Journal of
Sensor and Actuator Networks

Article

SCATTER: Service Placement in Real-Time Fog-Assisted IoT Networks

Fariba Khosroabadi [1,*,†], Faranak Fotouhi-Ghazvini [1,†] and Hossein Fotouhi [2,*]

1 Department of Computer Engineering and IT, University of Qom, Qom 3716146611, Iran; f-fotouhi@qom.ac.ir
2 School of Innovation, Design and Engineering, Mälardalen University, 721 23 Västerås, Sweden
* Correspondence: f.khosroabadi@stu.qom.ac.ir (F.K.); hossein.fotouhi@mdh.se (H.F.)
† These authors contributed equally to this work.

Abstract: Internet of Things (IoT) networks dependent on cloud services usually fail in supporting real-time applications as there is no response time guarantees. The fog computing paradigm has been used to alleviate this problem by executing tasks at the edge of the network, where it is possible to provide time bounds. One of the challenging topics in a fog-assisted architecture is to task placement on edge devices in order to obtain a good performance. The process of task mapping into computational devices is known as Service Placement Problem (SPP). In this paper, we present a heuristic algorithm to solve SPP, dubbed as clustering of fog devices and requirement-sensitive service first (SCATTER). We provide simulations using iFogSim toolkit and experimental evaluations using real hardware to verify the feasibility of the SCATTER algorithm by considering a smart home application. We compared the SCATTER with two existing works: edge-ward and cloud-only approaches, in terms of Quality of Service (QoS) metrics. Our experimental results have demonstrated that SCATTER approach has better performance compared with the edge-ward and cloud-only, 42.1% and 60.2% less application response times, 22% and 27.8% less network usage, 45% and 65.7% less average application loop delays, and 2.33% and 3.2% less energy consumption.

Keywords: cloud computing; fog computing; internet of things; quality of service; service placement; smart home

Citation: Khosroabadi, F.; Fotouhi-Ghazvini, F.; Fotouhi, H. SCATTER: Service Placement in Real-Time Fog-Assisted IoT Networks. *J. Sens. Actuator Netw.* **2021**, *10*, 26. https://doi.org/10.3390/jsan10020026

Academic Editor: Igor Bisio

Received: 3 February 2021
Accepted: 31 March 2021
Published: 6 April 2021

1. Introduction

Internet of Things (IoT) is one of the most popular topics in information and communication technology. So far, there has been a lot of efforts to implement and adapt IoT to the traditional Internet technologies. The purpose of IoT is to identify things and devices (IoT devices) connected to the Internet and reduce the human role in the management and control process. Furthermore, IoT is extending the power of the Internet over computers to a wide range of things, digital machines, animals, and people [1] so that this paradigm allows for a new type of interaction between things-humans and things-things.

Due to the existence of large number of IoT devices, huge amount of data is generated, which require processing and storage that eventually causes heavy traffic load on the network. Additionally, IoT devices have limitations in terms of computational resources, storage, battery power, and bandwidth (BW); thus, exploiting complex algorithms on such devices is impossible. Many of these limitations can be solved using the Cloud of Things (CoT) technology [2–4]. Cloud of Things refers to the integration of Internet of Things with Cloud Computing, which is a high-performance cloud-based IoT application platform, and allows to remotely monitor, manage and control the IoT-enabled devices.

The combination of the cloud and IoT technologies provides the opportunity of enhancing network performance by offloading the processing to the cloud and achieve better resource utilization. In most of the CoT models, application services are transferred to the cloud data centers to execute and store [5]. One of the drawbacks of these models is getting high and unpredictable delays. Cloud data centers usually reside in a multi-hop distance, causing delays and sometimes unbounded response time. The situation is even worse when employing mobile IoT devices.

Many IoT applications are classified as delay-sensitive application, e.g., smart home, health monitoring and factory automation; meaning that it is of paramount importance to guarantee delays in the system. In smart home, IoT devices are used for monitoring and controlling of different devices and appliances. Smart homes can cooperate in managing energy utilization in the building, enhancing security and improving healthcare applications. Some of the smart home applications are considered as emergency and delay sensitive. For example, fire events, healthcare systems, and intruder detection are considered as emergency scenarios. Relying on cloud services to provide processing algorithms for IoT networks will cause critical consequences to time sensitive applications. By increasing the number of IoT devices in the future, the communication medium will become even more congested, creating more collision, packet losses and delays. Thus, remote cloud data centers cannot adequately handle latency-sensitive requirements, location awareness, and high mobility [6–8].

Hence, to provide the requirements time sensitive IoT applications, Cisco has introduced a new paradigm, called fog computing. The term "fog" means cloud close to the ground, so fog computing provides processing, computing, communication, storage, networking functions, and control of tasks in the network edge and closer to the IoT and end-user devices, where data and tasks are generated [9–13]. The meaning of the closeness is the hierarchical level of the computational devices that host and execute the application services [14].

