Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Scalable and Flexible Platform for Service Placement inMulti-Fog and Multi-Cloud Environments

Sadoon Azizi (s.azizi@uok.ac.ir)
University of Kurdistan
Pedram Farzin (p.farzin@uok.ac.ir)
University of Kurdistan
Mohammad Shojafar (m.shojafar@surrey.ac.uk)
University of Surrey
Omer Rana (RanaOF@cardiff.ac.uk)
Cardiff University

Research Article

Keywords:

DOI: https://doi.org/

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Noname manuscript No.
(will be inserted by the editor)

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

Sadoon Azizi · Pedram Farzin · Mohammad Shojafar · Omer Rana

Received: 00 November 0000 / Revised: 00 December 0000/ Accepted: 00 February 0000

Abstract Provisioning services for Internet of Things (IoT) devices leads to several challenges: heterogeneity of IoT devices, varying Quality of Services (QoS) requirements, and increasing availability of both Cloud and Fog resources. The last of these is most significant to cope with the limitations of Cloud infrastructure providers (CIPs) for latency-sensitive services. Many Fog infrastructure providers (FIPs) have recently emerged and their number is increasing continually. FLEX is proposed in this work as a platform for selecting a location for service placement in a multi-Fog and multi-Cloud environment. For each service, FLEX broadcasts service requirements to the resource managers (RMs) of the available Fog and Cloud service providers and then selects the most suitable provider for that service. FLEX is scalable and flexible as it leaves it up to the RMs to have their own policy for the placement of submitted services. Service placement and resource selection has been formulated as an optimization problem and an efficient heuristic algorithm is proposed to solve it. Results show that the proposed algorithm can be used across both Cloud and Fog-based providers.

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

S. Azizi (Corresponding Author) and P. Farzin
Department of Computer Engineering and Information Technology, University of Kurdistan, Sanandaj, Iran
E-mail: {s.azizi, p.farzin}@uok.ac.ir

M. Shojafar
5G Innovation Centre, Institute for Communication Systems, University of Surrey, Guildford, United Kingdom
E-mail: m.shojafar@surrey.ac.uk

O. Rana
School of Computer Science and Informatics, Cardiff University, Cardiff, United Kingdom
E-mail: RanaOF@cardiff.ac.uk

1 Introduction

With recent advances in Internet of Things (IoT) technologies, the number of connected devices has increased exponentially [1,50]. These devices generate data that needs to be stored, processed, and analyzed to extract valuable information. To achieve these goals, many applications and services in IoT, big data, and machine learning have recently emerged [49,21,16,17]. The characteristics and requirements of these services vary in the number of resources and Quality of Service (QoS). For example, services such as healthcare systems [35], virtual reality [3], and autonomous and connected cars [30] are time-sensitive. In contrast, big data analysis [36], pollution monitoring [9], and scientific computations [22] may be delay-tolerant.

Cloud computing serves various services and applications by providing a model where you pay as you go. A cloud infrastructure provider (CIP) enjoys a large number of privileges, including hardly limited resources for computation and storage and little cost. Nevertheless, the network bandwidth may be overused, and communication may be delayed extremely [38,29,24,47], since CIPs are centralized by nature, unlike IoT devices, which are decentralized by nature. Such limitations account for the presentation of Fog computing to supplement the Cloud properly [6]. The Fog is aimed mainly at extending Cloud services and resources toward the network edge. More Fog infrastructure providers (FIPs) are likely to become capable of deploying their own infrastructures. A FIP may offer limited storage and computation resources despite its huge advantages to latency-sensitive services. Furthermore, the operation and maintenance of a Fog node (FN) are often costlier, making it less reasonable for an end user [42]. Joint multi-Fog and multi-Cloud environments are turning into environments frequently used to deploy emerging services capable of using both Fog and Cloud providers [23].

The placement of services and applications in Fog computing and Cloud computing has widely appealed in the past years to industrialists as well as researchers [26,33]. The service placement problem (SPP) is complex as it has to specify what services are needed, where end users are located, and how much each infrastructure provider (IP) costs. Due to the incremental trends in the number of IPs and services, the task has become a more serious issue. This makes it necessary to adopt a flexible, scalable platform for selecting a suitable IP for every service from among the available Fog and Cloud IPs.

1.1 Motivation & Context

There have recently been a large number of proposals of solutions to the SPP in the Fog or the Cloud [13,31,25,15,12,5,34,40,44], each concentrating on only one of the two environments rather than considering both at the same time [13, 5,34]. Few of these solutions have assumed integrated Fog and Cloud computing platforms that have been presented by particular IPs, including Google, Amazon, Microsoft, etc. [25,27]. Moreover, most of them fail to consider the cost and communication delay conditions required and intended by a service provider since there are strict requirements for time-sensitive services [12,15,40,44]. We raise the research questions below to attempt to resolve the above problems. i) How can an SPP be addressed in a hybrid multi-Fog and multi-Cloud environment through the design and implementation of an efficient platform? ii) How can the proposed platform identify the most suitable method in a certain context to take advantage of available schedulers, given that an IP would rather implement its own policies to schedule resources? iii) How can the placement algorithm of the proposed platform be used to meet the desired requirements of every service in terms of cost and QoS?

1.2 Contributions of the paper

We propose a flexible and scalable FLEX platform to place services in an environment involving multiple FIPs and CIPs. FLEX delegates the management of resources to the most suitable of the available IPs for each request for service according to requirements. This involves two significant advantages, namely flexibility and scalability. Flexibility means that the platform enables an IP to implement its policies of service placement and adapt them independently. Scalability means the platform is simple enough to support a new IP easily. The key contributions of this work are as follows:

– A flexible and scalable platform for service placement in a multi-Fog and multi-Cloud environment.
– Formulation of the placement problem using integer linear programming (ILP) and a heuristic algorithm to solve it efficiently.
– Using extensive simulations, the performance of the proposed algorithm is evaluated under different conditions.

This work extends our previous conference paper [10] in the following ways:

– Extending & widening the literature review;
– Using a UML sequence diagram to provide a conceptual design and an illustrative example to demonstrate the benefits of the proposed platform and algorithm;
– Providing a motivating example to clarify the anticipated application of the proposed platform;
– Showing that the service placement problem in multi-Fog and multi-Cloud environments is in the class of NP-hard problems;
– Comparing the performance of the proposed algorithm using an evolutionary search algorithm;
– Additional experiments to evaluate the effectiveness of the proposed algorithm under more scenarios;
– Adding the discussion and limitation section to shed light on future research directions.

The rest of this paper is organized as follows. In Section **??**, related works are reviewed and discussed. The high-level and detailed architecture of the proposed platform is described in Section 3. Section 4 presents the integer linear programming model of an SPP. To solve the model, an efficient heuristic algorithm is given in Section 5. Section 6 evaluates the proposed platform using various scenarios. The limitation of our work is discussed in Section 7. Finally, Section 8 concludes the paper, followed by a discussion of future research directions.

2 Related Work

We present a review of the literature on the SPP and proposed frameworks here. In [13], Grozev and Buyya investigate the deployment of applications in a multi-Cloud environment. Their proposed method aims to minimize the end user's delay and total cost by considering auto-scaling, selection of Cloud data centers, and distribution of load. Omer et al. [34] study the placement of IoT applications in a Cloud data center by modeling each application as a series of virtual machines (VMs) depending on one another. They adopt a mixed-integer linear programming (MILP) model to formulate the problem to minimize network consumption, resource wastage, and energy consumption in a Cloud data center. They present a priority-aware heuristic to solve the model with high efficiency. Javed et al. in [19] propose Cloud Market Maker (CMM), a dynamic pricing system for Cloud providers. The CMM adjusts the price of Cloud resources based on current market conditions. CMM also assists customers in the selection of the most suitable provider for their requirements.

Skarlat et al. [41] presented a framework for Fog computing according to the notion of Fog colonies, composed of Fog nodes and cells along with actuators and sensors. The nodes and cells are managed in a Fog colony by a Fog orchestration control node, which sends an IoT application to another colony or Cloud without adequate resources to support it. An extended version of Fog colonies is FogFrame [40], a decentralized framework for managing applications in a Fog landscape.

FOGPLAN, introduced by Yousefpour et al. [48], is a dynamic framework for provisioning IoT services within a Fog computing environment. It is aimed mainly at minimizing overall execution cost and delay violation using two proposed efficient greedy algorithms. Studying how heavily stateful low-latency services (LLAs) are provisioned in the Fog, Tasiopoulos et al. [45] use a mechanism of spot pricing [2] under a framework known as FogSpot to assign cloudlet computation resources to end users given their demands for service. Ghaemi et al. [11] present a serverless platform known as ChainFaaS based on blockchain technology to deliver computing service based on the Internet to end users using the computing capacity of a PC. It is aimed mainly to function as a reliable, transparent platform that decreases the cost for the user.

