FLEX: A Platform for Scalable Service Placement in Multi-Fog and Multi-Cloud Environments

Pedram Farzin
University of Kurdistan
Sanandaj, Iran
p.farzin@eng.uok.ac.ir

Sadoon Azizi
University of Kurdistan
Sanandaj, Iran
s.azizi@uok.ac.ir

Mohammad Shojafar
6GIC/University of Surrey
Guildford, UK
m.shojafar@surrey.ac.uk

Omer Rana
Cardiff University
Cardiff, UK
RanaOF@cardiff.ac.uk

Mukesh Singhal
University of California-Merced
Merced, USA
msinghal@ucmerced.edu

ABSTRACT

With the recent development in the Internet of Things (IoT), big data, and machine learning, the number of services has dramatically increased. These services are heterogeneous in terms of the amount of resources and quality of service (QoS) requirements. To cope with the limitations of Cloud infrastructure providers (CIPs) for latency-sensitive services, many Fog infrastructure providers (FIPs) have recently emerged and their numbers are increasing continually. Due to difficulties such as the different requirements of services, location of end-users, and profile cost of IPs, distributing services across multiple FIPs and CIPs has become a fundamental challenge. Motivated by this, a flexible and scalable platform, FLEX, is proposed in this work for the service placement problem (SPP) in multi-Fog and multi-Cloud computing. For each service, FLEX broadcasts the service's requirements to the resource managers (RMs) of all providers and then based on the RMs' responses, it selects the most suitable provider for that service. The proposed platform is flexible and scalable as it leaves it up to the RMs to have their own policy for service placement. The problem is formulated as an optimization problem and an efficient heuristic algorithm is proposed to solve it. Our simulation results show that the proposed algorithm can meet the requirements of services.

KEYWORDS

Internet of Things (IoT), Fog Computing, Cloud Computing, Multi-Fog and Multi-Cloud, Service Placement, Flexible and Scalable Platform, Quality of Service (QoS).

ACM Reference Format:
Pedram Farzin, Sadoon Azizi, Mohammad Shojafar, Omer Rana, and Mukesh Singhal. 2022. FLEX: A Platform for Scalable Service Placement in Multi-Fog and Multi-Cloud Environments. In *Australasian Computer Science Week 2022 (ACSW 2022), February 14–18, 2022, Brisbane, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3511616.3513105

1 INTRODUCTION

With the recent advances in the Internet of Things (IoT) technologies such as sensors, actuators, RFID, and wireless communications, the number of connected devices has increased exponentially [1]. These devices generate a huge amount of data that needs to be stored, processed, analyzed, and represented to extract valuable information from it. To achieve these goals, many applications and services in the areas of IoT, big data, and machine learning have recently emerged [13, 14, 36]. The characteristics and requirements of these services are different in terms of the amount of resources and quality of service (QoS) they need. For example, services such as healthcare systems [27], virtual reality [3], and autonomous and connected cars [22] are time-sensitive while big data analysis [33], pollution monitoring [8], and scientific computations [15] are delay-tolerant.

Cloud computing is a pay-as-you-go model that provides a ubiquitous computing environment for a wide variety of applications and services. Cloud infrastructure providers (CIPs) bring many advantages to their users, such as low monetary cost and virtually unlimited computing and storage resources. However, the centralized nature of CIPs can lead to high communication latency and network bandwidth consumption [17, 21]. This is in contradiction with the decentralized nature of IoT devices. To overcome with these limitations, Fog computing was introduced as a promising complement to the Cloud [6]. The main purpose of the Fog is extending the Cloud resources and services at the edge of the network. With the development of Fog computing, more and more Fog infrastructure providers (FIPs) are expected to deploy their own infrastructures. Although FIPs provide great benefits to latency-sensitive and bandwidth hungry services, their computing and storage resources are very limited compared with CIPs. Moreover, Fog nodes (FNs) are usually expensive to operate and maintain, which will cost more for end users [30]. Therefore, the combination of FIPs and CIPs, called multi-Fog and multi-Cloud, is becoming a common environment for deploying the emerging services [16].

Application and service placement across Fog and Cloud computing has recently attracted significant attention from both academia and industry [19, 25]. Difficulties such as the different requirements of services, location of end-users, and monetary cost of IPs, make the service placement problem (SPP) a complex task. Unfortunately, this is becoming even more challenging issue as the number of services and IPs is growing day by day. Hence, there is a need for a scalable and flexible platform to select an appropriate IP among the existing FIPs and CIPs for each service.

In recent years, many solutions have been proposed for the SPP in Fog and Cloud computing systems [5, 10, 11, 18, 23, 26, 28, 32]. However, some of them have focused only on the Fog or Cloud environment and do not consider both together [5, 11, 26]. There exist some works that are based on the integrated Fog and Cloud computing platform provided by a specific IP such as Amazon, Microsoft and Google [4, 18, 20]. As time-sensitive services require the low communication latency and computation-intensive services prefer the low monetary cost, a vast majority of the proposed solutions do not take into account the desired delay and cost requirements of service providers [10, 28]. To address these issues, we pose the following research questions: i) how can we design and implement an efficient platform to cope with the SPP in joint multi-Fog and multi-Cloud environments? ii) as existing and future IPs prefer to implement their own resource scheduling policies, how the proposed platform deals with this matter?, and iii) how the desired QoS and monetary cost requirements of each service are met using the proposed platform's placement algorithm?

In this work, we propose FLEX, a flexible and scalable platform for the problem of service placement in environments with multiple Fog and Cloud IPs. FLEX takes different approach from the previous solutions. For each service request, FLEX selects the most appropriate IP from the available ones based on the service's requirements and delegates the resource management to the IP. Pushing the resource management to IPs has two important benefits. First, it allows IPs to implement their own service placement policy, and evolve their policy independently, i.e., the flexibility feature of FLEX. Second, it keeps FLEX simple and makes it easier to support new IPs, i.e., the scalability feature of FLEX. To the best of authors' knowledge, this the first platform for the SPP in multi-Fog and multi-Cloud environments.

Taking into account the aforementioned issues, we propose FLEX as a flexible and scalable platform service placement in multi-fog and multi-cloud computing environments. The major contributions of this work are as follows:

- We propose FLEX, which is a flexible and scalable platform for the service placement problem in multi-Fog and multi-Cloud environments.
- We formulate the problem as an integer linear programming (ILP) model and propose a heuristic algorithm to efficiently solve it.
- Using extensive simulations, we evaluate the performance of the proposed algorithm and show its effectiveness under different experiments.

