the related works. Section 3 describes the online workload balancing algorithm. Section 4 provides a performance evaluation of the simulation results, followed by conclusions and proposed future research in Section 5.

2. Related Works

The mobile edge orchestrator, which is the central function in the MEC system, maintains all information on MEC, such as mobile edge geographic locations, available resource capacities, and the profiles of the applications creating the demand [21]. Additionally, the MEO is basically used to manage and control this information to make the offloading decisions for incoming tasks [22]. The MEO uses the requirements received from the application task for the decision on where to offload the process and on which target server. The basic decision method is made by matching the resource

FIGURE 1: MEC and the role of the edge orchestrator.

information with the application requirements so the MEO can select the target ME host [23].

Two major actions in the MEO are to monitor uncertain systems and determine the target virtual machines (VMs) for offloading the requested tasks. Many approaches have been proposed for solving the workload orchestration problem. The decisions made can be based on various conditions, such as a combination of network latency, data transfer size, and remote server capabilities [24]. Based on the WLAN, the metropolitan area network (MAN), WAN communications, and the requirements of the application task, the decision can be made based on the edge/fog computing infrastructure [25]. However, these approaches do not take network congestion into account.

Baktir et al. [26] proposed a server centric method by using the orchestration of software-defined networking on the edge server. In an edge computing system, Hegyi et al. [23] optimized placing the components of IoT applications, and Karagiannis and Papageorgiou [27] proposed placement based on the partitioned shortest path method to solve the placement optimization problem. In fog computing infrastructures, Santoro et al. [28] proposed efficient and optimized use of the infrastructure while satisfying the application requirements, naming the approach Foggy. On the other hand, Bittencourt et al. [25] proposed an edge-

ward placement algorithm where modules of the same application are grouped to be placed in the same device. This algorithm was designed according to three different policies: (1) delay-sensitive versus delay-tolerant, (2) resource management, and (3) allocation. The results in this paper conclude that scheduling strategies can be designed to cope with different incoming application classes to take advantage of both fog computing and cloud computing characteristics [25].

To apply a fuzzy logic-based approach to an MEO system, Sonmez et al. [12] proposed an automated management system to solve online problems. The script input parameters under consideration were WAN bandwidth, the length of the incoming tasks, VM utilization on the edge server, the delay sensitivity of the related tasks, MAN delay, local edge VM utilization, and remote edge VM utilization. The offloading decisions made consisted of local, neighboring edge, and cloud servers. In our work, we consider online balancing algorithm to improve the better service delay performance for IoT application when the system is overloaded.

To improve the system performance, online workload balancing algorithms [29, 30] operate in coordination with the MEO to ensure the workloads are distributed to the proximate server. A classic problem in the context of online computation is known as bin packing. Li et al. [31] concluded that certain bin-packing strategies were considered for renting

TABLE 1: Five baseline online workload balancing algorithms.

Name	Operation	Advantages	Disadvantages
<i>First fit</i> [49]	Allocates VMs in the order of requested tasks from applications sent from a mobile user to the MEO.	The fastest algorithm.	The remaining unused memory areas are wasted if it is smaller than the required resource capacity of the requested tasks. Consequently, a larger resource capacity for the application is not accomplished.
<i>Best fit</i> [41]	Allocates the VM that has the lowest resource capacity.	Memory utilization is better than <i>first fit</i> .	(i) Tends to full up a tiny resource capacity (ii) Shows several variances in load across the machine [38]
<i>Worst fit</i> [42, 43]	Assigns a VM that has the highest resource capacity and continues in a similar manner.	Reduces the waste of resource capacities in the overall system network.	When the system is overloaded, a larger resource capacity for the application cannot be accommodated because the highest resource capacity is already occupied.
<i>Next fit</i> [50]	Similar to first fit, but a pointer parameter in the next fit method points to the next VM to be allocated.	The same competitive ratio for each VM.	When the system is overloaded, the performance is poor [34].
<i>Random fit</i> [39, 40]	Allocates a random VM.	Distributes the requested tasks evenly based on resource capacities.	(i) Cannot provide for different computational complexities as a requirement of the applications (ii) Poor make span when multiple objectives are considered simultaneously in a cloud environment [38]

servers in the cloud. Some approaches have been studied for connecting the bin-packing problem to the centralized scheduling problem [29, 30, 32–37].

Sambit et al. [38] proposed a taxonomy for load balancing algorithms in the cloud. In online mode, some approaches were proposed, such as opportunistic load balancing (OLB) [39, 40], minimum execution time (MET) [41], minimum compilation time (MCT) [42], and simulated annealing (SA). In OLB [39, 40], the requested tasks are arbitrarily allocated to virtual machines based on the execution time of each task. On the other hand, based on the lowest execution time, MET [41] allocates each task to a VM. Considering both ready-to-execute time and the expected execution time of the tasks, MCT [42, 43] allocates tasks based on the shortest completion time. Based on a probability distribution, Moschakis et al. [44] proposed to model the workloads of IoT applications in order to maximize the performance of scheduling such applications. In [45], to formalize the load balancing, the authors used standard deviation which can measure the dispersion of a set of values. The lower standard deviation indicates that the values are closer to each other. In this paper, the authors use nondominated sorting genetic algorithm III (NSGA-III) to search for a set of edge server (ES) placement with low latency and a balanced workload.