To satisfy users' QoS needs, Mahmud et al. [25] introduce a method based on edge affinity for application placement on a Fog-Cloud system. In this method, applications are first classified based on the primary features, including IoT device frequency rate, data amount per input, and user-defined deadline. Then, a selection of allowable applications are placed on a Fog cluster to minimize the time required for delivering service. Hassan et al. [15] present an efficient policy for IoT service placement in a Fog-Cloud computing environment. They classify each service as normal or critical to provide FIPs with low energy consumption and IoT users with high QoS. They propose two algorithms: MinEng, to decrease energy consumption in the Fog environment, and MinRes, to minimize response time for critical services. Sami et al. [37] propose an on-demand fog computing architecture based on volunteer devices to deploy IoT services. The Memetic algorithm is presented for solving the container/ service placement in their proposed architecture. Sriraghavendra et al. [43] investigate QoS-aware service placement in a dynamic Fog-Cloud environment. They propose a deadline-oriented service placement (DoSP) strategy which uses a genetic algorithm to host IoT services on suitable Fog and Cloud nodes.

After proposing an ILP model for an SPP, Velasquez et al. [46] present an algorithm based on PageRank to use the popularity of applications for ranking them. The research is aimed mainly at decreasing latency for popular applications. Natesha and Guddeti [32] attempt to provision resources in the Fog by designing a framework with two levels. They formulate the SPP as an optimization problem with multiple minimization objectives for cost, energy consumption, and service time and use an elitism-based genetic algorithm (EGA) to solve it. Iyer et al.[18] propose a broker-based architecture to connect users to different Fog and Cloud service providers. Users submit their tasks to a broker and then the broker gets price offers from service providers. The users then apply a utility function to choose a particular provider for task execution. The utility function takes into account cost, risk and trust values to select the best provider. Cao et al. [7] present an integrated model for provisioning resources, known as Edge federation, to use multiple Edge infrastructure providers (EIPs) for placing latency-critical services. They formulate the process of provisioning as a linear program (LP) aimed to minimize resource cost and ensure service latency. Moreover, they extend their model in the Edge federation environment by proposing a dynamic service provisioning solution.

Although there exist a number of mechanisms for service placement in Fog and/or Cloud computing, none have focused on the following features collectively: (i) considering the service placement problem on joint multi-Fog and multi-Cloud environments, (ii) scalablity and flexibility by leaving it up to Fog and Cloud providers to have their own policy for service placement, (iii) automated selection of a provider by the platform and not the user, and (iv) taking into account the service preference of both delay and cost.

3 FLEX Platform

In this section, a motivating scenarios is presented followed by a conceptual architecture of the FLEX platform. How FLEX achieves flexibility and scalability is also described, through a description of components that are used to implement FLEX.

3.1 Motivation Example

A practical use case scenario is used to illustrate how the FLEX platform can be used. Consider a provider offering service S_x to a set of IoT users in a smart city. The service provider searches for a suitable infrastructure provider to host the service. It must check each available Fog/ Cloud IP before making a decision, which is a time-consuming and error prone process. For example, assume that the service provider decides to submit S_x to infrastructure provider IP_y as it is a well-known provider. However, there exists another infrastructure provider, IP_z, which is a new provider and is not selected by the service provider because the service provider does not have sufficient knowledge of its reputation and cost model. This may leads to an increase in cost and a decrease in the QoS for the S_x's users.

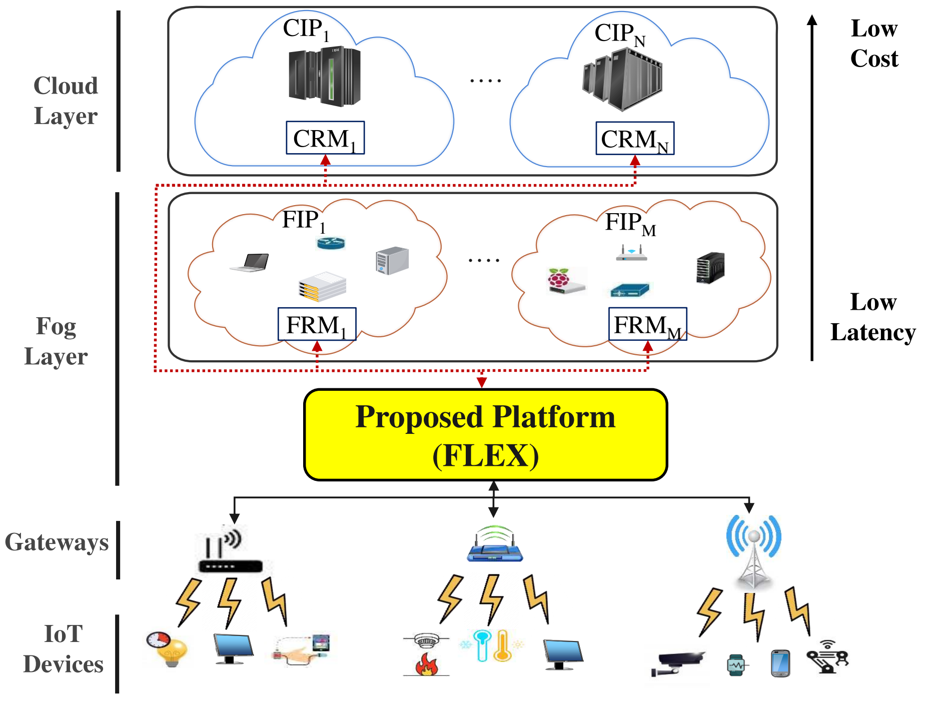


Fig. 1: FLEX architecture.

Let us now look at the scenario from the flexibility and scalability perspective. Assume infrastructure providers IP_a and IP_b have some available computing nodes to host service S_x. There are mainly *two* approaches to do this. First, the aforementioned infrastructure providers register their available resources in the marketplace and the system selects the most suitable node and assign the service S_x on that node. In this case, the system act as a centralized resource scheduler. Second, the system submits the service requirements and users' locations to infrastructure providers IP_a and IP_b, delegating control over the resource scheduling to them. In this case, each IP can run its own service placement algorithm, allowing the system to be highly scalable and providing flexibility to infrastructure providers. The main idea of the proposed platform is based on the second approach.

3.2 High-level Architecture

A general architecture of the multi-Fog and multi-Cloud environment is shown in Fig. 1. Using the proposed platform to augment the IoT-Fog-Cloud pattern with one more layer, we end up with an environment composed of five parts, including the *IoT devices*, *gateways*, *FIPs*, *CIPs*, and the *proposed platform*, as detailed in the following subsections.

– **IoT devices:** These are end-point devices including sensors, actuators, radio frequency identification devices, industry devices, wearable devices, cameras,

smartphones, smart meters, and smart home appliances. They are distributed around the world with Internet connection via gateways with various wireless technologies including 3G/4G/5G, Bluetooth, Wi-Fi, and ZigBee. These devices can hardly host latency-sensitive services or computing-intensive ones due to their limited battery power, storage, and processing resources.
– **Gateways:** These include home switches, cellular base stations, Wi-Fi access points, and similar Edge devices located in the vicinity of an IoT device.
– **FIPs:** There are several FIPs in a multi-Fog environment each of which provides the resources required for networking, computing, and storage. Called FNs, Fog layer devices include cloudlets, servers, PCs, switches, routers, micro data centers, and Raspberry Pies. They are often resource-richer than Edge devices. There is a specialized node in every FIP that is dedicated to resource management and establishment of a persistent communication with the proposed platform. Known as a Fog resource manager (FRM) [25], this node can host services and applications as containers or VMs.
– **CIPs:** These are located in the top layer of the vertical dimension to use their robust, large-scale data centers to provide a large number of services. A series of virtualized storage and computing resources make up a cloud data center. As with FIPs, there is a node known as the Cloud resource manager (CRM) that allows each CIP to communicate with the proposed platform as well as managing its resources. Although normally more cost-effective than an FIP, a CIP is unsuitable for time-sensitive services, as it is often far from IoT devices.
– **Proposed platform (FLEX):** The major design philosophy behind FLEX is to enable efficient distribution of services and applications among FIPs and CIPs to provide flexibility and scalability in a multi-Fog and multi-Cloud computing environment. FLEX places services after receiving them along with predefined requirements from their various end-users and providers. For this purpose, it broadcasts the requirements and user locations of a submitted service or batch of services to the resource managers (RMs) of all Fog and Cloud IPs. It then chooses the most suitable service provider in each case according to the cost and communication delay that IPs offer. It should be noted that FLEX allows the IP resource manager to consider its own policy for service placement on available nodes.