The remainder of the paper is organized as follows. Section 2 reviews and discusses the related work. The proposed platform including its high-level and detail architecture is described in Section 3. In Section 4, the ILP model of the SPP is presented. The proposed heuristic algorithm for solving the model is given in Section 5. Section 6 evaluates the proposed platform in various cases. Finally, Section 7 concludes the paper, followed by future research directions.

2 RELATED WORK

In this section, we review and discuss some related works that have focused on the service placement problem and frameworks proposed to this problem.

Grozev and Buyya in [11] study the service deployment across multiple Clouds. They propose an approach that considers different aspects including cloud data center selection, load distribution, and auto-scaling that minimizes the overall cost and delay for endsusers. In [26], Omer et al. have focused on the IoT service placement in Cloud data centers. The authors model each service as a set of interdependent virtual machines and formulate the problem as a mixed integer linear programming (MILP) model with the aim of minimizing the energy consumption, resource wastage and network consumption of a Cloud data center. To efficiently solve the model, the propose a priority-aware heuristic algorithm.

Skarlat et al. [29] propose a fog computing framework based on the concept of Fog colony. A Fog colony consists of sensors and actuator devices, Fog cells and Fog nodes. Within each fog colony, there is a fog orchestration control node that manages Fog cells and Fog nodes. If there are insufficient resources to support an IoT application, the fog orchestration control node sends the application to another colony or cloud. As an extension of this work, the authors in [28] propose FogFrame which provides a decentralized approach to manage applications within a Fog landscape.

Yousefpour et al. [35] introduce FOGPLAN as a dynamic framework for IoT service provisioning in a Fog computing environment. The main goal of FOGPLAN is minimizing the delay violations and total cost. To achieve these goals, two efficient greedy algorithms are proposed by the authors. In [31], the authors investigate the provisioning of heavily stateful low latency services (LLAs) over the Fog environment. They develop FogSpot which exploits the spot pricing mechanism [2] to allocate cloudlets' computing resources to end-users based on their services' demand. Ghaemi et . al in [9] introduce ChainFaaS a blockchain-based serverless platform to use the personal computers' computational capacity to deliver internet-based computing services to end-users. The main aim of ChainFaaS is reducing the cost of users and providing a transparent and reliable platform.

Furthermore, Mahmud et al. [18] propose an Edge affinity based approach to place applications in a Fog-Cloud computing environment in order to meet the QoS requirements of users. Their approach includes three phases. At the first phase, applications are classified according to their main characteristics, i.e., user-defined deadline, amount of data per input, and frequency rate of IoT devices. Then, the allowable number of applications are selected to be hosted on a Fog cluster. Finally, the selected applications are placed on the Fog cluster with the aim of minimizing the service delivery time. In [12], Hassan et al. propose an efficient policy for the placement of IoT services on Fog-Cloud systems. To provide high QoS for IoT users and low energy consumption for FIPs, the authors classify services into critical and normal ones. They propose MinRes algorithm for critical services to minimize response time and MinEng algorithm to reduce the energy consumption of the Fog environment.

In [34], authors first present an ILP model for the SPP and then propose a PageRank-based algorithm to rank applications according to their popularity. The main goal of this work is reducing the latency of popular applications. Natesha and Guddeti in [24] design a two-level framework for resource provisioning in a Fog computing environment. The authors formulate the service placement problem as a multi-objective optimization with the objectives of minimizing the service time, consumed of energy and cost. The problem is solved by an elitism-based genetic algorithm (EGA). Cao et al. [7] propose Edge federation, an integrated resource provisioning model to host latency-critical services in the multiple Edge infrastructure providers (EIPs). The provisioning process is formulated as an linear programming (LP) with the aim of guaranteeing the service latency and minimizing the resource cost. The authors also present a dynamic service provisioning solution for their model in the Edge federation environment.

Although the above-mentioned works have taken valuable steps towards service placement in Fog and Cloud computing systems, none of them have focused on the service placement on the joint multi-Fog and multi-Cloud environments. Also, none of them pay attention to the flexibility and scalability features.

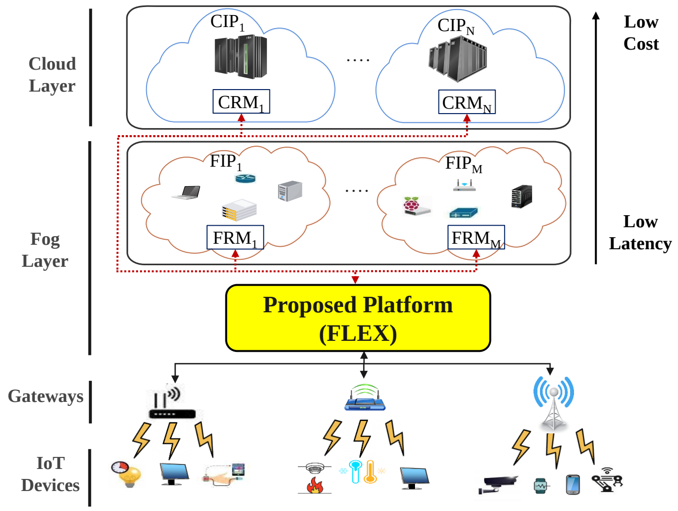


Figure 1: FLEX architecture.

3 FLEX PLATFORM

In this section, we first provide the high-level architecture of the FLEX platform and discuss how FLEX achieves flexibility and scalability. We then describe the components of FLEX.

3.1 High-level Architecture

Fig. 1 depicts the overall view of the multi-Fog and multi-Cloud computing environment. We augment the architectural pattern of IoT-Fog-Cloud with one additional layer, i.e., the proposed platform. The environment consists of five parts, namely, *IoT devices*, *gateways*, *FIPs*, *CIPs*, and the *proposed platform*, each of which are explained below.