In fog computing systems, any resource with computational power, storage, and networking capabilities is called a fog device/fog node. In fog computing, the fog nodes are geographically distributed, and operate as a bridge between end-user devices and the cloud [9].The heterogeneous resources, such as smartphones, personal computers, laptops, access points, and cloudlets, are known as fog devices [15]. The fog computing extends the cloud computing, and it has many advantages. For instance, (i) the fog computing executes applications closer to end-user and IoT devices, (ii) it improves latency, response time, and cloud workload for real-time applications [16], (iii) it increases network scalability, and (iv) it supports mobility in devices [11,17,18].

Fog computing platform still has a long way to go, and it is in the infancy stage. Despite all the mentioned achievements, fog computing is faced with new challenges that require more research and attention [19,20]. The placement of services is an optimization problem in fog computing. Based on this optimization problem, services should be place on computational resources close to end-user more efficiently. We refer to this problem as the Service Placement Problem (SPP) [21]. The service placement may require transferring application tasks from a fog device to another at a high-level or same level.

The main motivation for this research are as follow: (i) implementing an approach using a real test-bed for use in smart systems, such as smart homes and health care systems, where system response time in reaction to a specific event is critical, (ii) efficient management of computing resources in the network in response to the demands of IoT devices, by introducing a fog-assisted cloud architecture, (iii) reducing communication delay and execute services faster by placement of services at the edge of the network and closer to IoT devices, and (iv) covering up some of the weaknesses of past works, such as linking fog nodes on a same level, to make the most of using from the fog layer, and using experimental data to evaluate algorithm results, and approximating algorithm results to reality.

In this paper, we propose a different fog enabled cloud-based service placement approach for a smart home applications. We optimize the utilization of resources, serving delay-sensitive applications, and task execution according to application requirements, e.g., computing, storage, and BW demands. Our policy considers and optimizes various parameters, including response time, network usage, average application loop delays, and energy consumption. The proposed model is validated in a real environment for smart home application. For evaluation, we integrated the proposed approach into the iFogSim simulator [5] and compared our proposed model (SCATTER (clustering of fog devices and requirement-sensitive service first)) with two existing works, edge-ward and cloud-only proposed in Reference [5].

The main contributions of our work are: (i) introducing a hierarchical fog-cloud architecture for optimization of SPP in which fog nodes are clustered as multiple groups and highlighting horizontal scalability of the fog, so that it improves the maximum use of computing resources in fog layer, (ii) implementing a request handler mechanism on the iFogSim simulation environment using real-data obtained from a test-bed to validate the results obtained from the implementation of the algorithm in the real data framework and increases the reliability of the algorithm and its possibility of implementation in the real world, (iii) a comparison against two existing works to show proposed approach improves Quality of Service (QoS) parameters in terms of application response time, network usage, average application loop delays, and energy consumption, and (iv) design and implementation of an approach to manage computing resources in the network more effectively in response to the demands of different IoT devices to improve the execution speed of real-time applications and prevent traffic flow on one fog node by clustering fog nodes and balancing the network load on different fog nodes.

The rest of the paper is organized as follows. In Section 2, we overview the state-of-the-art works in the area of service placement. We present the system model in Section 3, describe the proposed architecture in the fog computing framework, and formulate the SPP. In Section 4, the proposed heuristic solutions are detailed. Section 5 describes the case study and experimental setup. We implement policy in a simulator and evaluate results in Section 6. Finally, Section 7 presents conclusions and future work.

2. Related Works

In this section, we review the related works that deal with the SPP in the fog computing platform. Research into IoT applications in fog-cloud computing exists with various unsolved research challenges [18], such as mobility of fog nodes, heterogeneous fog nodes, high scalability environment, improving the QoS requirements, and so on. The solutions have been defined for environments, such as smart homes, smart cities, healthcare, or mobile micro-cloud. The existing works try to solve the SPP and optimize it by highlighting one or more challenges. But this does not mean that researchers do not consider the other challenges and aspects of the SPP. Rather, in all existing works, several challenges are considered in designing and implementing the proposed solution, but more emphasis is placed on one challenge and solving the problem through it. So these challenges are completely related and interconnected and we need to investigate related works in these topics, because all of them exist in the real world together and affect each other. In each work, according to the available computing facilities and resources, some of them are emphasized more. We describe some of existing works for solving the SPP in the fog computing platforms with focusing on improving the QoS requirements and high scalibility.