3.3 Detailed Architecture

Fig. 2 shows the components of FLEX: *Service Receiver*, *Service Analyzer*, *Admission Control*, and *Provider Selector*. Details on their functionality are explained below.

– *Service Receiver:* It functions as an interface for submission of provided services. The requirements of every service, known as the service profile, are specified in this step. These include the privacy and security concerns, cost and delay priority, and bandwidth, computing, and storage resource amount. In a latency-sensitive service, for instance, delay needs to be given a far greater weight than cost.
– *Service Analyzer:* It analyzes every service and uses the service profile to store it in the database of services. Services are classified and sorted here according to a number of important parameters including the requirement for resource, level of sensitivity to privacy and security, and latency-sensitivity degree.

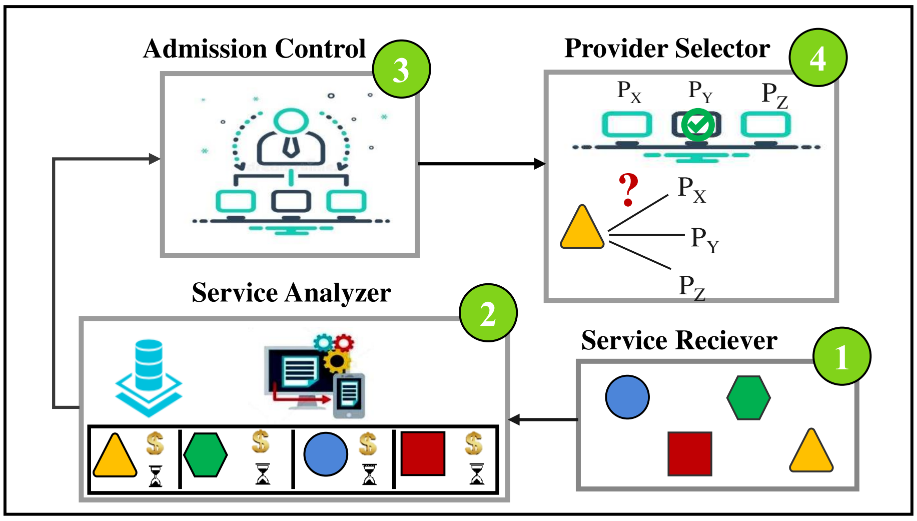


Fig. 2: The details of FLEX architecture.

- *Admission Control:* It broadcasts requests for service received from the service analyzer to the Fog and Cloud RM of each Fog and Cloud IP, a stage known as the process of matchmaking. The RM examines whether the requirements of each service in each IP can be met. In that case, it obtains cost by estimating the communication delay between the user's location and the node considered for service placement. The RM then includes cost and delay information in response to admission control, which receives responses from all providers and then sends the list of providers capable of hosting the service to provider selector along with relevant information.
- *Provider Selector:* It chooses the most appropriate provider for placing each service according to the list received from admission control. Various methods of multi-criteria decision-making (MCDM) are used to carry out the process of provider selection. We propose an efficient heuristic algorithm in Section V and use it in provider selector. Once an IP is chosen, information is provided to the IP and service users. The corresponding gateways then store the placed node address so that the requests for service are redirected to that node.

It should be mentioned that the cross-IP interactions between FRMs and CRMs and admission control occur only once, immediately after the request for placement of service. There will be direct communication between the chosen IP and the end user once served.

The process of placing a service on a multi-Fog and multi-Cloud environment is illustrated in Fig. 3. The process is initiated when some IoT devices send a service request, say S_x, to FLEX in order to find an appropriate provider for hosting it (step 1). Once FLEX receives the request, it estimates its resource requirement (e.g., CPU, memory, etc) and the importance of delay and cost for that service, and then forwards the service's resource requirement to all available FIPs and CIPs (step 2). After receiving the request, each FIP and CIP checks whether it has enough resources to host the service or not (step 3). If a provider does not have sufficient resources, it rejects the request and informs FLEX (step 4). Otherwise, it calculates the delay and cost for the service S_x and sends the related values to FLEX (steps 5-6). Based on the values, FLEX obtains the suitability of each provider in order to decide which one is the best to host the service. Following this, the selected provider is informed by FLEX (step 7) and it sends the address of hosted node, say $F_{j,l}$ to FLEX (step 8). Next, FLEX publishes this address to all of the IoT devices that have requested for this service (step 9). Now, the set of IoT devices I can directly communicate with the hosted node $F_{j,l}$ and submit their offloaded tasks to this node (step 10). The hosted node executes the tasks and returns the results (steps 11-12).

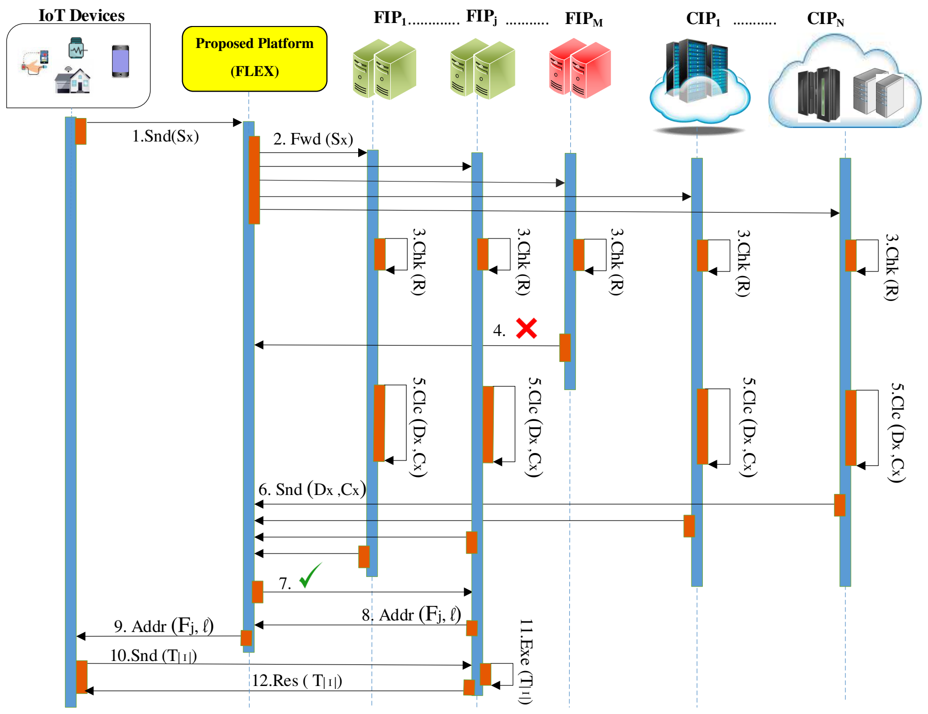


Fig. 3: Sequence diagram.

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 $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ denotes the set of n services in which each service $\mathcal{S}_i$ has some specific characteristics. The resource requirement of service $\mathcal{S}_i$ is represented as $\mathcal{S}_i^r$ where r is belong to R={CPU, memory, bandwidth, storage}. Also, $\mathcal{S}_i^s$ is used to indicate the size of the service $\mathcal{S}_i$ (in terms of million instructions – MI). For each service $\mathcal{S}_i$, a coefficient $\alpha_i \in [0, 1]$ 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. We use $\mathcal{F}_{j,l}^r$ to show the resource capacity of the l-th FN of the FIP $\mathcal{F}_j$ along different $r \in R$ dimensions. Similarly, let $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N\}$ is the set of N CIPs where each provider $\mathcal{C}_k$ has $|\mathcal{C}_k|$ cloud nodes (CNs). To denote the resource capacity of the l-th CN of the CIP $\mathcal{C}_k$ along the dimension r the symbol $\mathcal{C}_{k,l}^r$ is used. $c(\mathcal{F}_j^r)$ and $c(\mathcal{C}_k^r)$ are respectively used to define the resource cost of the FIP $\mathcal{F}_j$ and CIP $\mathcal{C}_k$ along each dimension. It is worth mentioning that FIPs and CIPs provide their resources in the form of VMs and/or containers to host IoT services.

4.2 Decision Variables

Decision variables $x_{j,l}^i$ and $y_{k,l}^i$ are two main variables of Our model which are defined as below.