- **IoT devices:** This part includes different end-point devices, such as sensors, actuators, RFIDs, smart home appliances, wearables, smartphones, cameras, smart meters, and industry devices. These devices are geographically distributed worldwide and connected to the Internet through gateways using different wireless technologies such as Wi-Fi, Bluetooth, ZigBee, and 3G/4G/5G. Since most IoT devices are resource-limited in terms of processing, storage, and battery power, they cannot host latency-sensitive and computation-intensive services.
- **Gateways:** IoT gateways are edge devices, such as Wi-Fi access points, cellular base stations, and home switches, which are located in close proximity to IoT devices.
- **FIPs:** A multi-Fog environment consists of several FIPs in which each FIP provides its computing, storage, and networking resources. The devices in the Fog layer are known as FNs and usually are richer in resources than edge devices. FNs can be Raspberry Pies, routers, switches, personal computers, servers, cloudlets, and micro data centers. Each FIP dedicates a specialized node, named Fog resource manager (FRM) [18], to manage its resources and establish a persistent communication with the FLEX platform. These nodes could host applications and services in the form of virtual machines or containers.
- **CIPs:** In the top layer of the vertical dimension, CIPs are located, where they provide a lot of services through their large-scale and robust data centers. A cloud data center consists of a pool of virtualized computational and storage resources. CIPs manage their resources and communicate with FLEX through their Cloud resource manager (CRM) node, similar to FIPs. CIPs usually are far from IoT devices, which makes them inappropriate for time-sensitive services. However, they usually are more cost-effective than FIPs.
- **Proposed platform (FLEX):** The main design philosophy of FLEX is to provide a flexible and scalable platform for multi-Fog and multi-Cloud environments to distribute applications and services among FIPs and CIPs efficiently. FLEX receives services and predefined requirements from different service providers and end-users and then performs service placement. When a service or a batch of services are submitted to the FLEX, it broadcasts the service's requirements and users' location to the resource managers (RMs) of all FIPs and CIPs. Then, based on the communication delay and monetary cost offered by IPs, FLEX selects the most appropriate provider for each service. It is worth mentioning that FLEX leaves it up to the RM of IPs to place the service on available nodes based on their placement policy.

3.2 Detailed Architecture

The components of the FLEX are shown in Fig. 2. FLEX has four main components: *Service Receiver*, *Service Analyzer*, *Admission Control*, and *Provider Selector*. The functionality of each component is explained below.

- *Service Receiver:* This component provides an interface for service providers to submit their services. At this step, the requirement of each service, i.e., the service profile, including the amount of computing, storage, and bandwidth resources,

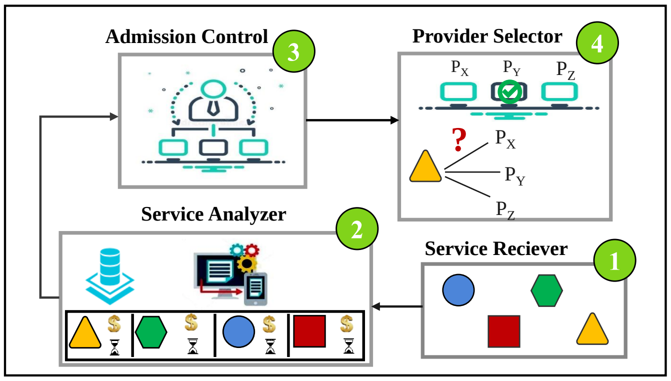


Figure 2: The details of FLEX architecture.

a priority of delay and cost, and security and privacy concerns, is specified. For example, a latency-sensitive service must give much higher weight to delay in comparison with cost.

- *Service Analyzer:* This component analyzes each service and stores it into the service database based on the profile of services. Service analyzer should classify and sort services based on some essential aspects such as the degree of latency-sensitivity, security and privacy sensitivity level, and amount of resource requirements.
- *Admission Control:* The admission control receives service requests from the service analyzer and broadcasts the services to the FRM and CRM of all FIPs and CIPs. This step is called matchmaking process. Within each IP, for each service, the RM checks to see if it can meet the requirements of that service. If yes, then it estimates the communication delay from the user location to the considered node for hosting the service and calculates the cost. Then it responds to the admission control and includes information about the delay and cost. After receiving the responses from all providers, admission control sends the list of providers that can host the service with relevant information to the next component, i.e., provider selector.
- *Provider Selector:* Based on the admission control's list, the provider selector selects the most suitable IP to host each service. The provider selection process can be done using different multi-criteria decision-making (MCDM) methods. In Section V, we propose an efficient heuristic algorithm to be applied in this component. After an IP is selected, both the service user(s) and the IP are informed. Then, the address of the hosted node is stored in the corresponding gateway(s) to redirect the service requests to the relevant hosted node.

Note that the cross-IP interactions between the admission control and Fog and Cloud RMs happen only once, immediately after the service placement request. After the end-user(s) is served, the communication between the end-user and selected IP will be direct.

4 ADDRESSING SERVICE PLACEMENT IN FLEX

In this section, we formulate the service placement problem as an integer linear programming model.

4.1 Sets

Let $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ be the set of n services where each i-th service $\mathcal{S}_i$ has some specific characteristics. The resource requirement of service $\mathcal{S}_i$ can be represented as $\mathcal{S}_i^r$ where r can belong to R={CPU, memory, bandwidth, storage}. Also, we use $\mathcal{S}_i^s$ to denote the size (in terms of the number of million instructions – MI) of the service $\mathcal{S}_i$. For each service $\mathcal{S}_i$, a coefficient $\alpha \in [0, 1]$ shows the importance of delay and cost for that service. Let $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_M\}$ be the set of M FIPs where each FIP $\mathcal{F}_j$ has $|\mathcal{F}_j|$ FNs. The notation $\mathcal{F}_{j,l}^r$ is used to denote the resource capacity of the l-the FN of the FIP $\mathcal{F}_j$ along different $r \in R$ dimensions. Similarly, let $C = \{C_1, C_2, \ldots, C_N\}$ be the set of N CIPs where each CIP C_k has $|C_k|$ cloud nodes (CNs). We use $C_{k,l}^r$ to denote the resource capacity of the l-the CN of the CIP C_k along each of r dimensions. We use $c(\mathcal{F}_j^r)$ and $c(C_k^r)$ to define the resource cost of the FIP $\mathcal{F}_j$ and CIP C_k, respectively, along each dimension.

4.2 Decision Variables

Our model has two main decision variables, i.e., $x_{j,l}^i$ and $y_{k,l}^i$, which are defined as follows.

$$x_{j,l}^i = \begin{cases} 1 & \text{if service } \mathcal{S}_i \text{ is hosted on the } \mathcal{F}_{j,l} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and

$$y_{k,l}^i = \begin{cases} 1 & \text{if service } \mathcal{S}_i \text{ is hosted on the } C_{k,l} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

4.3 Delay

Here we present a formulation of the delay of service placement strategy in our system model. To obtain the delay of a request for the i-th service $\mathcal{S}_i$, denoted by D_i, we should take into account the following delay factors in the model.