Based on these approaches, we summarize five basic online workload balancing algorithms: *first fit*, *best fit*, *random fit*, *worst fit*, and *next fit*. The operations, advantages, and disadvantages of each algorithm are discussed in Table 1. On the other hand, the fuzzy logic-based approach is significant for solving online, and real-time problems because it based on fuzzy logic are made considering edge and cloud computing characteristics [46]. Additionally, the role of a fuzzy-based MEO is to find a target server based on the profile of an incoming application task and mobile edge computing characteristics [19]. By combining the fuzzy logic-based approach and our proposed workload balancing

algorithm, our proposal gets better performance results when supporting fuzzy-based MEO in IoT environments, as discussed in the next section. We will summarize our contributions between this paper and the existing Fuzzy-based MEO approaches in Table 2.

3. Online Workload Balancing Algorithm for a Mobile Edge Orchestrator

In the MEO, fuzzy logic is suitable for addressing technical challenges. One of important challenges is known as changing uncertain systems, i.e., CPU utilization on a virtual machine (VM) and the bandwidth fluctuation. We will explain why fuzzy logic is an excellent alternative for handling real-world imprecision in MEO.

- (1) Based on well-understood principles and the use of imprecise information provided in the form of human language, it can handle uncertainty in predictable environments [47]
- (2) Fuzzy logic-based approaches have lower computational complexity than other decision-making algorithms [48]

Moreover, under rapid changes in the various workloads, the existing decision-making algorithms based on multicriteria optimization with a mathematical model need to know details about resource utilization by the server and details on the network's condition. In contrast, fuzzy logic allows for the consideration of multiple parameters in the same framework.

3.1. Fuzzy-Based Mobile Edge Orchestrator Preliminaries. The main objective of the fuzzy-based mobile edge orchestrator is to handle the multicriteria decision-making process to obtain crisp output by considering multiple parameters in

TABLE 2: Comparison with the existing fuzzy-based MEO approaches.

K.P.I	Our paper	Nguyen et al. [19]	Hossain et al. [20]	Sonmez et al. [12]
Object	Task offloading decision and load balancing scheme	Task offloading decision	Task offloading decision	Task offloading decision
Input parameters	Basic fuzzy input and simple optimal problem	Basic fuzzy input and (WLAN) delay parameters	Basic fuzzy input and MEC VM utilizations parameters	Basic fuzzy input: WAN bandwidth, task length, average VM utilization on the edge server, delay sensitivity of task, MAN delay, local edge VM utilization, remote edge VM utilization
Target server	(i) Local edge server (ii) Best neighboring edge server (iii) Cloud server	(i) Mobile device server (ii) Edge server (local and neighbor) (iii) Cloud server	(i) Local edge server (ii) Neighboring edge server (iii) Cloud server	(i) Local edge server (ii) Neighboring edge server (iii) Cloud server
Load balancing scheme	Simple optimal problem	(i) Worst fit	(i) Not available	(i) Not available
Advantages	(i) When the system is overloaded, it provides not only the lowest failed task ratio but also the shortest processing time for healthcare and AR application	(i) It solves the bottlenecks of the multi-tier edge computing architectures because of the essential factors: WLAN delay, MAN delay, local and neighboring VM utilizations	(i) It can accommodate more computation workload in the MEC system and reduce reliance on the remote cloud	(i) It outperforms in terms of resource utilization and response time
Disadvantages	(i) Not consider task migration strategy	(i) Unbalanced loads among edge servers (ii) Not consider task migration strategy	(i) Unbalanced loads among edge servers (ii) Not consider task migration strategy	(i) Unbalanced loads among edge servers (ii) Not consider task migration strategy

the same framework. First, we explain the global fuzzy logic system. The fuzzy logic system (FLS) consists of four main components: the fuzzifier, the rules, the inference engine, and the centroid defuzzifier, as shown in Figure 2. The fuzzification step is used to transfer crisp input to the fuzzy input set by using membership functions. The defuzzification step is used to again transfer from the fuzzy output set to crisp output, a.k.a., the offloading decision. We summarize the operation of the FLS as follows [12, 17, 18, 51, 52].

- (1) Fuzzification is used to transform crisp input sets to fuzzy sets. Note that a crisp set is converted to a linguistic variable (LV) for each indicator. The LV is decomposed into linguistic terms (LTs). We use a membership function to quantify an LT
- (2) Fuzzy input sets are introduced to the inference engine used to evaluate and combine the fuzzy rules in order to make an inference
- (3) The resulting fuzzy output, called a crisp output value, is processed in the defuzzification step by using a centroid defuzzifier method

For example, we consider a fuzzy-based mobile edge orchestrator as being used to make offloading decisions for incoming tasks. Two offloading decisions are designed based on the crisp output value. To do that, the system will determine the input variables that have a significant

FIGURE 2: Fuzzy logic system.

effect on performance results. These variables will be sent to the fuzzy-based mobile edge orchestrator, and they play a decisive role in the task-offloading decision process. According to the profile of the application creating the demand, we take the MAN delay, local edge VM utilization, and remote edge VM utilization to provide formal input variables. We describe the role of these crisp input variables as follows.

- (1) MAN delay (denoted by a): If the MAN resources are congested, the local edge server is chosen [12]
- (2) Local edge VM utilization (denoted by b): Many tasks fail because of a limited resource capacity on the local edge server; therefore, local edge VM utilization has to be considered

FIGURE 3: The shapes of FLS membership functions.