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

and

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

4.3 Delay

In this subsection, the delay model for a service service placement strategy in the proposed system is presented. To calculate the delay of a request for the i-th service $\mathcal{S}_i$, denoted by D_i, the following delay factors should be taken into account in the model.

– **Communication time (c_i):** This is the amount of time a request gets to reach from an IoT device $\mathcal{I}_z$ to the Fog or Cloud node that the i-th service $\mathcal{S}_i$ is hosted on it.

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

where $D\left(\mathcal{I}_z \rightarrow \mathcal{F}_{j,l}\right)$ and $D\left(\mathcal{I}_z \rightarrow \mathcal{C}_{k,l}\right)$ are the communication delay from IoT device $\mathcal{I}_z$ to the l-th FN of the FIP $\mathcal{F}_j$ and l-th CN of the CIP $\mathcal{C}_k$, respectively. Note that if more than one IoT device requests a service, the average of their communication time is taken into account.

– **Execution time (e_i):** This is the amount of time required to process the service request. Thus, we have

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

where $\mathbf{S}_i^{CPU}$ is the CPU requirement of the i-th service $\mathbf{S}_i$ (in terms of million instruction per second - MIPS). Hence, the following equation can be used to obtain the delay.

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

We use the following equation to obtain the total weighted delay for a service placement strategy.

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

4.4 Cost

The cost of a service depends on the provider selected for hosting that service. The following equation can be used to calculate the cost of service $\mathbf{S}_i$.

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

Therefore, the total weighted cost for all of n services can be obtained as follows.

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

4.5 The Objective Function

The main goal of the proposed FLEX is to solve the problem of service placement in multi-Fog and multi-Cloud environments in a way that the total weighted delay and cost of the services is simultaneously minimized. Hence, the final objective function can be expressed as follows. Let $\mathbb{S} = \{1, 2, \ldots, \max(|\mathcal{F}_j|, |\mathcal{C}_k|)\}$

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

Subject to the following constraints:

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

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

$$\sum_{i=1}^{n} \mathcal{S}_i^r \times y_{k,l}^i \leq \mathcal{C}_{k,l}^r, \ \forall k \in \mathcal{C}, \forall r \in R, \forall l \in \{1, 2, \ldots, |\mathcal{C}_k|\} \tag{12}$$

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

where constraint (10) ensures that each service can be hosted only to one computing node. Constraints (11) and (12) respectively guarantee that resource demand of services must not exceed from the capacity of each Fog and Cloud node. Finally, the domain of variables are specified by the constraint (13).

Theorem 1 *SPP in multi-Fog and multi-Cloud environments is in the class of NP-hard problems.*

Proof. The SPP is a kind of a weighted bin-packing problem. In this case, a service represents an *item*, and the FIPs/CIPs are represented *bins*. In weighted bin-packing problem, the aim is to select the minimum weighted bin among the weighted bins such that to minimize the cost of the used item. Similarly, in SPP, our aim is to find the best FIPS/CIPs that be allocated to the services and help to minimize cost and delay of the service (i.e., item). Since the weighted bin packing problem is NP-hard [14], our SPP is also NP-hard.

5 FLEX Heuristic Algorithm

For efficient solution of the SPP, we propose and describe a new heuristic here, which is aimed mainly to minimize service delay and cost at the same time. For that purpose, providers are ranked in the algorithm according to the delay and cost preferences of the service. Therefore, we refer to the proposed heuristic as *minimum cost and delay first (MCD1)*, where a provider is chosen if it minimizes weighted delay and cost. The proposed heuristic is detailed below.

5.1 Proposed algorithm

Let us assume that we submit a list of n services to the proposed algorithm in order to obtain an appropriate provider for each. The algorithm first considers the sensitivity of the services to delay to sort them, prioritizing those with higher α values. Admission control then broadcasts the requirements of every given service and the user locations to all Fog and Cloud IPs, as detailed in Subsection 3.2. Next, the algorithm waits for all Fog and Cloud RMs that are contacted to respond to admission control. The responses include three values for each service: (i) a Boolean value that shows the capability of the provider of hosting the service, (ii) the service monetary cost, and (iii) an estimate of the delay between the service placement candidate and user gateway. For each service, the candidate provider list is sent by admission control according to the Boolean value to provider selector, where *MCD1* is run once the list is received, so that the most appropriate provider is chosen for each service.

The MCD1 pseudocode is shown in Algorithm 1. $\mathcal{S}_i$ represents the service received by the algorithm, $\mathbf{P}$ indicates the list of candidate Fog and Cloud IPs, $\mathbf{C}$ and $\mathbf{D}$ denote the cost and delay vectors for service $\mathcal{S}_i$, presenting the most appropriate provider according to the service preferences. Let C_{ij} and D_{ij} be the cost and delay of provider P_j for service $\mathcal{S}_i$, as can be seen in lines 1 and 2. The providers with maximal cost and delay are found in lines 3 and 4. Lines 5 to 9 involve a **loop** that aims at scoring providers according to the delay and cost reported about them and the preferences for service $\mathcal{S}_i$. For that purpose, we obtain the objective function for each of the providers according to the predefined weight of service $\mathcal{S}_i$ assigned to cost and delay after normalizing them in lines 6 and 7. After that, the algorithm generates the vector for the objective function, as can be seen in line 10. In lines 11 to 13, the provider that exhibits the lowest value for the objective function is finally found and chosen at the destination provider to host service $\mathcal{S}_i$.

Algorithm 1 MCD1 Algorithm

INPUT: $\mathcal{S}_i$, **P**:list of available 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 service $\mathcal{S}_i$

```
 1: Let $C_{ij} \in \mathbf{C}$ is the monetary cost of provider $P_j$ for $\mathcal{S}_i$;
 2: Let $D_{ij} \in \mathbf{D}$ is the delay of provider $P_j$ for $\mathcal{S}_i$;
 3: $C_j^{max} \leftarrow$ find maximum $C_{ij} \in \mathbf{C}$;
 4: $D_j^{max} \leftarrow$ find maximum $D_{ij} \in \mathbf{D}$;
 5: for Each $P_j \in \mathbf{P}$ 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 $\mathbf{F}$ is objective function vector for $\mathcal{S}_i$;
11: $F_j^{min} \leftarrow$ find minimum $F_{ij} \in \mathbf{F}$;
12: Let $P_{index}$ is the provider with the value of $F_j^{min}$;
13: return $P_{index}$ as the destination provider for hosting $\mathcal{S}_i$;
```

5.2 Time complexity

Here the time complexity of the proposed algorithm is analyzed. The complexity of lines 3 and 4 is $\mathcal{O}(M+N)$, where M and N are the number of available FIPs and CIPs, respectively. Note that $|\mathbf{P}|= M+N$. Lines 5-9, i.e., the normalization step, also requires $\mathcal{O}(M+N)$. Again, the time complexity of line 11 is $\mathcal{O}(M+N)$. Therefore, the overall time complexity of the proposed MCD1 for one service is $\mathcal{O}(|\mathbf{P}|)$, i.e., $\mathcal{O}(M+N)$.

5.3 An illustrative example

Now, we give an illustrative example to show how our proposed platform works (see Fig. 4). Let FLEX receives two requests for service S_x from two different IoT devices. Based on the requests' characteristics, FLEX estimates the resource requirements of that service, e.x., $S_x^{CPU} = 600$ MIPS, $S_x^{mem} = 1024$ MB, and

$S_x^s = 1000$ MI plus the coefficient $\alpha_x = 0.8$. Now, FLEX sends this service request to the resource managers of two FIPs, say FIP_1 and FIP_2, and two CIPs, say CIP_1 and CIP_2. Then, each provider calculates the delay and cost for the service S_x and informs FLEX about the results. FLEX gathers the responses and creates a table like Table 1. Finally, it obtain the objective function for each provider (see eq.16) and selects the one with the minimum value. Here FIP_2 is selected and service S_x is hosted on it.