- **Communication time (c_i):** This is the time it takes for a request to reach from an IoT device $\mathcal{I}_z$ to the computational node that the i-th service $\mathcal{S}_i$ is hosted on it.

$$c_i = \sum_{j=1}^{M} \sum_{k=1}^{N} \sum_{l=1}^{\max(|\mathcal{F}_j|,|C_k|)} \Big[D\left(\mathcal{I}_z \to \mathcal{F}_{j,l}\right) \times x_{j,l}^i + D\left(\mathcal{I}_z \to C_{k,l}\right) \times y_{k,l}^i \Big], \quad \forall i \in \mathcal{S} \tag{3}$$

where $D\left(\mathcal{I}_z \to \mathcal{F}_{j,l}\right)$ and $D\left(\mathcal{I}_z \to C_{k,l}\right)$ are the communication delay from IoT device $\mathcal{I}_z$ to the l-the FN of the FIP $\mathcal{F}_j$ and l-the CN of the CIP C_k, respectively.

- **Execution time (e_i):** This is the time needed to process the service request. Thus, we have

$$e_i = \frac{\mathrm{S}_i^s}{\mathrm{S}_i^{CPU}}, \quad \forall i \in \mathcal{S} \tag{4}$$

where S_i^{CPU} is the CPU requirement of the i-th service S_i (in unit of million instruction per second - MIPS). Hence, the delay is achieved using the following constraint.

$$D_i = 2 \times c_i + e_i, \quad \forall i \in \mathcal{S} \tag{5}$$

Therefore, we use the following equation to have the total weighted delay of a service placement strategy.

$$\mathbb{D} = \sum_{i=1}^{n} \alpha_i \times D_i \tag{6}$$

4.4 Cost

Depending on the selected provider for the i-th service S_i , the cost for that service can be calculated as follows.

$$C_i = \sum_{j=1}^{M}\sum_{k=1}^{N}\sum_{\forall r \in R}\left[c(\mathcal{F}_j^r)\times x_{j,l}^i + c(C_k^r)\times y_{k,l}^i\right], \quad \forall i \in \mathcal{S} \tag{7}$$

Therefore, the total weighted cost for all of n services can be given as the below.

$$\mathbb{C} = \sum_{i=1}^{n}(1-\alpha_i)\times C_i \tag{8}$$

4.5 The Objective Function

The main goal of the FLEX is to solve the service placement problem to minimize the total weighted delay and cost of the services simultaneously. Hence, our final objective function can be represented as follows. Let $\mathbb{S} = \{1, 2, \dots, \max(|\mathcal{F}_j|, |C_k|)\}$

$$\min\left(\mathbb{C} + \mathbb{D}\right) \tag{9}$$

Subject to the following constraints:

$$\sum_{i=1}^{n} x_{j,l}^i + y_{k,l}^i = 1, \ \forall j \in \mathcal{F}, \forall k \in C, \forall l \in \mathbb{S} \tag{10}$$

$$\sum_{i=1}^{n} \mathcal{S}_i^r \times x_{j,l}^i \le \mathcal{F}_{j,l}^r, \ \forall j \in \mathcal{F}, \forall r \in R, \forall l \in \{1, 2, \dots, |\mathcal{F}_j|\} \tag{11}$$

$$\sum_{i=1}^{n} \mathcal{S}_i^r \times y_{k,l}^i \le C_{k,l}^r, \ \forall k \in C, \forall r \in R, \forall l \in \{1, 2, \dots, |C_k|\} \tag{12}$$

$$x_{j,l}^i \in \{0,1\}, y_{k,l}^i \in \{0,1\} \tag{13}$$

where equality (10) is service constraint and ensures that each service can be assigned only on one computing node. Constraints (11) and (12) guarantee that resource demand of all services must not exceed from capacity of fog nodes and cloud nodes, respectively. Finally, constraint (13) specifies the problem's variable domains (binary).

Theorem 4.1. *SPP in multi-Fog and multi-Cloud environments is in the class of NP-hard problems.*

Proof.

5 FLEX'S HEURISTIC ALGORITHM

In this section, we describe our proposed heuristic algorithm to solve the service placement problem efficiently. The main goal of the proposed algorithm is jointly minimizing the cost and delay for each service. To achieve this goal, it ranks providers based on the service's preferences in terms of cost and delay. So, we name the proposed algorithm as the *minimum cost and delay first (MCD1).* A provider which offers the minimum weighted cost and delay is selected. The detail of the proposed algorithm is as follows.

Assume a list of n services is submitted to FLEX to find a suitable provider for each of them. First of all, FLEX sorts services based on their delay sensitivity, i.e., it tries to give higher priority to the services with the higher α value. Then, as we discussed in subsection 3.2, for a given service, the admission control broadcasts the service's requirements and its user's location to all FIPs and CIPs. After that, it waits until all contacted FRMs and CRMs respond to the admission control. For each service, the response of the FRMs and CRMs includes three values: (i) A Boolean value to indicate whether the provider can serve the service; (ii) The amount of monetary cost for the service, and, (iii) An estimation of the delay from the user's gateway to the candidate node for hosting the service.

Based on the Boolean value, the admission control sends the list of candidate providers to the provider selector component for each service. After receiving the list of candidate providers, the provider selector executes the *MCD1* algorithm to select the most suitable provider for each service.

Algorithm 1 shows the pseudocode of the proposed MCD1 algorithm. The algorithm receives the service $\mathcal{S}_i$, **P** is the list of candidate FIPs and CIPs, **C** is the monetary cost vector for service $\mathcal{S}_i$ and **D** as the delay vector for service $\mathcal{S}_i$, and introduces the most suitable provider based on the preferences of service $\mathcal{S}_i$. Let C_{ij} and D_{ij} are the monetary cost and delay of provider P_j for service $\mathcal{S}_i$, respectively (see lines 1 and 2). In lines 3 and 4, the algorithm respectively finds the provider with the maximum cost and delay. The goal **of the loop**, i.e., lines 5 to 9, is to score providers based on their reported cost and delay and service preferences of service $\mathcal{S}_i$. To this end, we first normalize their cost and delay (lines 6 and 7) and then calculate the objective function for each provider based on the predefined weighted of service $\mathcal{S}_i$ assigned to delay and cost. Next, the objective function vector is created (line 10). Finally, line 11 finds the provider with the minimum objective function and selects it at the destination provider for hosting the service $\mathcal{S}_i$ (lines 12 and 13).