TABLE 3: Example fuzzy rules found empirically for a fuzzy-based mobile edge orchestrator.

Rule index	a	b	c	Decision
R1	Low	Low	Light	Local device
R2	Low	High	Normal	Local device
R3	High	High	Normal	Remote edge
R4	Medium	Medium	Heavy	Local edge
R5	High	High	Heavy	Remote edge

- (3) Remote edge VM utilization (denoted by c): If the neighboring edge CPU utilization is lower than the local edge server, the task should be offloaded to the neighboring edge sever

Crisp input variables: Based on the characteristics of the requested task, we define three of the abovementioned input variables for the fuzzy-based mobile edge orchestrator, as shown below:

$$F = \{a, b, c\}. \quad (1)$$

Linguistic variables: The linguistic variables of $\{a, b\}$ are described as low, medium, and high, abbreviated as L, M, and H, respectively. In addition, L (light), M (normal), and H (heavy) are used for linguistic variables $\{c\}$.

Membership function: In our model, we use the most commonly used triangular form representing the membership function. The values used in membership functions for $\{a, b, c\}$ were tested by Sonmez et al. [12]. The shape of the membership functions for all crisp input variables is depicted in Figure 3.

We associate a grade with each linguistic term, and the crisp value is transformed into a fuzzy value in the fuzzification step by using the following membership functions:

$$F_a(x) = [\mu_a^L(w), \mu_a^M(x), \mu_a^H(x)]. \quad (2)$$

$$F_b(x) = [\mu_b^L(w), \mu_b^M(x), \mu_b^H(x)], \quad (3)$$

$$F_c(x) = [\mu_c^L(w), \mu_c^M(x), \mu_c^H(x)].$$

Fuzzy rules: A simple IF-AND-THEN rule, a.k.a., hypotheses followed by a conclusion, is used to conduct the fuzzy rule set [53]. To determine the fuzzy rules, we vary and find empirically fuzzy rule sets. Among the different fuzzy rule sets, we chose the best rule combination in the computational experiments [12]. The number of fuzzy rules is $n = 3^4 = 81$ based on four membership functions with three linguistic terms. Table 3 shows example fuzzy rules found empirically for the placement problem.

In the aggregation step, we use minimum ($\min$) and maximum ($\max$) functions based on the rule set chosen in the fuzzy rule step. We calculate a fuzzy value for selecting the mobile device, the local edge, and the candidate edge server as follows:

$$\mu_{\text{mobiledevice}} = \max \{ \mu_{\text{mobiledevice}}^{R1}, \dots, \mu_{\text{mobiledevice}}^{Rn} \}. \quad (4)$$

$$\mu_{\text{localedge}} = \max \{ \mu_{\text{localedge}}^{R1}, \dots, \mu_{\text{localedge}}^{Rn} \}, \quad (5)$$

where n is 81, and the $\min$ functions are based on the fuzzy rule step. For example, in Table 3, the $\min$ functions are given as

$$\begin{aligned} \mu_{\text{localedge}}^{R1} &= \min \{ \mu_a^{R1}(m), \mu_b^{R1}(n), \mu_c^{R1}(l), \mu_d^{R1}(k) \}, \\ \mu_{\text{remoteedge}}^{R3} &= \min \{ \mu_a^{R3}(m), \mu_b^{R3}(n), \mu_c^{R3}(l), \mu_d^{R3}(k) \}, \end{aligned} \quad (6)$$

where m, n, l are the crisp input parameters for the fuzzy inference system.

Defuzzification: In the defuzzification step, according to Mendel [53], the centroid defuzzifier method, which is used to calculate the inference, achieves the center of gravity (COG) of the area under the curve, as shown in Figure 4. It is calculated as

$$\omega = \frac{\int_{x \in X} x \mu_i(x)}{\int_{x \in X} \mu_i(x)}, \quad (7)$$

where i is a local edge and a remote edge. After applying the centroid defuzzifier, a crisp output value, ω , is in the range [

FIGURE 4: The centroid for the defuzzification process.

FIGURE 5: Two stage fuzzy-based mobile edge orchestrator using the online workload balancing algorithm.

0,100]. Based on ω , we define the offloading decision as follows:

$$decision = \begin{cases} \text{Local edge server} & \omega_1 < 50 \\ \text{Remote edge server} & \text{Otherwise.} \end{cases} \quad (8)$$

Similar to the fuzzy-based mobile edge orchestrator [12], our system also has a two-stage FLS to find the target server: a local edge server, a neighboring edge server, or a cloud server. To find the best neighboring edge server to improve the number of requested tasks that are processed successfully, we add a function named the online workload balancing algorithm, as shown in Figure 5. The first stage was discussed above according to three input variables: MAN delay, local edge VM utilization, and remote edge VM utilization. Consequently, two offloading decisions are made for a local edge server and a remote edge server. The second-stage operation is on four input variables: WAN bandwidth, the length of the

incoming task, VM utilization on the edge server, and delay sensitivity of the incoming task [12]. These input variables are defined as follows:

- (1) *WAN bandwidth*: If the WAN communication delay is higher than the application QoS requirement, the offloading decision cannot be to a cloud server [12]
- (2) *Task length*: The execution time is calculated based on the task length. A complicated task should be offloaded to a cloud server to reduce execution time
- (3) *Average VM utilization*: This provides the remaining computational capacity. Congestion is defined as an average edge VM utilization that is greater than a set threshold level [12]
- (4) *Delay sensitivity task*: We determine the tolerance of the task if it takes a long time due to network congestion or server utilization levels

FIGURE 6: The role of online workload balancing algorithm in a fuzzy-based MEO architecture.