In many application domains with stringent requirements, cloud computing in unable to provide QoS requirements. Hence, to achieve this goal, authors in Reference [22] have formulated an Integer Nonlinear Programming and introduced two greedy algorithms that must run periodically. The results of their experiments were validated based on real-world traffic traces and a Discrete Time Markov Chain (DTMC-based) traffic generator. In Reference [23], the authors have presented an energy management approach that has been implemented as a service over the fog computing platform. In Reference [24], the authors solved the SPP using the classification and regression tree algorithm in mobile fog computing to minimize response time and power consumption. In this model, the power consumption of mobile devices were compared to WiFi's power consumption, which was eventually offloaded if needed. A framework called FogFrame was implemented in Reference [25] that defines the necessary communication constructions for retaining execution of applications tasks in the fog layer. Evaluating their approach allows them to navigate experiments that show how the framework performs service placement, deployment, and execution. Most research about fog-cloud computing uses a default case study of a simulator in your implementations and evaluations. In Reference [18], the authors proposed a framework for resource management in fog-cloud based environments, which was validated via the deployment of a smart home case study. The authors in Reference [5] described two case studies and demonstrated effectiveness of iFogSim for evaluating resource management techniques including cloud-only and edge-ward approaches for application service placement. In cloud-only, processing services are transmitted to the cloud data center and in edge-ward services push towards fog devices when enough resources are available. Some research works for solving SPP employ optimization techniques [26,27], such as greedy, heuristics, ILP, and genetic algorithms. In Reference [21], the authors consider SPP for IoT applications over fog resources as an optimization problem and proposed a problem resolution heuristic using the genetic algorithm. In this work, fog devices are considered in fog colonies. The fog colonies act as edge-based computing networks that are composed of a set of fog cells connected to fog devices at a high-level, where they can work autonomously. They improved application response time and the cost of service execution. An algorithm is investigated in Reference [28] based on simultaneous wireless information and power transfer technology. They have used an offloading policy based on the Lyapunov optimization for minimization of delay. The mobility of fog nodes and end-user devices is an issue that has attracted many research works [29–31]. In Reference [32], the authors proposed a solution for the task placement problem for two IoT applications in a dynamic fog computing environment. In this work, the mobility of fog nodes and end-user devices are considered, and the problem has been formulated as an ILP, which was solved using IBM ILOG CPLEX Optimization (CPLEX) optimizer [33]. Although when the number of services and fog nodes increases, solving the SPP is complicated—ILP is an NP-hardness (Non-deterministic Polynomial-time hardness) problem. In Reference [24], a weighted cost model and application placement technique based on the Memetic algorithm was proposed, due to the heterogeneity of applications and devices to minimize the execution time and energy consumption. In Reference [3], the authors proposed a multi-tier architecture for IoT service provisioning based on context-aware information, such as location, resource consumption of services, and application response times. The authors in Reference [14] proposed a decentralized heuristic placement algorithm for micro-services-based IoT applications that was evaluated in the iFogSim simulator. In this work, the fog layer was considered as clusters, where each cluster has a controller node. According to the experimental results, this solution minimizes latency and network usage. Reference [34] has presented a resource placement method for fog-based IoT systems to reduce their latency and energy consumption. The main idea of this work is to use the idle processing capacity of fog nodes in each zone through the maximal placement of services on these nodes. Similarly, in Reference [35], an energy-aware allocation strategy is proposed for SPP, so that it improves the energy consumption and the delay application in comparison with edge-ward allocation and cloud-only policies.

Some of the related works have employed commercial tools, such as Amazon Web Service (AWS) and Google Cloud Drive, to deploy various applications in the cloud. These tools provide cloud services from several data centers and available zones located in different parts of the world. AWS was created in order to develop an organized set of APIs, products, and services for the customers, and Google Cloud Drive is a program that revolves around offering tools, resources, and training in order to fulfill the IT business necessities [36]. In Reference [37], a location-aware online game to which users are connected from different geographic locations is presented to highlight the feasibility of fog computing. Multiple game clients that were in close proximity to the edge node established connection with the edge server to play the game. In this work, the game server was hosted in an Amazon data center on a t2.micro virtual machine, so that service players could be offloaded onto the edge nodes located closer to the players to improve quality of service. They found that the average response time for a user and the volume of traffic between the edge and the cloud server are improved compared to the cloud-only model. Reference [38] has addressed the data protection and the performance issues by proposing a region-based trust-aware model for trust translation among fog nodes of regions, and introducing a fog-based privacy-aware for access control at fog nodes. It keeps track of changes of data location among regions periodically by using a Location Register Database, as well as an enhanced verification procedure. It implements the Location Register Database at the cloud for synchronizing among Location Register Databases at fog nodes. Specifically, The AWS was used to provide resources and the Location Register Database as global location database. There are some limitation with AWS and Google Cloud Drive. One weaknesses with AWS and Google Cloud Drive is their limited resources due to the geographical location of IoT devices. So, where IoT devices are located can determine just how many cloud resources will have access to. There are several other limiting issues, such as (i) lack of knowledge due to the confidentiality in their implementations, (ii) expensive technical support, and (iii) lack of proper execution of the services due to the technical faults.