Algorithm 1 MCD1 Algorithm

```
INPUT: $\mathcal{S}_i$, P:list of candidate FIPs and CIPs, C: monetary cost vector for $\mathcal{S}_i$,
D:delay vector for $\mathcal{S}_i$
OUTPUT: Selecting the most suitable provider for $\mathcal{S}_i$
1: Let $C_{ij} \in$ C is the monetary cost of provider $P_j$ for $\mathcal{S}_i$;
2: Let $D_{ij} \in$ D is the delay of provider $P_j$ for $\mathcal{S}_i$;
3: $C_j^{max} \leftarrow$ find maximum $C_{ij} \in$ C;
4: $D_j^{max} \leftarrow$ find maximum $D_{ij} \in$ D;
5: for each $P_j \in$ P do
6:     $C_{ij}^{norm} \leftarrow C_{ij}/C_{ij}^{max}$;
7:     $D_{ij}^{norm} \leftarrow D_{ij}/D_{ij}^{max}$;
8:     $F_{ij} \leftarrow \alpha_i \times C_{ij}^{norm} + (1-\alpha_i) \times D_{ij}^{norm}$;
9: end for
10: Let F is objective function vector for $\mathcal{S}_i$;
11: $F_j^{min} \leftarrow$ find minimum $F_{ij} \in$ F;
12: Let $P_{index}$ is the provider with the value of $F_j^{min}$;
13: return $P_{index}$ as the destination provider for hosting $\mathcal{S}_i$;
```

The time complexity analysis of our proposed algorithm is as follows. The worst-case complexity of lines 3 and 4 is $O(M + N)$, where M and N are the number of FIPs and CIPs, respectively. Note that $|\mathbf{P}| = M + N$. The normalization step, i.e., lines 5-9, also requires $O(M + N)$. Again, the worst-case time complexity of line

11 is $O(M+N)$. Therefore, the overall time complexity of MCD1 for one service is $O(|\mathbf{P}|)$, i.e., $O(M+N)$.

6 PERFORMANCE EVALUATION

This section presents comprehensive experiments to evaluate the FLEX platform's performance. To this end, we have implemented FLEX using a custom simulation environment written in Java programming language. All the experiments were carried out on a PC with Intel Core i7-4790 CPU 3.6 GHz (4 processors), 8 GB RAM and windows 10 OS.

6.1 Simulation settings

To fully understand the advantages of our proposed platform and its heuristic algorithm, we tacked into account four experiments to show the impact of different scenarios (Table 1). For each experiment, we considered two different values for the rate of latency-sensitive services to all services and the rate of FIPs to all providers, i.e., 25% and 75%. The latency-sensitive services put a lot of emphasis on delay, e.g., $\alpha \geq 0.9$.

Table 1: Experiments settings. Rate of latency-sensitivity:= $R(s)$; Rate of FIPs:= $\mathcal{F}(f)$.

Experiments	Services	Providers	$R(s)$	$\mathcal{F}(f)$
1	100	8	(25%, 75%)	(25%, 75%)
2	100	20	(25%, 75%)	(25%, 75%)
3	500	8	(25%, 75%)	(25%, 75%)
4	500	20	(25%, 75%)	(25%, 75%)

Since the real dataset is not available for simulating the environment, including services and FIPs and CIPs, we used a synthetic dataset in our experiments Table 2 and Table 3 show the attributes of services and Fog, Cloud nodes, respectively. We considered the limited value for the number of Fog nodes of each FIP, i.e., [6,12]. However, for CIPs, such a restriction is not imposed.

Table 2: Attributes of Services.

Parameter	Value	Unit
CPU requirements	[300, 800]	(MIPS)
Memory requirements	[0.5,2]	(GB)
Number of instructions	[400, 1500]	(MI)

Table 3: Attributes of Fog/Cloud nodes.

Parameter	Fog	Cloud	Unit
Processing power	[600,2000]	[4000,10000]	(MIPS)
Memory capacity	[4,8]	[16,32]	(GB)
CPU usage cost	[0.3,0.7]	[0.2,0.4]	(G$ per MIPS)
Memory usage cost	[0.05,0.08]	[0.03,0.06]	(G$ per MB)
Communications delay	[5,15]	[50,1250]	(ms)

6.2 Simulation metrics

To evaluate the performance of the FLEX's heuristic algorithm, the following metrics are used in the experiments.

- **Average Weighted Delay (AWD):** We use the following equation to measure the delay provided by a service placement strategy to host n services on a multi-Fog and multi-Cloud environment.

$$AWD = \frac{1}{n} \times \sum_{i=1}^{n} \alpha_i \times \mathcal{D}_i \tag{14}$$

- **Average Weighted Cost (AWC):** This metric is used to evaluate the performance of a service placement strategy in term of cost.

$$AWC = \frac{1}{n} \times \sum_{i=1}^{n} (1-\alpha_i) \times C_i \tag{15}$$

- **Objective Function (OF):** It is minimum values of cost and delay simultaneously point to the performance of a service placement strategy in both of delay and cost perspective.

$$OF = \frac{1}{n} \times \left(\sum_{i=1}^{n} \alpha_i \times \frac{\mathcal{D}_i}{\mathcal{D}_i^{max}} + (1-\alpha_i) \times \frac{C_i}{C_i^{max}} \right) \tag{16}$$

where $\mathcal{D}_i^{max}$ and C_i^{max} denote the maximum possible delay and cost which can respectively provided for i-th service $\mathcal{S}_i$. It is worth mentioning that the decreased value of this metric represents the enhanced performance of a placement policy in terms of both delay and cost.

6.3 Baseline Algorithms

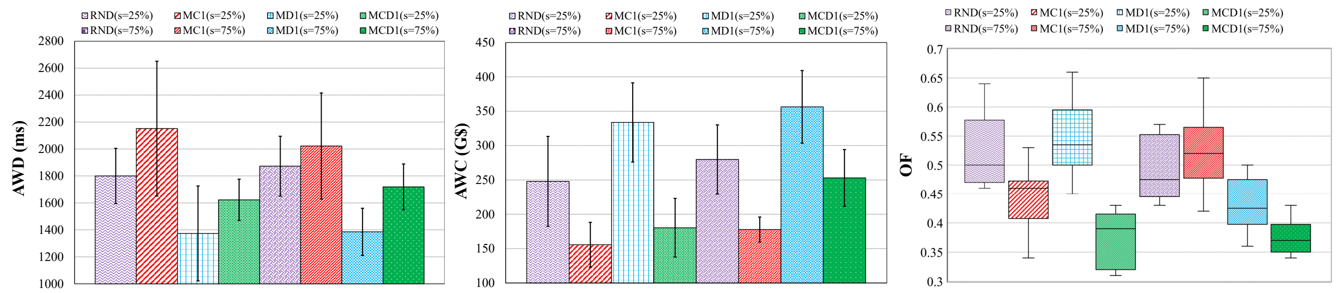
The performance of the proposed FLEX's heuristic algorithm, i.e., MCD1, is compared with the following baselines.

- **Random (RND):** This strategy selects a random provider for each service.
- **The Most Cost-effective Provider First (MC1):** For each service placement request, MC1 selects the provider which offers the minimum cost for that service.
- **The Minimum Delay Provider First (MD1):** This algorithm selects the provider with the minimum delay for each service.