Result: The best neighboring edge server, O_i .

Initialization;

L_j : List of neighboring edge servers to edge server j ;

i : Mobile device uploading an application task to connecting edge server j ;

k : Edge server close to edge server j ;

(x_j, y_j) : The geographic location of edge server j ;

(x_k, y_k) : The geographic location of edge server k ;

for $k \in L_j$ **do**

 Find k satisfy constraint (13c);

 Find VM resource capacity on edge server k to satisfy constraint (13c);

if There is available resource capacity **then**

$O_i = \text{cloud}$;

break;

end

end

if There is no available resource capacity **then**

$O_i = \text{cloud}$;

else

$L_j = L_j \setminus \{k\}$;

end

ALGORITHM 1: Online workload balancing algorithm.

The decisions in the second stage are about the neighboring edge server and the cloud server. The mobile user will send the profile of the requested task, including its character-

istics (such as the length of the task, the delay sensitivity, and the required computation resources). By using network information (i.e., VM utilization on the edge server, WAN

TABLE 4: Simulation parameters [12].

Parameters	Value
Simulation time; warm-up period	33 min/3 min
WAN/WLAN bandwidth	Empirical
MAN bandwidth	MMPP/M/1 model
LAN propagation delay	5 ms
Number of VMs per edge/cloud server	8/4
Number of cores per edge/cloud VM CPU	2/4 minutes
VM CPU speed per edge/cloud	10/100 GIPS
Mobility model	Random way point
Propagation of selecting a location type	Equal
Number of locations, type 1/2/3	2/4/8
Mean dwell time in type 1/2/3	2/5/8 ms

bandwidth, and MAN delay) and matching it with the requirements received from the application, the orchestrator determines the target mobile edge host to process the application.

3.2. Online Workload Balancing Algorithm. Under IoT systems, it is necessary to design an edge computing system to handle a dynamic flow of requests that can be efficiently processed. Therefore, the workload orchestration problem must be studied based on both computational and networking resources [12]. Moreover, the MEO needs to scale its capacity with respect to a large number of devices in order to avoid task failure and to reduce service time. To cope with this issue, we propose the online workload balancing algorithm used in a fuzzy-based MEO architecture as shown in Figure 6 in order to improve system performance. The goal of our algorithm is to ensure the workloads distributed to the appropriate servers when the mobile user requests arrive at MEO in real-time scenario.

3.2.1. Transmission Delay at the Edge Servers. According to the standard practice in computing offloading modeling [5, 54–56], the time it takes to transmit the requested task of an application from local edge server j connecting to mobile user i to neighboring edge server k depends on the size of the input data that the task is associated with, b_{ij} , which is given as

$$t_{jk}^i = \frac{b_{ij}}{R_{\text{MAN}}} + t_{jk}^{\text{MAN}}, \quad (9)$$

where R_{MAN} is the rate of transmission in the MAN. The MAN propagation delay is calculated as

$$t_{jk}^{\text{MAN}} = \frac{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{s_{\text{MAN}}}, \quad (10)$$

where s_{MAN} is the wave propagation speed.

3.2.2. Processing Time. In accordance with the usual representation of processing time in the task allocation problem [5, 54], the time needed for a task from mobile user i at edge server j based on the number of CPU cycles required is given as

$$t_{jk} = \frac{\beta_{i,j}}{f_k^e}, \quad (11)$$

where the k^{th} edge node allocates the i^{th} device at f_k^e giga instructions per second (GIPS). $\beta_{i,j}$ is the number of CPU cycles required to compute one-bit data for this task [57].

3.2.3. The Best Neighboring Edge Server Decision Mechanism. In our model, the MEO will determine whether the tasks will be executed on the edge server or the cloud server based on the abovementioned offloading decision. However, to minimize latency for each task, and to balance the edge servers, the best neighboring edge server is chosen. First, the total delay of a task from the i^{th} device served is

$$T_{j,k} = t_{jk}^i + t_{j,k}. \quad (12)$$

We aim at minimizing the delay of each task and attaining a balance between the resource capacities of edge servers that are managed by the MEO. The optimization problem can be formulated as follows:

$$\min_k T_{j,i}, \quad (13a)$$

$$s.t., \quad (13b)$$

$$\min t_{jk}^{\text{MAN}}, \quad k \in L_j, \quad (13c)$$

$$\beta_{j,i} \leq B_k, \quad k \in L_j, \quad (13d)$$

where L_j is a list of edge servers neighboring edge server j that are not used at the current time. The solution of the optimized problem in (15) can easily be found as follows. First, based on constraint (17), the MEO will look up the neighboring edge server to edge server j that is close to j based on geographic location. Second, the MEO will check the available resource capacity of each VM in edge server k . If constraint (18) can be satisfied, the task requested by the application will be processed by edge server k . If not, the MEO will choose the next-best neighboring edge server to edge server j based on geographic location. If there is no available resource capacity to process the task requested by the application, the task will be uploaded to the cloud server.

The algorithm takes the user's specified constraints into consideration, including server geographic location and the available resource capacity. The details of the online workload balancing algorithm are shown in Algorithm 1.

TABLE 5: Application types used [12].