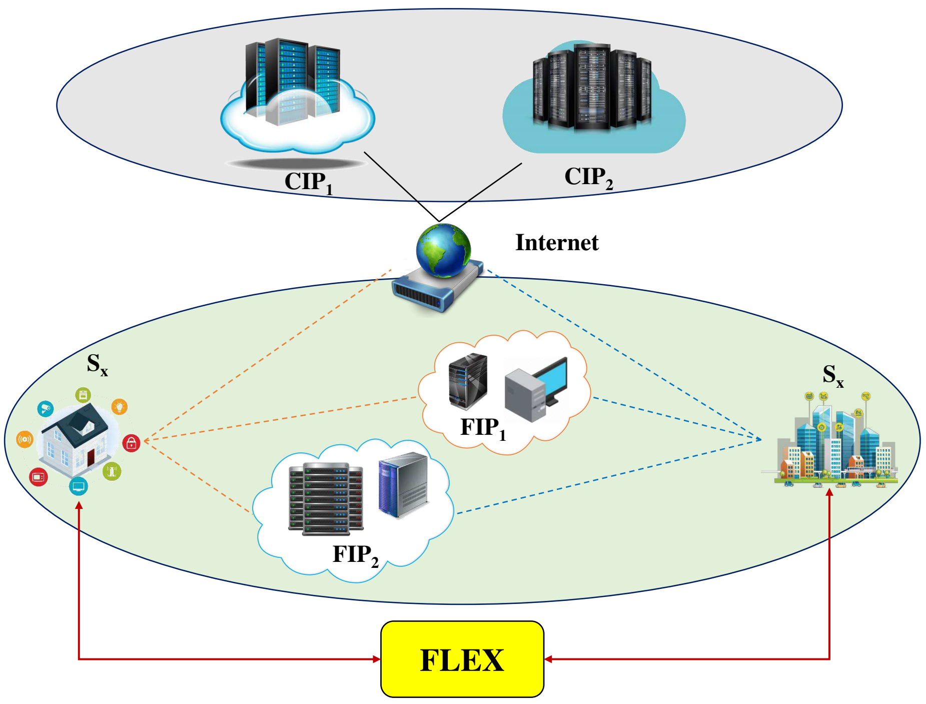


Fig. 4: An illustrative example.

Table 1: Information gathered from providers. OF:= Objective Function; FIP:= Fog infrastructure provider; CIP:= Cloud infrastructure provider.

Providers	**Delay [ms]**	**Cost [G$ per hour]**	**OF**
FIP_1	1696.7	301.4	0.45
FIP_2	1676.8	306.5	0.44
CIP_1	212.3	241.0	0.50
CIP_2	2668.5	191.9	0.59

6 Performance Evaluation

To evaluate the performance of the FLEX platform, we conducted comprehensive experiments. To this end, we have implemented FLEX using our 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. To achieve results with high statistical confidence, we repeated each experiment 10 times.

6.1 Simulation settings

For a full understanding of the merits of FLEX and the relevant heuristic, we demonstrated the effects of a number of scenarios considering four experiments (Table 2). In all the experiments, we set the ratio of latency-sensitive services, greatly emphasizing delay, to all services to 25% and the ratio of FIPs to all providers to 75%. The latency-sensitive services put a lot of emphasis on delay, e.g., $\alpha_i \geq 0.9$.

Table 2: 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%, 50%, 75%)	(25%, 50%, 75%)
2	100	20	(25%, 50%, 75%)	(25%, 50%, 75%)
3	500	8	(25%, 50%, 75%)	(25%, 50%, 75%)
4	500	20	(25%, 50%, 75%)	(25%, 50%, 75%)

We employed a synthetic dataset in the experiments due to the unavailability of the real one for simulation of services, Fog and Cloud providers, and other aspects of the environment. The attributes of services are shown in Table 3, and those of Fog and Cloud nodes appear in Table 4. While we imposed no restriction on the number of CNs in each CIP, we considered the limited value for the number of FNs for an FIP, i.e., a value within the [6, 12] range.

Table 3: Attributes of Services.

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

6.2 Simulation metrics

To evaluate the performance of the proposed heuristic algorithm for the FLEX framework, we use the following metrics in our experiments.

Table 4: 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)

- **Average Weighted Delay (AWD):** The following equation is used to measure the delay given by a service placement strategy to host n submitted services on the considered 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):** To evaluate the effectiveness of a service placement policy in terms of monetary cost, we use the following metric.

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

- **Objective Function (OF):** To measure the performance of a service placement strategy in both of delay and cost perspective, we define the following metric.

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

  where $\mathcal{D}_i^{max}$ and $\mathcal{C}_i^{max}$ denote the maximum possible delay and monetary cost which are provided for i-th service $\mathcal{S}_i$, respectively. It is worth mentioning that the decreased value of this metric represents the enhanced performance of a placement strategy 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.

- **The Most Cost-effective Provider First (MC1):** For each service, MC1 selects the provider with the minimum cost for that service.
- **The Minimum Delay Provider First (MD1):** This algorithm selects the provider which offers the minimum delay for each service placement request.
- **Genetic Algorithm (GA):** GA is one of the most used meta-heuristic techniques for the SPP in Fog and Cloud environments, e.g., [39, 28, 32, 43]. Here, we have implemented our custom GA according to the OF provided in eq. 16. The initial population and the number of iterations are set to 100 and 500, respectively. For parent selection, the roulette wheel technique and uniform probability are respectively applied for the first and second parents. Regarding crossover, single-point operation is used. Each chromosome participates in one-point mutation with the rate of 10%. The location of the mutant gene is selected randomly and replaced by a different value.

6.4 Results

In this section, we present and discuss the results of the four considered experiments. Tables 5 through 8 respectively show the simulation results for experiments *one* to *four* in terms of AWD and AWC. Also, Figures 5 through 8 demonstrate the box plot of OF values for experiments *one* to *four* respectively. We make some important observations from the obtained results as follows.

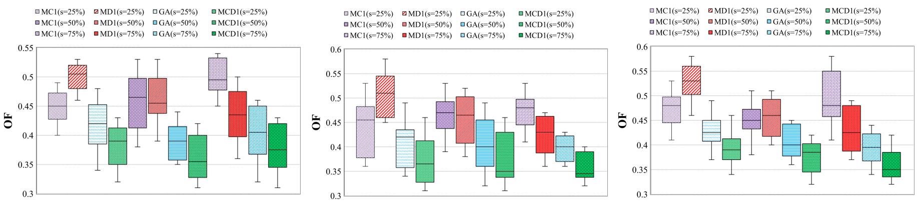
6.4.1 Experiment one

It can be seen in Table 5 that AWD is significantly reduced for all the policies as FIP rate rises from $f = 25\%$ to $f = 75\%$, while the reverse is true of AWC, which is in line with expectations because an FIP often exhibits greater cost but less delay than a CIP. Moreover, GA and MCD1 meet the end user's QoS needs by hosting more services on FIPs as latency-sensitive service rate rises from $s = 25\%$ to $s = 75\%$. Whereas AWC rises, therefore, AWD decreases. MCD1 obtains far greater AWC and AWD than GA in all cases, however. Furthermore, the smallest value of AWD (AWC) is provided by MD1 (MC1), which exhibits the largest value of AWC (AWD) in virtually all cases, in line with expectations.

From Fig. 5a to Fig. 5c, 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. Here, the improvement is up to 13.7% in comparison with the second-best approach, i.e., GA.

Table 5: AWD and AWC results of `experiment one` for #Service=100, #Providers=8.

Comparing Algorithms	**Ratio**	**AWD (ms)**			**AWC (G$)**		
		f=25%	f=50%	f=75%	f=25%	f=50%	f=75%
MC1	s=25%	1884	1839	1786.3	173.6	198.2	196.3
		(std=132.3)	(std=89.5)	(std=81.8)	(std=23.3)	(std=39.3)	(std=33.4)
	s=50%	1903	1842.5	1768	168.2	200	216.2
		(std=285.3)	(std=132.5)	(std=133.1)	(std=23)	(std=40.8)	(std=38.1)
	s=75%	2004.7	1836.3	1729.5	191.2	211.3	223.2
		(std=260)	(std=92.2)	(std=128.1)	(std=34.9)	(std=41.1)	(std=31.3)
MD1	s=25%	1307.5	1299.2	1254.6	304	357.7	341.6
		(std=309.3)	(std=120.1)	(std=145.8)	(std=46.2)	(std=36)	(std=44.8)
	s=50%	1356.3	1250.8	1249.6	329.7	366.2	328.8
		(std=203.3)	(std=72.5)	(std=257.1)	(std=48.9)	(std=33.6)	(std=30.7)
	s=75%	1338.8	1246.9	1188.4	346.6	345	351.6
		(std=213.7)	(std=129.3)	(std=200.9)	(std=44)	(std=28.6)	(std=28.4)
GA	s=25%	1788.3	1691.8	1700.5	228.4	238	263.9
		(std=99)	(std=96.3)	(std=81.5)	(std=35.3)	(std=39.4)	(std=43.5)
	s=50%	1664.3	1618.8	1590.6	243.7	272.2	283.5
		(std=172.3)	(std=162)	(std=192.1)	(std=46.6)	(std=44.4)	(std=49.8)
	s=75%	1568.2	1484.9	1472	285.5	275	310.6
		(std=205.5)	(std=110.2)	(std=92.4)	(std=60.8)	(std=43.2)	(std=34.1)
MCD1	s=25%	1691.5	1618	1643.3	207.2	226.8	252.3
		(std=103.3)	(std=101.1)	(std=108.1)	(std=29.8)	(std=38.2)	(std=32.6)
	s=50%	1576.3	1545.8	1498.6	234.7	254	269.1
		(std=156.3)	(std=178.5)	(std=184.1)	(std=54)	(std=33.6)	(std=42.8)
	s=75%	1497.4	1409.2	1379.5	259.8	263.3	290.1
		(std=229.4)	(std=126.1)	(std=172.6)	(std=42.2)	(std=33.1)	(std=45.3)