6.4 Results

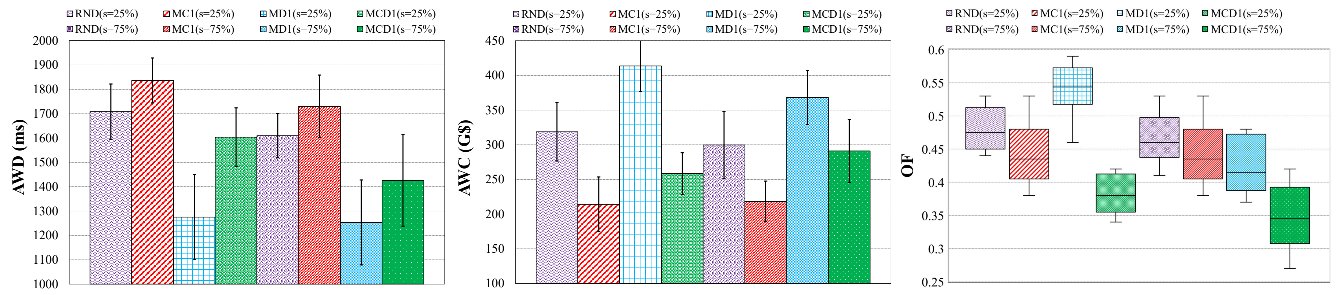
In this subsection, we discuss the results of the four considered experiments.

6.4.1 Experiment one. Fig. 3 and Fig. 4 show the simulation results for experiment one. Generally speaking, as the rate of FIPs increases from $f = 25\%$ (Fig. 3) to $f = 75\%$ (Fig. 4), the AWD of all policies significantly decreases. However, this is vice-versa for the AWC. This is expected since FIPs usually provide lower delay but a higher cost in comparison with CIPs. Also, by increasing the rate of latency-sensitive services, i.e., s, the proposed MCD1 places a higher percentage of services on FIPs to satisfy the QoS's end-user requirement. Thus, its AWD is decreased while its AWC is increased. The other important point is that MC1 (MD1) gives the minimum AWC (AWD) while it has the maximum AWD (AWC). From Fig. 3c and Fig. 4c, it is evident that MCD1 has excellent performance. This is because the proposed MCD1 is the service's profile aware as it selects the most suitable provider based on both delay and cost.



(a) AWD (b) AWC (c) OF

Figure 3: Simulation results for `experiment one` with #Services=100, #Providers=8, f=25%.



(a) AWD (b) AWC (c) OF

Figure 4: Simulation results for `experiment one` with #Service=100, #Providers=8, f=75%.

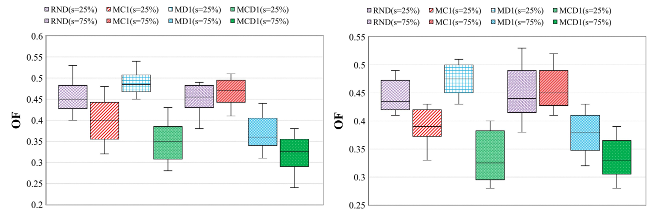
6.4.2 Experiment two. Table 4 demonstrates the performance of the algorithms in terms of the AWD and AWC. As can be found from the table, the proposed MCD1 achieves better trade-off between the AWD and AWC than the other strategies. The OF results for Experiment two have been displayed in Fig. 5. From the figure, we can observe that the proposed MCD1 significantly performs better than the others in all cases. In particular, the proposed policy can reduce the OF by 25.6%, 26.8%, and 11.4% compared to RND, MC1, and MD1, respectively, for the case s=75% and f=75%.

Table 4: AWD and AWC results of `experiment two` for #Service=100, #Providers=20.

Comparing Algorithms	Ratio s	AWD (ms) f=25%	AWD (ms) f=75%	AWC (G$) f=25%	AWC (G$) f=75%
RND	25%	1854.3	1646.8	260.6	302.7
		(std=283.3)	(std=214.5)	(std=30.1)	(std=52)
	75%	1721.7	1642.8	247.5	288.2
		(std=143.6)	(std=119)	(std=30.1)	(std=39.9)
MC1	25%	2152.5	1960.2	171.6	197.2
		(std=400.3)	(std=221.1)	(std=11.8)	(std=24.2)
	75%	1918.8	1803.9	165.4	196.6
		(std=252.7)	(std=126.3)	(std=12.9)	(std=24)
MD1	25%	1364.3	1075.8	350.5	370.4
		(std=160.4)	(std=252.3)	(std=43.5)	(std=40.3)
	75%	1359.2	991.9	348	376.5
		(std=217.5)	(std=244.2)	(std=36.4)	(std=27.8)
MCD1	25%	1719.5	1742	207.3	233.2
		(std=176.3)	(std=107.1)	(std=8.1)	(std=25.8)
	75%	1543.9	1274.2	238.5	290.8
		(std=159.4)	(std=142.1)	(std=33.6)	(std=32)

Table 5: AWD and AWC results of `experiment four` for #Service=500, #Providers=20.

Comparing Algorithms	Ratio s	AWD (ms) f=25%	AWD (ms) f=75%	AWC (G$) f=25%	AWC (G$) f=75%
RND	25%	1731.8	1546	251.4.2	323.5
		(std=61.8)	(std=120.2)	(std=33.6)	(std=31.7)
	75%	1726.5	1597.2	244.8	317.3
		(std=172.4)	(std=166.2)	(std=48.4)	(std=57.8)
MC1	25%	1929.2	1779.3	172.6	190.6
		(std=346.6)	(std=179.9)	(std=10.3)	(std=25.5)
	75%	2107	1758.3	169.3	186.6
		(std=452.3)	(std=117.7)	(std=7.6)	(std=21.4)
MD1	25%	1172.7	1044.7	433.4	385.6
		(std=167.7)	(std=182.8)	(std=70)	(std=40.1)
	75%	1316	1102.9	404.5	384.8
		(std=141)	(std=163.3)	(std=46.3)	(std=33.2)
MCD1	25%	1653.3	1612.6	199.8	223.3
		(std=67.7)	(std=99)	(std=6.7)	(std=26.9)
	75%	1472.6	1336.6	287.3	298.6
		(std=131.6)	(std=153.3)	(std=31.1)	(std=19.9)



(a) OF- f=25% (b) OF- f=75%

Figure 5: OF results for `experiment two` with #Service=100, #Providers=20.