	AR	Healthcare	Compute-intensive	Infotainment
Usage percentage (%)	30	20	20	30
Task interval (sec)	2	3	20	7
Delay sensitivity (%)	0.9	0.7	0.1	0.3
Active/idle period (sec)	40/20	45/90	60/120	30/45
Upload/download data (kb)	1500/25	20/1250	2500/200	25/1000
Task length (GI)	9	3	45	15
VM utilization on edge (%)	6	2	30	10
VM utilization on cloud (%)	0.6	0.2	3	1

4. Performance Evaluation

To evaluate our proposal, we used a realistic model known as the multitier edge computing infrastructure, named Edge-CloudSim [58]. The simulation parameters are presented in Table 4. According to [12, 58], the CPU speed of VMs per edge and cloud servers is 10 GIPS and 100GIPS, and the edge and cloud servers operate 8 VMs and 4 VMs when the system can handle over 500 mobile devices, respectively. Moreover, the characteristics of the potential applications vary according to the requirements. In this paper, we consider different incoming tasks from four applications: augmented reality, healthcare, compute-intensive, and infotainment applications. They have different profiles in terms of task arrival distribution, delay tolerance, and task size, as shown in Table 5. The interarrival times and task sizes are exponentially distributed random variables [12].

According to [12, 58], to simulate more realistically, we use four different application types in our study. First, an augmented reality application on Google Glass is presented in [59]. Second, the infotainment application is discussed in [47]. Third, a health application that uses a foot-mounted inertial sensor to analyze the walking pattern of the users is studied in [60]. Finally, the compute intensive application is discussed in [12]. All applications are set up in EdgeCloudSim simulator [58].

In our simulations, we assume that the mobile devices offload tasks that belong to a predefined set of application categories. The user wearing the smart glass offloads captured pictures to the remote servers, which provide face recognition service. For some time, it collects data and sends it to the edge services. The user with the foot-mounted inertial sensor offloads sensor data to the remote servers which provide a fall risk detection service. The AR application requires high CPU computation, a small amount of data to download, and a bigger amount of data to upload [58]. Meanwhile, the infotainment application requires less CPU power compared to the augmented reality application. To process these tasks, infotainment and compute-intensive applications send their tasks to the remote servers, which provide related services.

In the performance evaluation, we discuss three main results from different perspectives. We demonstrate the following. (1) The online balancing algorithms have different effects according to each method. We suggest that *random*

fit, *worst fit*, and *next fit* balancing algorithms have better performance in terms of average failed task ratio, as shown in Figure 7. (2) We compare our proposed fuzzy-based *FuB* algorithm with five basic methods in terms of average failed task ratio, service time, and processing time, as shown in Figure 8. (3) We compare our proposal with the existing methods, as shown in Figure 9. Simulation results show that our proposal outperforms in terms of average failed task ratio, service time, processing time, and VM utilization when the system is overloaded.

4.1. A Comparison between Nonbalancing and Balancing Algorithms Used for a Fuzzy-Based MEO. To emphasize why the fuzzy-based MEO uses a balancing algorithm, we compared it with nonbalancing algorithms in the same environments. We considered an important metric in system performance: the failed task ratio. This metric is calculated by computing the ratio between the number of failed tasks and the total number of tasks. Figure 7 shows the failed task ratio for *First fit* [49], *Best fit* [41], *Worst fit* [42, 43], *Next fit* [50], and *Random fit* [39, 40] algorithms. When the system load is under 600 mobile users, the results of *best fit* and *first fit* balancing algorithms are worse than nonbalancing algorithms, whereas *random fit*, *worst fit* and *next fit* have the same results. All balancing algorithms were better than nonbalancing algorithms when the number of mobile users was higher than 600. We classified balancing algorithms into two groups according to performance results: group A (*best fit* and *first fit*) and group B (*random fit*, *worst fit* and *next fit*). Moreover, group B showed performance results that were better than group A, as shown in Figure 7. Therefore, we suggest that group B has to be considered for the fuzzy-based MEO.

4.2. A Comparison of Fuzzy-Based MEO Using Different Workload Balancing Algorithms and Our Proposal. In this section, we compare our proposed *FuB* algorithm and the two abovementioned groups in terms of average failed tasks, service times, and processing times based on all application types and for different application types, as shown in Figures 8, 10, and 11.

First, based on all application types, Figure 8 shows the average failed tasks, service times, and processing times. Our proposed *FuB* algorithm has a higher number of successfully processed tasks than groups A and B, as shown in

FIGURE 7: Average failed task ratio in the same environments.

FIGURE 8: Average failure task, service times, and processing times based on all application types.

Figure 8(a). The *FuB* algorithm considers both user configuration requirements and the current server's available capacity without having any information about future resource requests. Moreover, the *FuB* algorithm considers the specified constraints of server geographic location while eliminating servers (see Algorithm 1).

Figures 8(b) and 10 show the service times. Service time is calculated based on network delay and processing time. Because group A has fewer successfully processed tasks than group B and the *FuB* algorithm, as seen in Figure 10(a), the service time for these successfully processed tasks is lower. Moreover, the worst fit algorithm showed the best performance, compared with group B in terms of the failed task ratio. The worst fit and *FuB* algorithms had better perfor-

mance than both groups A and B. Therefore, they take longer while having lower failed task ratios than group A. Figure 10(d) clearly shows the service time for each application type. According to Table 5, the task lengths generated are small, medium, big, and very big, for healthcare, AR, infotainment, and compute-intensive applications, respectively.