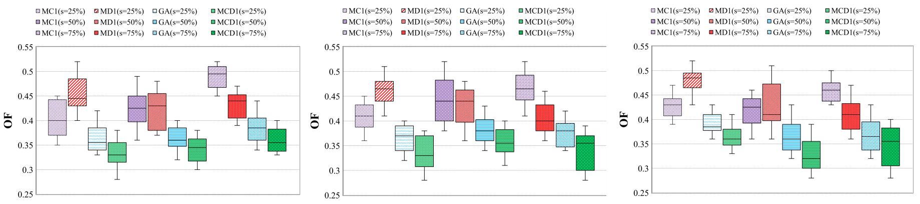


(a) f=25% (b) f=50% (c) f=75%

Fig. 5: Simulation results for `experiment one` with #Service=100, #Providers=8.

6.4.2 Experiment two

Table 6 demonstrates the performance of the algorithms in terms of the AWD and AWC. There are two important observations. First, as the rate of latency-sensitive services increases, the AWD and AWC of MC1 and MD1 do not change much. However, these values remarkably change for MCD1 and GA. The reason behind this is that MC1 (MD1) always tries to host services on CIPs (FIPs) as far as possible. In MCD1 and GA when the rate of latency-sensitive services is low, i.e., $s = 25\%$, most of services are placed on CIPs which lead to higher delay and lower cost. By increasing the vaule of s, FIPs are selected for most of services which results in lower delay at higher cost. Second, although MC1 (MD1) provides the best AWC (AWD), it suffers from the worst AWD (AWC). However, the proposed MCD1 gives the second-best performance both in terms of cost and delay. The OF results for Experiment two have been displayed in Fig. 6. As can be seen from the figure, the proposed MCD1 achieves better trade-off between the AWD and AWC than the other strategies and significantly performs better than the others in all cases. In particular, the proposed policy can reduce the OF value by 26.8%, 16.1%, and 8.8% compared to MC1, MD1, and GA, respectively, for the case s=75% and f=50%.



(a) f=25% (b) f=50% (c) f=75%

Fig. 6: Simulation results for `experiment two` with #Service=100, #Providers=20.

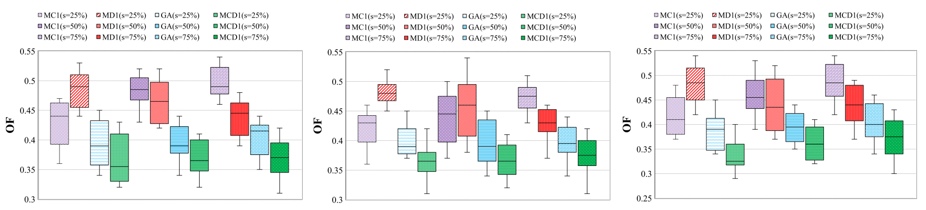
6.4.3 Experiment three

Table 7 and Fig. 7 show the results of simulation for experiment three. Overall, the algorithms behave almost similarly to those in experiment one. AWD generally rises as more services are provide, which makes sense because a larger number of

Table 6: AWD and AWC results of `experiment two` for #Service=100, #Providers=20.

Comparing Algorithms	**Ratio**	**AWD (ms)**			**AWC (G$)**		
		f=25%	f=50%	f=75%	f=25%	f=50%	f=75%
MC1	s=25%	1793.5 (std=118.5)	1752.3 (std=99.6)	1774.8 (std=96.1)	174.6 (std=20.9)	182.5 (std=22)	180 (std=10.3)
	s=50%	1784 (std=106.4)	1831.1 (std=122.2)	1789.3 (std=134.1)	172.7 (std=8.5)	173 (std=12.6)	185.1 (std=11.3)
	s=75%	1789.8 (std=153.5)	1825.1 (std=117.8)	1802.6 (std=87.1)	171.3 (std=14)	172.4 (std=12.9)	180 (std=13.7)
MD1	s=25%	1311.4 (std=265)	1225.1 (std=91.4)	1193 (std=290.8)	323.3 (std=34.3)	354.6 (std=26.5)	345.4 (std=37.4)
	s=50%	1333.5 (std=137.8)	1254.3 (std=100)	1170 (std=225)	347.5 (std=27.8)	357.2 (std=41.3)	349 (std=38.7)
	s=75%	1275.4 (std=148.8)	1224.8 (std=121.3)	1223.3 (std=202.1)	351.1 (std=32.3)	355.8 (std=40.2)	350.3 (std=38.8)
GA	s=25%	1656.7 (std=112.2)	1647.4 (std=99.9)	1714.4 (std=107.5)	223.3 (std=17.3)	229.9 (std=34.3)	223.6 (std=14.8)
	s=50%	1630.8 (std=154.5)	1531.3 (std=194.9)	1642.3 (std=179.5)	242 (std=40.2)	265.2 (std=73.7)	258 (std=44.8)
	s=75%	1500.4 (std=170.5)	1447.1 (std=97.7)	1486.3 (std=97.1)	275.7 (std=40.8)	293.6 (std=49.9)	301.6 (std=45.5)
MCD1	s=25%	1613.9 (std=100.9)	1599.8 (std=114)	1661.6 (std=121.1)	204.3 (std=8.6)	220.7 (std=32.1)	210 (std=13.3)
	s=50%	1557.5 (std=161.3)	1455.4 (std=189.7)	1575.1 (std=171.8)	224.7 (std=31.9)	255 (std=54.6)	242.8 (std=36.5)
	s=75%	1461.5 (std=168.7)	1384.5 (std=94)	1398.5 (std=91.1)	241.2 (std=30.2)	286.3 (std=44.1)	277.1 (std=439.4)

services should be placed on CIPs in that case. As the rates of FIPs and latency-sensitive services both rise, significantly greater values of AWD are exhibited by MD1 and the proposed algorithm than by MC1 and GA. For AWC, however, the rise is far less in the case of MCD1 than for MD1, since the former algorithm prefers a CIP for a latency-tolerant service despite the simultaneous availability of FIPs.



(a) f=25% (b) f=50% (c) f=75%

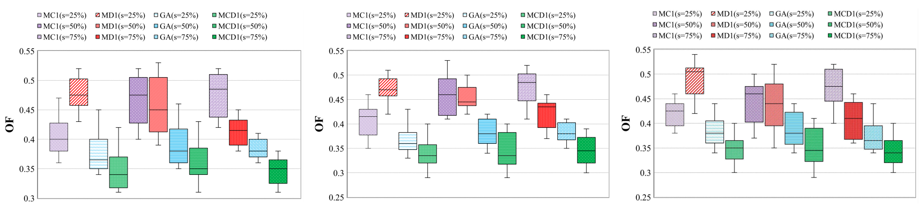
Fig. 7: Simulation results for `experiment three` with #Service=500, #Providers=8.

Table 7: AWD and AWC results of `experiment three` for #Service=500, #Providers=8.