6.4.3 *Experiment three.*

The simulation results for Experiment three are presented in Fig. 6 and Fig. 7. The overall behavior of the algorithms is almost similar to experiment one. In general, as the number of services increases, the AWD is also increases. This is expected since more services need to be placed on CIPs. By growing the rate of both latency-sensitive services and FIPs, the MD1 and proposed MCD1 algorithms perform significantly better in terms of the AWD compared with RND and MC1. However, the increase in the AWC of the MCD1 is much lower than MD1. This is due the fact that our MCD1 prefers CIPs for latency-tolerant services even though FIPs are also available.

6.4.4 *Experiment four.*

Table 5 and Fig. 8 show the results of experiment four. From the table and figure, we can observe that by considering both of the AWD and AWC, the proposed MCD1 shows the best performance in all cases. When the rate of latency-sensitive services is low, i.e., s=25%, MC1 performs better than MD1. However, this is vice-versa for the cases that s=75%. In particular, for the scenario with s=25% and f=75%, the improvement of the proposed MCD1 in terms of the average of OF is 28.5%, 13.7% and 33.1% in comparison with RND, MC1 and MD1, respectively.

7 CONCLUSIONS AND FUTURE WORK

This work introduces FLEX, a novel platform for the service placement problem in multi-Fog and multi-Cloud environments. FLEX provides two important features, flexibility and scalability. FLEX is flexible as it allows Fog and Cloud IPs to implement their own service placement strategy. It is scalable as new Fog and Cloud providers can easily be added to the platform. We formulated the service placement problem as an optimization problem with the aim of delay and cost minimization. Then, we proposed a delay and cost-aware algorithm to efficiently solve the problem. We implemented FLEX using a simulation environment and conducted various experiments to numerically evaluate the performance of the FLEX. Compared with the baseline policies, simulation results show that the proposed heuristic algorithm performs significantly better than the others. As a future work, we intend to use game theory techniques for the provider selection phase.

REFERENCES

[1] Accessed: 2021-09-05. Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[2] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. 2013. Deconstructing Amazon EC2 spot instance pricing. *ACM Transactions on Economics and Computation (TEAC)* 1, 3 (2013), 1–20.

[3] Derian Alencar, Cristiano Both, Rodolfo Antunes, Helder Oliveira, Eduardo Cerqueira, and Denis Rosário. 2021. Dynamic Microservice Allocation for Virtual Reality Distribution with QoE support. *IEEE Transactions on Network and Service Management* (2021).

[4] Sadoon Azizi, Fariba Khosroabadi, and Mohammad Shojafar. 2019. A priority-based service placement policy for Fog-Cloud computing systems. *Computational Methods for Differential Equations* 7, 4 (Special Issue) (2019), 521–534.

[5] Gaurav Baranwal, Ravi Yadav, and Deo Prakash Vidyarthi. 2020. QoE aware IoT application placement in fog computing using modified-topsis. *Mobile Networks and Applications* 25, 5 (2020), 1816–1832.

[6] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. 2014. Fog computing: A platform for internet of things and analytics. In *Big data and internet of things: A roadmap for smart environments*. Springer, 169–186.

[7] Xiaofeng Cao, Guoming Tang, Deke Guo, Yan Li, and Weiming Zhang. 2020. Edge federation: Towards an integrated service provisioning model. *IEEE/ACM Transactions on Networking* 28, 3 (2020), 1116–1129.

[8] Swati Dhingra, Rajasekhara Babu Madda, Amir H Gandomi, Rizwan Patan, and Mahmoud Daneshmand. 2019. Internet of Things mobile–air pollution monitoring system (IoT-Mobair). *IEEE Internet of Things Journal* 6, 3 (2019), 5577–5584.

[9] Sara Ghaemi, Hamzeh Khazaei, and Petr Musilek. 2020. ChainFaaS: An open blockchain-based serverless platform. *IEEE Access* 8 (2020), 131760–131778.

[10] Mohammad Goudarzi, Huaming Wu, Marimuthu Palaniswami, and Rajkumar Buyya. 2020. An application placement technique for concurrent IoT applications in edge and fog computing environments. *IEEE Transactions on Mobile Computing* 20, 4 (2020), 1298–1311.

[11] Nikolay Grozev and Rajkumar Buyya. 2014. Multi-cloud provisioning and load distribution for three-tier applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 9, 3 (2014), 1–21.

[12] Hiwa Omer Hassan, Sadoon Azizi, and Mohammad Shojafar. 2020. Priority, network and energy-aware placement of IoT-based application services in fog-cloud environments. *IET communications* 14, 13 (2020), 2117–2129.

[13] Álvaro Brandón Hernández, María S Perez, Smrati Gupta, and Victor Muntés-Mulero. 2018. Using machine learning to optimize parallelism in big data applications. *Future Generation Computer Systems* 86 (2018), 1076–1092.

[14] Nathaniel Hudson, Hana Khamfroush, and Daniel E Lucani. 2021. QoS-aware placement of deep learning services on the edge with multiple service implementations. *arXiv preprint arXiv:2104.15094* (2021).

[15] Li Liu, Miao Zhang, Rajkumar Buyya, and Qi Fan. 2017. Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. *Concurrency and Computation: Practice and Experience* 29, 5 (2017), e3942.

[16] Juan Luo, Luxiu Yin, Jinyu Hu, Chun Wang, Xuan Liu, Xin Fan, and Haibo Luo. 2019. Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT. *Future Generation Computer Systems* 97 (2019), 50–60.

[17] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. 2018. Fog computing: A taxonomy, survey and future directions. In *Internet of everything*. Springer, 103–130.

[18] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2019. Edge affinity-based management of applications in fog computing environments. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 61–70.

[19] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2020. Application management in fog computing environments: A taxonomy, review and future directions. *ACM Computing Surveys (CSUR)* 53, 4 (2020), 1–43.

[20] Redowan Mahmud, Satish Narayana Srirama, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2020. Profit-aware application placement for integrated fog–cloud computing environments. *J. Parallel and Distrib. Comput.* 135 (2020), 177–190.

[21] Mithun Mukherjee, Lei Shu, and Di Wang. 2018. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Communications Surveys & Tutorials* 20, 3 (2018), 1826–1857.

[22] Ashish Nanda, Deepak Puthal, Joel JPC Rodrigues, and Sergei A Kozlov. 2019. Internet of autonomous vehicles communications security: overview, issues, and directions. *IEEE Wireless Communications* 26, 4 (2019), 60–65.

[23] BV Natesha and Ram Mohana Reddy Guddeti. 2018. Heuristic-based IoT application modules placement in the fog-cloud computing environment. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 24–25.

[24] BV Natesha and Ram Mohana Reddy Guddeti. 2021. Adopting elitism-based Genetic Algorithm for minimizing multi-objective problems of IoT service placement in fog computing environment. *Journal of Network and Computer Applications* 178 (2021), 102972.