Similarly, Figure 8(c) and Figure 11 show the processing times for the application tasks. In all cases, group A showed longer processing times because the requested tasks were allocated to the first server with free resources or the smallest free-resource capacity. It makes for a long queuing time at the edge server [54]. *Worst fit* and *FuB* algorithms have approximately the same performance results when the system load is low. However, when the system is overloaded, if

FIGURE 9: Average failed tasks, service times, and processing times based on all application types.

big and very big tasks for infotainment and compute-intensive applications are processed on the edge servers, healthcare and AR applications can be offloaded to the cloud server. Therefore, the processing times for healthcare and AR applications were lower with the *worst fit* algorithm.

The following concluding remarks can be made based on the experimental results in Figures 8, 10, and 11:

- (1) The *FuB* algorithm shows the lowest failed task ratio compared with the five basic balancing algorithms used for a fuzzy-based MEO. Among the five basic balancing algorithms, *worst fit* shows the best performance results in terms of the failed task ratio
- (2) *Best fit* and *first fit* take the longest service time and processing time because the requested task will be

allocated to the first available server with resource capacity. *Random fit* randomly chooses the server with available resource capacity that meets the requirement of the requested process, whereas *next fit* starts searching for available resource capacity from where it left off, not from the beginning. Two methods have the highest failed task ratios (Figure 8(a)); therefore, the processing and service times for these successfully requested tasks are lower

- (3) If the system has a low number of mobile users (e.g., fewer than 1600 devices used), our proposal and *worst fit* have approximately the same results. However, when the system is overloaded, according to Algorithm 1, using the *FuB* algorithm, the healthcare

FIGURE 10: Average service times for different application types.

and AR applications can be offloaded to the cloud server to improve service time and processing time

- (4) To compare our proposal to other existing algorithms, the existing fuzzy-based MEO will use the *worst fit* balancing algorithm. This is discussed in the next section

4.3. A Comparison of Our Proposal and Existing Benchmark Schemes. In this paper, we compare two benchmark schemes that use the fuzzy logic approach for an MEO in IoT environments, as follows:

- (1) *Fuzzy approach* [12] (named *fuzzy*): The fuzzy-based approach considers both computational and communication resources and makes a decision. The MEO

finds a target server: either a local edge server, the nearest neighboring edge server or a cloud server

- (2) *Competitor approach* [17] (named *comp*): This approach utilizes fuzzy logic to decide whether to execute tasks on an edge server or a cloud server

Figure 9 shows the average failed tasks, service times, and processing times based on all application types. When the system is overloaded, our proposal has the lowest failed task ratio, as shown in Figure 9(a). The *FuB* balancing algorithm provides the proximate server in real time by considering user configuration requirements, server geographic locations, and available resource capacities. *Fuzzy* uses the *worst fit* balancing algorithm to distribute tasks to the proximate server that has the largest available free resource capacity, so that the resource capacity left will be big enough to be useful.

FIGURE 11: Average processing times for different application types.

On the other hand, *comp* prefers offloading the tasks to the edge server. Therefore, the failed task ratio in *comp* was the highest, compared to *FuB* and *fuzzy* algorithms, because of the impact of MAN congestion on task failures. Consequently, the service time and processing time for tasks in *comp* were lowest when the system is overloaded, as shown in Figures 9(b) and 9(c).

Figure 12 shows the processing times for each application type. The task lengths generated are small, medium, big, and very big, for healthcare, AR, infotainment, and compute-intensive applications, respectively (see Table 5). Although *comp* prefers offloading tasks to the edge server, the healthcare, AR, infotainment, and compute-intensive applications will be processed on the edge server. Hence, when the number of mobile users is less than 1400, the processing time in

comp was the best. When the number of mobile users is too high, the number of failed tasks in *comp* is high. Consequently, to execute fewer requested tasks than *FuB* and *fuzzy*, *comp* had the lowest performance. Now, we compare our proposal (*FuB*) and *fuzzy*. For healthcare and AR applications when the system load is not high (e.g., under 1400 mobile users), *FuB* and *fuzzy* have approximately the same results. Nevertheless, when the system is overloaded, by using Algorithm 1, *FuB* can support the proximate server in real time by considering user configuration requirements, server geographic locations, and available resource capacities. Accordingly, *FuB* provides better results, as shown in Figures 12(a) and 12(b). On the other hand, the big and very big task sizes of infotainment and compute-intensive applications are mostly executed on the cloud server, which provides

FIGURE 12: Average processing times.

better results because a VM running on a cloud server is very powerful. Inevitably, the processing times for *FuB* and *FuB* have approximately the same results and are lower than *comp*.

5. Conclusions

In this paper, we proposed the online workload balancing algorithm support for a mobile edge orchestrator. Our proposal can support the proximate server when user requests arrive at the MEO in real time by considering user configuration requirements, server geographic location, and available resource capacity for the online algorithm. According to simulation outcomes, our proposal provides better results when the system is overloaded, compared with the existing algorithms. For

healthcare and AR applications, our proposal provides not only the lowest failed task ratio but also the shortest processing time. In future work, we will apply a genetic algorithm to task scheduling in order to improve quality of service.

Data Availability

The data used to support this work are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Institute for Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00818, Machine Learning Based Low Bandwidth Image Communication Edge Computing System for Proactive Anomaly Detection on Smart Plant Environment and No. 2017-0-00294, service mobility support distributed cloud technology).