Comparing Algorithms	Ratio	AWD (ms)			AWC (G$)		
		f=25%	f=50%	f=75%	f=25%	f=50%	f=75%
MC1	s=25%	1830.7	1727.7	1757.6	179.2	197.1	206.6
		(std=175.1)	(std=117.2)	(std=103.1)	(std=15.6)	(std=37.8)	(std=30.8)
	s=50%	1814.2	1783.7	1703.9	184	189.9	207.2
		(std=109.3)	(std=91.2)	(std=100.6)	(std=17.8)	(std=21.9)	(std=47.3)
	s=75%	1786.6	1771.5	1656.1	193.4	195.7	219.4
		(std=72.2)	(std=78.6)	(std=112.1)	(std=19)	(std=24.3)	(std=37.8)
MD1	s=25%	1400.9	1306.8	1155.3	344.5	359.9	367.4
		(std=63.7)	(std=48.4)	(std=157.9)	(std=45.5)	(std=31)	(std=37.1)
	s=50%	1370.2	1313.8	1183.3	338.2	380.8	371
		(std=61.5)	(std=122.2)	(std=120.7)	(std=33.8)	(std=32)	(std=33.7)
	s=75%	1361.2	1293.2	1234.8	353.6	344.8	371.6
		(std=101.2)	(std=172.3)	(std=126)	(std=36.8)	(std=27)	(std=37.1)
GA	s=25%	1689	1656.2	1706.2	239.3	256.9	269.1
		(std=113.9)	(std=78.8)	(std=69.9)	(std=33)	(std=32.5)	(std=35.4)
	s=50%	1655.4	1641.6	1504	249.1	265	260.7
		(std=148.4)	(std=110.4)	(std=157.7)	(std=31.9)	(std=34.3)	(std=41.1)
	s=75%	1564.5	1537	1450.7	274.6	279.3	282.9
		(std=112.1)	(std=147.7)	(std=138.5)	(std=34.3)	(std=35.6)	(std=36.9)
MCD1	s=25%	1624.1	1581	1649.7	215.3	238.4	240.8
		(std=92.3)	(std=85.6)	(std=86.8)	(std=24.9)	(std=30.7)	(std=30.6)
	s=50%	1578.5	1573.8	1434	230.8	249.4	244.6
		(std=132.5)	(std=139.3)	(std=174.8)	(std=22.6)	(std=26.8)	(std=45.1)
	s=75%	1492.1	1443.9	1345.6	246.5	249.3	265.6
		(std=110.7)	(std=135.1)	(std=134.3)	(std=22.1)	(std=38.7)	(std=38.4)

6.4.4 Experiment four

Table 8 and Figs. 8a to 8c show the results of experiment four. From the table and figures, we can again observe that by considering both of the AWD and AWC, the proposed MCD1 shows the best performance in all cases.



(a) f=25% (b) f=50% (c) f=75%

Fig. 8: Simulation results for `experiment one` with #Service=500, #Providers=20.

7 Discussion

In the following, we discuss the limitations of FLEX and give some research directions in the domain of service placement in multi-Fog and multi-Cloud environments.

Table 8: AWD and AWC results of `experiment four` for #Service=500, #Providers=20.

Comparing Algorithms	**Ratio**	**AWD (ms)**			**AWC (G$)**		
		f=25%	f=50%	f=75%	f=25%	f=50%	f=75%
MC1	s=25%	2391.7	2017.3	1803.4	167.2	187.9	193.2
		(std=552.2)	(std=285.6)	(std=60.3)	(std=20.2)	(std=19.9)	(std=20.8)
	s=50%	2110.1	1952.5	1789.1	173.4	171.4	184.6
		(std=470.1)	(std=274.3)	(std=70)	(std=15)	(std=8.6)	(std=13.4)
	s=75%	2201	2047.3	1805.3	166.1	176.7	186.3
		(std=432.7)	(std=347.3)	(std=83)	(std=16.9)	(std=14.1)	(std=21.8)
MD1	s=25%	1215	1143.6	1054.3	361.9	387.6	373.7
		(std=162.4)	(std=148.2)	(std=232.2)	(std=38.1)	(std=37)	(std=39.1)
	s=50%	1222.8	1165.6	1086.7	349	381.2	379.7
		(std=172.2)	(std=197.9)	(std=258.7)	(std=45.4)	(std=45.6)	(std=23)
	s=75%	1242.4	1160.6	1057.5	358.1	359.5	350.4
		(std=153.6)	(std=126.2)	(std=244)	(std=38.2)	(std=24)	(std=29.6)
GA	s=25%	2196.7	1829.4	1700.1	204.6	221.3	241.3
		(std=515.6)	(std=179.1)	(std=62.9)	(std=27.1)	(std=30.1)	(std=34.8)
	s=50%	1680.3	1592.9	1555.8	264.4	279.5	273.5
		(std=436.3)	(std=229.9)	(std=210.7)	(std=58.1)	(std=53.8)	(std=52.2)
	s=75%	1435.9	1381.1	1345.7	282.3	287.2	294.2
		(std=113.2)	(std=93.8)	(std=189.5)	(std=35.5)	(std=35.8)	(std=30.4)
MCD1	s=25%	2128.7	1742	1624.6	190.6	201.2	219.5
		(std=519.2)	(std=131.4)	(std=83.4)	(std=23.3)	(std=17.8)	(std=21.6)
	s=50%	1562.3	1520.2	1480	246.4	232.7	251.6
		(std=441.9)	(std=219.7)	(std=241.8)	(std=46.4)	(std=47.2)	(std=33.6)
	s=75%	1354	1309.8	1225.8	265	252.4	277.1
		(std=141.5)	(std=78.4)	(std=226.3)	(std=31.1)	(std=42.1)	(std=39.7)

One of the main responsibilities of FLEX is the estimation of a service's resource requirements. However, this is not a trivial task and needs careful analysis of the set of IoT requests for that service. The dynamicity of requests such as their frequency rate, priority of delay and cost, and security and privacy concerns can further increase the complexity of this phase. To address this issue, we aim to use machine learning methods such as deep reinforcement learning (DRL) as future work.

In IoT-Fog-Cloud architectures, changes in the profile of IoT devices, e.g., mobility, and Fog/Cloud infrastructure providers, e.g., availability, is inevitable and this gives rise to the service migration problem. Therefore, as another future research direction, this important feature can be investigated and applied in FLEX.

Serverless computing is an emerging computing and a "pay-per-use" model for on-demand processing of requests [20, 8, 4]. This model allows application and service developers to focus only on their code without thinking about managing servers. Given that the number of Serverless computing providers in Edge and Cloud environments is expected to significantly increase, this is viable for FLEX to be extended for function placement in Serverless computing with multiple providers.

8 Conclusions and Future Work

The current research presented a novel platform, known as FLEX, for the SPP in a multi-Fog and multi-Cloud computing environment, which offers two significant characteristics: flexibility and scalability. It is considered flexible since it enables FIPs and CIPs to implement their own strategies for placing services. It is regarded as a scalable platform since it facilitates the addition of new Fog and Cloud providers. We formulated the SPP as a problem of optimization in order to minimize cost and delay and then solved the problem efficiently by presenting a cost- and delay-aware algorithm. We used a simulation environment to implement FLEX and assessed its performance by conducting a variety of experiments. According to the results of simulation, the presented heuristic exhibits significantly higher performance than the baseline policies. Application of techniques from the game theory in the provider selection step is a future line of research that we plan to pursue. Also, we aim to take more criteria into account for the provider selection such as reliability, availability, and so on.

Declarations

`Ethical Approval`

This manuscript has not been published and is not under consideration for publication elsewhere.

`Competing interests`

The authors have no competing interests to declare that are relevant to the content of this article.

`Authors' contributions`

Sadoon Azizi conceived of the presented idea. Sadoon Azizi and Pedram Farzin wrote the original draft. Sadoon Azizi, Mohammad Shojafar and Omer Rana shaped the research and commented on the manuscript. Pedram Farzin made the simulations. All authors reviewed the manuscript.

`Funding`

No funding was received for conducting this study.

`Availability of data and materials`

The dataset used in this study is available in the text.

References

1. Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (Accessed: 2021-09-05)
2. Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., Tsafrir, D.: Deconstructing amazon ec2 spot instance pricing. ACM Transactions on Economics and Computation (TEAC) **1**(3), 1–20 (2013)
3. Alencar, D., Both, C., Antunes, R., Oliveira, H., Cerqueira, E., Rosário, D.: Dynamic microservice allocation for virtual reality distribution with qoe support. IEEE Transactions on Network and Service Management (2021)
4. Aslanpour, M.S., Toosi, A.N., Cicconetti, C., Javadi, B., Sbarski, P., Taibi, D., Assuncao, M., Gill, S.S., Gaire, R., Dustdar, S.: Serverless edge computing: vision and challenges. In: 2021 Australasian Computer Science Week Multiconference, pp. 1–10. IEEE (2021)
5. Baranwal, G., Yadav, R., Vidyarthi, D.P.: Qoe aware iot application placement in fog computing using modified-topsis. Mobile Networks and Applications **25**(5), 1816–1832 (2020)