[25] Zahra Makki Nayeri, Toktam Ghafarian, and Bahman Javadi. 2021. Application placement in Fog computing with AI approach: Taxonomy and a state of the art survey. *Journal of Network and Computer Applications* (2021), 103078.

[26] Shvan Omer, Sadoon Azizi, Mohammad Shojafar, and Rahim Tafazolli. 2021. A priority, power and traffic-aware virtual machine placement of IoT applications in cloud data centers. *Journal of Systems Architecture* 115 (2021), 101996.

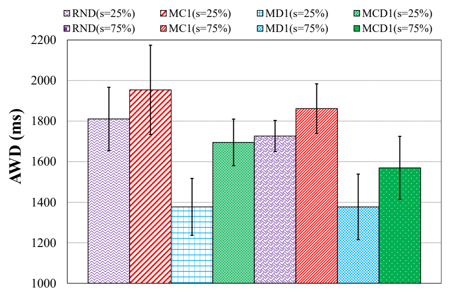
[27] Jun Qi, Po Yang, Geyong Min, Oliver Amft, Feng Dong, and Lida Xu. 2017. Advanced internet of things for personalised healthcare systems: A survey. *Pervasive and Mobile Computing* 41 (2017), 132–149.

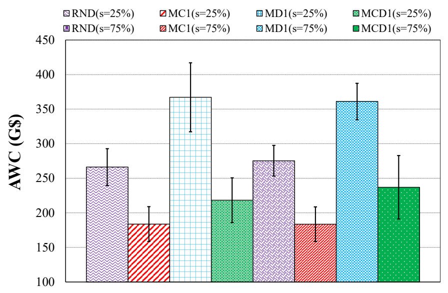
[28] Olena Skarlat and Stefan Schulte. 2021. FogFrame: a framework for IoT application execution in the fog. *PeerJ Computer Science* 7 (2021), e588.

[29] Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. 2016. Resource provisioning for IoT services in the fog. In *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*. IEEE, 32–39.

[30] Balázs Sonkoly, János Czentye, Márk Szalay, Balázs Németh, and László Toka. 2021. Survey on Placement Methods in the Edge and Beyond. *IEEE Communications Surveys & Tutorials* (2021).

[31] Argyrios Tasiopoulos, Onur Ascigil, Ioannis Psaras, Stavros Toumpis, and George Pavlou. 2019. Fogspot: Spot pricing for application provisioning in edge/fog computing. *IEEE Transactions on Services Computing* (2019).


(a) AWD


(b) AWC

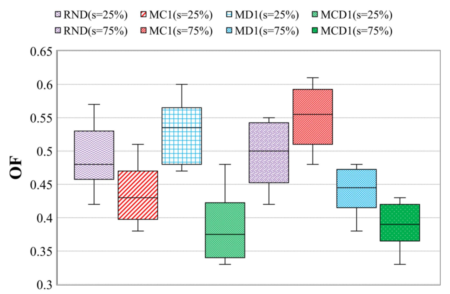
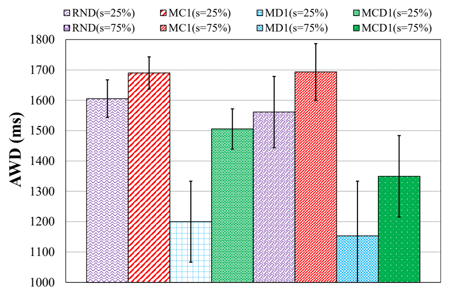
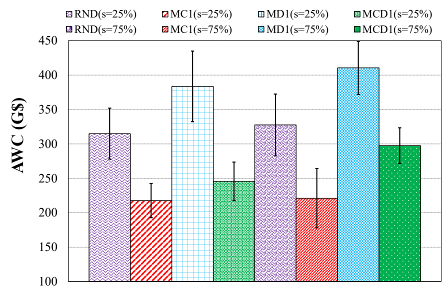

(c) OF

Figure 6: Simulation results for `experiment three` with #Service=500, #Providers=8, f=25%.


(a) AWD


(b) AWC

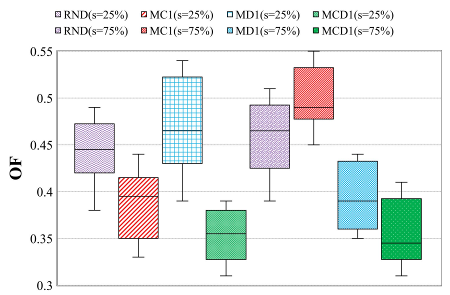
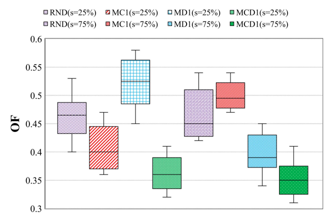

(c) OF

Figure 7: Simulation results for `experiment three` with #Service=500, #Providers=8, f=75%.


(a) OF- f=25%

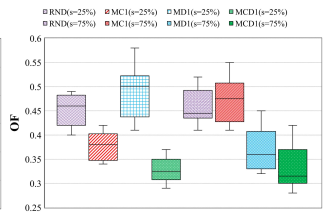

(b) OF- f=75%

Figure 8: OF results for `experiment four` with #Service=500, #Providers=20.

[32] Farhad Tavousi, Sadoon Azizi, and Abdulbaghi Ghaderzadeh. 2022. A fuzzy approach for optimal placement of IoT applications in fog-cloud computing. *Cluster Computing* 25 (2022), 303–320.

[33] Muhammad Habib ur Rehman, Ibrar Yaqoob, Khaled Salah, Muhammad Imran, Prem Prakash Jayaraman, and Charith Perera. 2019. The role of big data analytics in industrial Internet of Things. *Future Generation Computer Systems* 99 (2019), 247–259.

[34] Karima Velasquez, David Perez Abreu, Luís Paquete, Marilia Curado, and Edmundo Monteiro. 2020. A rank-based mechanism for service placement in the fog. In *2020 IFIP Networking Conference (Networking)*. IEEE, 64–72.

[35] Ashkan Yousefpour, Ashish Patil, Genya Ishigaki, Inwoong Kim, Xi Wang, Hakki C Cankaya, Qiong Zhang, Weisheng Xie, and Jason P Jue. 2019. FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework. *IEEE Internet of Things Journal* 6, 3 (2019), 5080–5096.

[36] Kamal Aldein Mohammed Zeinab and Sayed Ali Ahmed Elmustafa. 2017. Internet of things applications, challenges and related future technologies. *World Scientific News* 2, 67 (2017), 126–148.