References

- [1] W. Yu, F. Liang, X. He et al., "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.
- [2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [3] S. Li, L. D. Xu, and S. Zhao, "5g internet of things: a survey," *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, 2018.
- [4] K. Imagane, K. Kanai, J. Katto, T. Tsuda, and H. Nakazato, "Performance evaluations of multimedia service function chaining in edge clouds," in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–4, Las Vegas, NV, USA, 2018.
- [5] K. Kumar and Yung-Hsiang Lu, "Cloud computing for mobile users: can offloading computation save energy?," *Computer*, vol. 43, no. 4, pp. 51–56, 2010.
- [6] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, 2020.
- [7] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.
- [8] Y. Cai, F. R. Yu, and S. Bu, "Cloud computing meets mobile wireless communications in next generation cellular networks," *IEEE Network*, vol. 28, no. 6, pp. 54–59, 2014.
- [9] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, 1611 pages, 2013.
- [10] E. Ahmed, A. Ahmed, I. Yaqoob et al., "Bringing computation closer toward the user network: is edge computing the solution?," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 138–144, 2017.
- [11] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: a survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.
- [12] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 769–782, 2019.
- [13] P. Mach and Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
- [14] J. Shuja, K. Bilal, W. Alasmay, H. Sinky, and E. Alanazi, "Applying machine learning techniques for caching in next-generation edge networks: a comprehensive survey," *Journal of Network and Computer Applications*, vol. 181, article 103005, 2021.
- [15] W. Alasmay and S. Valaee, "Crowd sensing in vehicular networks using uncertain mobility information," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11227–11238, 2019.
- [16] S. K. Zaman, A. Jehangiri, T. Maqsood et al., "Mobility-aware computational offloading in mobile edge networks: a survey," *Cluster Computing*, vol. 24, 2021.
- [17] H. Flores and S. Srirama, "Adaptive code offloading for mobile cloud applications: exploiting fuzzy sets and evidence-based learning," in *MCS '13: Proceeding of the fourth ACM workshop on Mobile cloud computing and services*, pp. 9–16, New York, NY, USA, 2013.
- [18] S. M. Hosseini, M. Kazemini, M. Mehrjoo, and S. M. Barakati, "Fuzzy logic based mobile data offloading," in *2015 23rd Iranian Conference on Electrical Engineering*, pp. 397–401, Tehran, Iran, 2015.
- [19] V. Nguyen, K. Tran, T. Nguyen, C. S. Hong, and E.-N. Huh, "Flexible computation offloading in a fuzzy-based mobile edge orchestrator for iot applications," *Journal of Cloud Computing*, vol. 9, p. 11, 2020.
- [20] M. D. Hossain, T. Sultana, V. Nguyen et al., "Fuzzy based collaborative task offloading scheme in the densely deployed small-cell networks with multi-access edge computing," *Applied Sciences*, vol. 10, no. 9, p. 3115, 2020.
- [21] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: the role of mec in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84–91, 2016.
- [22] ETSI, *Mobile edge computing: A key technology towards 5G*, 2015, 2020, http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf.
- [23] A. Hegyi, H. Flinck, I. Ketyko, P. Kuure, C. Nemes, and L. Pinter, "Application orchestration in mobile edge cloud: Placing of iot applications to the edge," in *2016 IEEE 1st International Workshops on Foundations and Applications of Self Systems (FASW)*, pp. 230–235, Augsburg, Germany, 2016.
- [24] H. Flores, X. Su, V. Kostakos et al., "Large-scale offloading in the internet of things," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 479–484, Kona, HI, USA, 2017.
- [25] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.
- [26] A. C. Baktir, A. Ozgovde, and C. Ersoy, "Enabling service-centric networks for cloudlets using sdn," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 344–352, Lisbon, Portugal, May 2017.
- [27] V. Karagiannis and A. Papageorgiou, "Network-integrated edge computing orchestrator for application placement," in *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–5, Tokyo, Japan, Nov 2017.