6. Bonomi, F., Milito, R., Natarajan, P., Zhu, J.: Fog computing: A platform for internet of things and analytics. In: Big data and internet of things: A roadmap for smart environments, pp. 169–186. Springer (2014)
7. Cao, X., Tang, G., Guo, D., Li, Y., Zhang, W.: Edge federation: Towards an integrated service provisioning model. IEEE/ACM Transactions on Networking **28**(3), 1116–1129 (2020)
8. Castro, P., Ishakian, V., Muthusamy, V., Slominski, A.: The rise of serverless computing. Communications of the ACM **62**(12), 44–54 (2019)
9. Dhingra, S., Madda, R.B., Gandomi, A.H., Patan, R., Daneshmand, M.: Internet of things mobile–air pollution monitoring system (iot-mobair). IEEE Internet of Things Journal **6**(3), 5577–5584 (2019)
10. Farzin, P., Azizi, S., Shojafar, M., Rana, O., Singhal, M.: Flex: a platform for scalable service placement in multi-fog and multi-cloud environments. In: Australasian Computer Science Week 2022, pp. 106–114 (2022)
11. Ghaemi, S., Khazaei, H., Musilek, P.: Chainfaas: An open blockchain-based serverless platform. IEEE Access **8**, 131,760–131,778 (2020)
12. Goudarzi, M., Wu, H., Palaniswami, M., Buyya, R.: An application placement technique for concurrent iot applications in edge and fog computing environments. IEEE Transactions on Mobile Computing **20**(4), 1298–1311 (2020)
13. Grozev, N., Buyya, R.: Multi-cloud provisioning and load distribution for three-tier applications. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **9**(3), 1–21 (2014)
14. Hartmanis, J.: Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). Siam Review **24**(1), 90 (1982)
15. Hassan, H.O., Azizi, S., Shojafar, M.: Priority, network and energy-aware placement of iot-based application services in fog-cloud environments. IET communications **14**(13), 2117–2129 (2020)
16. Hernández, Á.B., Perez, M.S., Gupta, S., Muntés-Mulero, V.: Using machine learning to optimize parallelism in big data applications. Future Generation Computer Systems **86**, 1076–1092 (2018)
17. Hudson, N., Khamfroush, H., Lucani, D.E.: Qos-aware placement of deep learning services on the edge with multiple service implementations. arXiv preprint arXiv:2104.15094 (2021)
18. Iyer, G.N., Raman, V., Aswin, K., Veeravalli, B.: On the strategies for risk aware cloud and fog broker arbitrage mechanisms. In: 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), pp. 794–799. IEEE (2020)
19. Javed, B., Bloodsworth, P., Rasool, R.U., Munir, K., Rana, O.: Cloud market maker: An automated dynamic pricing marketplace for cloud users. Future Generation Computer Systems **54**, 52–67 (2016)
20. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., et al.: Cloud programming simplified: A berkeley view on serverless computing. arXiv preprint arXiv:1902.03383 (2019)
21. Kassab, W., Darabkh, K.A.: A–z survey of internet of things: Architectures, protocols, applications, recent advances, future directions and recommendations. Journal of Network and Computer Applications **163**, 102,663 (2020)
22. Liu, L., Zhang, M., Buyya, R., Fan, Q.: Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. Concurrency and Computation: Practice and Experience **29**(5), e3942 (2017)
23. Luo, J., Yin, L., Hu, J., Wang, C., Liu, X., Fan, X., Luo, H.: Container-based fog computing architecture and energy-balancing scheduling algorithm for energy iot. Future Generation Computer Systems **97**, 50–60 (2019)
24. Mahmud, R., Kotagiri, R., Buyya, R.: Fog computing: A taxonomy, survey and future directions. In: Internet of everything, pp. 103–130. Springer (2018)
25. Mahmud, R., Ramamohanarao, K., Buyya, R.: Edge affinity-based management of applications in fog computing environments. In: 12th IEEE/ACM International Conference on Utility and Cloud Computing, pp. 61–70. IEEE/ACM (2019)
26. Mahmud, R., Ramamohanarao, K., Buyya, R.: Application management in fog computing environments: A taxonomy, review and future directions. ACM Computing Surveys (CSUR) **53**(4), 1–43 (2020)
27. Mahmud, R., Srirama, S.N., Ramamohanarao, K., Buyya, R.: Profit-aware application placement for integrated fog–cloud computing environments. Journal of Parallel and Distributed Computing **135**, 177–190 (2020)

28. Maia, A.M., Ghamri-Doudane, Y., Vieira, D., de Castro, M.F.: An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing. Computer Networks **194**, 108,146 (2021)
29. Mukherjee, M., Shu, L., Wang, D.: Survey of fog computing: Fundamental, network applications, and research challenges. IEEE Communications Surveys & Tutorials **20**(3), 1826–1857 (2018)
30. Nanda, A., Puthal, D., Rodrigues, J.J., Kozlov, S.A.: Internet of autonomous vehicles communications security: overview, issues, and directions. IEEE Wireless Communications **26**(4), 60–65 (2019)
31. Natesha, B., Guddeti, R.M.R.: 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), pp. 24–25. IEEE (2018)
32. Natesha, B., Guddeti, R.M.R.: 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**, 102,972 (2021)
33. Nayeri, Z.M., Ghafarian, T., Javadi, B.: Application placement in fog computing with ai approach: Taxonomy and a state of the art survey. Journal of Network and Computer Applications p. 103078 (2021)
34. Omer, S., Azizi, S., Shojafar, M., Tafazolli, R.: A priority, power and traffic-aware virtual machine placement of iot applications in cloud data centers. Journal of Systems Architecture **115**, 101,996 (2021)
35. Qi, J., Yang, P., Min, G., Amft, O., Dong, F., Xu, L.: Advanced internet of things for personalised healthcare systems: A survey. Pervasive and Mobile Computing **41**, 132–149 (2017)
36. ur Rehman, M.H., Yaqoob, I., Salah, K., Imran, M., Jayaraman, P.P., Perera, C.: The role of big data analytics in industrial internet of things. Future Generation Computer Systems **99**, 247–259 (2019)
37. Sami, H., Mourad, A.: Dynamic on-demand fog formation offering on-the-fly iot service deployment. IEEE Transactions on Network and Service Management **17**(2), 1026–1039 (2020)
38. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE internet of things journal **3**(5), 637–646 (2016)
39. Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., Leitner, P.: Optimized iot service placement in the fog. Service Oriented Computing and Applications **11**(4), 427–443 (2017)
40. Skarlat, O., Schulte, S.: Fogframe: a framework for iot application execution in the fog. PeerJ Computer Science **7**, e588 (2021)
41. Skarlat, O., Schulte, S., Borkowski, M., Leitner, P.: Resource provisioning for iot services in the fog. In: 2016 IEEE 9th international conference on service-oriented computing and applications (SOCA), pp. 32–39. IEEE (2016)
42. Sonkoly, B., Czentye, J., Szalay, M., Németh, B., Toka, L.: Survey on placement methods in the edge and beyond. IEEE Communications Surveys & Tutorials (2021)
43. Sriraghavendra, M., Chawla, P., Wu, H., Gill, S.S., Buyya, R.: Dosp: A deadline-aware dynamic service placement algorithm for workflow-oriented iot applications in fog-cloud computing environments. In: Energy Conservation Solutions for Fog-Edge Computing Paradigms, pp. 21–47. Springer (2022)
44. Sterz, A., Felka, P., Simon, B., Klos, S., Klein, A., Hinz, O., Freisleben, B.: Multi-stakeholder service placement via iterative bargaining with incomplete information. IEEE/ACM Transactions on Networking (2022)
45. Tasiopoulos, A., Ascigil, O., Psaras, I., Toumpis, S., Pavlou, G.: Fogspot: Spot pricing for application provisioning in edge/fog computing. IEEE Transactions on Services Computing (2019)
46. Velasquez, K., Abreu, D.P., Paquete, L., Curado, M., Monteiro, E.: A rank-based mechanism for service placement in the fog. In: 2020 IFIP Networking Conference (Networking), pp. 64–72. IEEE (2020)
47. Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., Jue, J.P.: All one needs to know about fog computing and related edge computing paradigms: A complete survey. Journal of Systems Architecture **98**, 289–330 (2019)
48. Yousefpour, A., Patil, A., Ishigaki, G., Kim, I., Wang, X., Cankaya, H.C., Zhang, Q., Xie, W., Jue, J.P.: Fogplan: A lightweight qos-aware dynamic fog service provisioning framework. IEEE Internet of Things Journal **6**(3), 5080–5096 (2019)

49. Zeinab, K.A.M., Elmustafa, S.A.A.: Internet of things applications, challenges and related future technologies. World Scientific News **2**(67), 126–148 (2017)
50. Zikria, Y.B., Ali, R., Afzal, M.K., Kim, S.W.: Next-generation internet of things (iot): Opportunities, challenges, and solutions. Sensors **21**(4), 1174 (2021)