- [28] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, and S. Cretti, "Foggy: a platform for workload orchestration in a fog computing environment," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 231–234, Hong Kong, China, 2017.
- [29] B. Li, M. Li, and X. Wu, "Well-behaved online load balancing against strategic jobs," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1243–1251, Richland, SC, 2019.
- [30] P. Berman, M. Charikar, and M. Karpinski, "On-line load balancing for related machines," *Journal of Algorithms*, vol. 35, no. 1, pp. 108–121, 2000.
- [31] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud," in *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 2–11, New York, NY, USA, 2014.
- [32] S. Kamali, "Efficient bin packing algorithms for resource provisioning in the cloud," in *International Workshop on Algorithmic Aspects of Cloud Computing*, pp. 84–98, Cham, 2016.
- [33] M. R. Garey and R. L. Graham, "Bounds for multiprocessor scheduling with resource constraints," *SIAM Journal on Computing*, vol. 4, no. 2, pp. 187–200, 1975.
- [34] S. Kamali and A. López-Ortiz, "Efficient online strategies for renting servers in the cloud," in *International Conference on Current Trends in Theory and Practice of Informatics*, Berlin, Heidelberg, 2014.
- [35] M. Garey, R. Graham, D. Johnson, and A. C.-C. Yao, "Resource constrained scheduling as generalized bin packing," *Journal of Combinatorial Theory, Series A*, vol. 21, no. 3, pp. 257–298, 1976.
- [36] J. Kumar, A. K. Singh, and A. Mohan, "Resource-efficient load-balancing framework for cloud data center networks," *ETRI Journal*, vol. 43, no. 1, pp. 53–63, 2019.
- [37] M. Gabay and S. Zaourar, "Vector bin packing with heterogeneous bins: application to the machine reassignment problem," *Annals of Operations Research*, vol. 242, pp. 161–194, 2016.
- [38] S. K. Mishra, B. Sahoo, and P. P. Parida, "Load balancing in cloud computing: a big picture," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 2, pp. 149–158, 2020.
- [39] S.-C. Wang, K. Yan, W.-P. Liao, and S.-S. Wang, "Towards a load balancing in a three-level cloud computing network," in *2010 3rd International Conference on Computer Science and Information Technology*, pp. 108–113, Chengdu, China, 2010.
- [40] E. M. Rewehel, M.-S. M. Mostafa, and M. O. Ragaie, "New sub-task load balancing algorithm based on olb and lbmm scheduling algorithms in cloud," in *Proceedings of the 2014 International Conference on Computer Network and Information Science*, pp. 9–14, USA, 2014.
- [41] V. R. Kanakala, V. K. Reddy, and K. Karthik, "Performance analysis of load balancing techniques in cloud computing environment," in *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–6, Coimbatore, India, 2015.
- [42] S. I. Kim, H. T. Kim, G. S. Kang, and J.-K. Kim, "Using dvfs and task scheduling algorithms for a hard real-time heterogeneous multicore processor environment," in *Proceedings of the 2013 Workshop on Energy Efficient High Performance Parallel and Distributed Computing*, pp. 23–30, New York, NY, USA, 2013.
- [43] I. S. AlShawi, L. Yan, W. Pan, and B. Luo, "Lifetime enhancement in wireless sensor networks using fuzzy approach and a-star algorithm," *IEEE Sensors Journal*, vol. 12, no. 10, pp. 3010–3018, 2012.
- [44] I. A. Moschakis and H. D. Karatza, "Towards scheduling for internet-of-things applications on clouds: a simulated annealing approach," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 8, pp. 1886–1899, 2015.
- [45] X. Xu, B. Shen, X. Yin et al., "Edge server quantification and placement for offloading social media services in industrial cognitive iov," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2910–2918, 2021.
- [46] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, 2019.
- [47] J. Guo, B. Song, Y. He, F. R. Yu, and M. Sookhak, "A survey on compressed sensing in vehicular infotainment systems," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2662–2680, 2017.
- [48] V. Salmani, R. Ensafi, N. Khatib-Astaneh, and M. Naghibzadeh, "A fuzzy-based multi-criteria scheduler for uniform multiprocessor real-time systems," in *10th International Conference on Information Technology (ICIT 2007)*, pp. 179–184, Rourkela, India, Dec 2007.
- [49] G. Dósa and J. Sgall, "First Fit bin packing: a tight analysis," in *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, *International Proceedings in Informatics (LIPIcs)*, pp. 538–549, Dagstuhl, Germany, 2013.
- [50] D. Johnson, *Near-Optimal Bin Packing Algorithms [Ph.D. thesis]*, Massachusetts Institute of Technology, 2010.
- [51] S. Ghosh, Q. Razouqi, H. J. Schumacher, and A. Celmins, "A survey of recent advances in fuzzy logic in telecommunications networks and new challenges," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 3, pp. 443–447, 1998.
- [52] S. Rathore, P. K. Sharma, A. K. Sangaiah, and J. J. Park, "A hesitant fuzzy based security approach for fog and mobile-edge computing," *IEEE Access*, vol. 6, pp. 688–701, 2018.
- [53] J. M. Mendel, "Fuzzy logic systems for engineering: a tutorial," *Proceedings of the IEEE*, vol. 83, no. 3, pp. 345–377, 1995.
- [54] A. Jaddoa, G. Sakellari, E. Panaousis, G. Loukas, and P. G. Sarigiannidis, "Dynamic decision support for resource offloading in heterogeneous internet of things environments," *Simulation Modelling Practice and Theory*, vol. 101, article 102019, 2020.
- [55] X. Ma, C. Lin, H. Zhang, and J. Liu, "Energy-aware computation offloading of iot sensors in cloudlet-based mobile edge computing," *Sensors*, vol. 18, no. 6, p. 1945, 2018.
- [56] J. Niu, W. Song, and M. Atiquzzaman, "Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications," *Journal of Network and Computer Applications*, vol. 37, pp. 334–347, 2014.
- [57] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031–5044, 2019.
- [58] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: an environment for performance evaluation of edge computing systems," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, p. e3493, 2018.

- [59] M. Silva, D. Freitas, E. Neto, C. Lins, V. Teichrieb, and J. M. Teixeira, "Glassist: using augmented reality on google glass as an aid to classroom management," in *2014 XVI Symposium on Virtual and Augmented Reality*, pp. 37–44, Piata Salvador, Brazil, 2014.
- [60] C. Tunca, N. Pehlivan, N. Ak, B. Arnrich, G. Salur, and C. Ersoy, "Inertial sensor-based robust gait analysis in non-hospital settings for neurological disorders," *Sensors*, vol. 17, no. 4, p. 825, 2017.