In reality, the number of IoT devices is enormous and scattered. Thus, some of the related works focus on high scalability in the fog environment. In Reference [39], the authors presented a task orchestration platform on a large scale that focuses on the role of edge computing to enable the self-capabilities of IoT. They also proposed a task orchestration algorithm that is based on a fuzzy decision tree. It leverages reinforcement learning, which allows it to adapt to unpredictable environmental changes. The evaluation of their proposed algorithm is done by PureEdgeSim [40] simulator. In Reference [41], a self-similarity-based load balancing mechanism for large-scale fog computing was proposed. They also presented an adaptive threshold policy, which accurately defines the load threshold on each fog node. In the real world, an integrated fog-cloud network consists of heterogeneous fog nodes that vary in terms of computational and storage resources. Some fog nodes have additional resources for sharing with their neighboring nodes [42]. Hence, to tackle the heterogeneity problem, authors in Reference [43] proposed a paired offloading of multiple tasks policy in heterogeneous fog networks, in which multiple IoT devices, multiple fog servers, and multiple cloud servers are available, so that their algorithm optimizes the execution time and energy consumption. The simulation results showed that their algorithm could offer the near-optimal performance at two orders of magnitude lower complexity and higher efficiency, comparing to a centralized optimal algorithm for computation offloading. In Reference [44], a generic three-layer fog computing network architecture is proposed for solving SPP. Thus, a large number of user equipment are in the user layer, which have the mobility and require large amount of computation resource. Fog layer also consists of a large scale of fog computing nodes, which are widely deployed among the users. In their approach, the mobility and task offloading are modeled as a MINLP (Mixed-Integer Nonlinear Programming) problem. This work is divided into two parts: (i) task offloading and (ii) resource allocation. For task offloading, a coefficient-based fog node selection algorithm was proposed, and for solving the computation resource allocation problem, a distributed resource optimization algorithm based on the genetic algorithm has been implemented. The authors in Reference [27] proposed a deep reinforcement learning to solve the offloading problem of multiple service nodes for the cluster and multiple dependencies for mobile tasks in large-scale heterogeneous mobile edge computing. The simulation results show that their offloading strategy has better performance in energy consumption, load balancing, latency and average execution time.

Some of the limitations of existing works are: (i) following the traditional hierarchical structure of fog-cloud, not making optimal use of the computing resources available in the fog layer and executing more services on cloud data centers, which increases the user response time [5,23,24,26,44], (ii) using hypothetical data instead of experimental data and, thus, uncertainty about the feasibility of implementing the algorithm in the real world [5,14,24,26,27,34,40,41,43,45], (iii) modeling the problem as an ILP and the complexity of problem solving when increasing the number of services and fog nodes on a large scale [21,22,32,44], (iv) ignoring priority for the applications and demands of services [5,14,18,21–23,25–27,34,35,40,43], (v) ignoring mobility of fog nodes [3,14,18,21–23,25,26,35,41,43,45], and (vi) improving only two or three QoS metrics [14,23,25,43]. In the proposed algorithm, we have tried to overcome some of these limitations.

For the sake of clarity, the characteristics of the state-of-the-art works have been summarized in Table 1 by indicating performance metrics and features. The performance metrics of these works are classified as energy consumption, average applications loop delays, application response times, network usage, and cost of execution. The SPP is summarized according to the features considered, such as mobility of users and fog nodes, the IoT scope, the architecture of the fog layer, used data type for evaluation that can be real data obtained a case study or dataset, and non-real. In the SCATTER, all performance metrics are considered, and the scope of the system is to provide a general IoT system while the fog layer architecture is clustering and employs real data for evaluation.

Table 1. Summary of the related works for Service Placement Problem (SPP) in fog-cloud environments.

Refs	Objective/Performance Metrics					Features			
	Energy	Delay	Response Time	Network Usage	Cost	Mobility	Scope [a]	Architecture [b]	Used Data [c]
[22]	n	y	n	n	y	n	C	H	R
[23]	y	y	n	n	y	n	I	H	R
[24]	y	n	y	n	y	n	G	C	N
[25]	n	y	y	n	n	n	G	H	R
[21]	n	y	y	n	y	n	G	H	N
[28]	y	y	y	n	y	n	G	C	R
[32]	n	y	y	n	n	y	C	C	R
[44]	y	y	n	n	n	y	G	H	N
[42]	n	y	n	n	y	n	G	H	N
[43]	y	y	n	n	y	n	G	H	N
[3]	y	y	y	n	y	n	C	C	R
[14]	n	y	n	y	n	n	G	C	N
[46]	n	y	n	y	n	n	G	H	N
[18]	y	y	y	y	n	n	G	H	R
[39]	y	y	n	n	n	y	G	H	N
[41]	n	y	n	n	n	n	C	C	N
SCATTER	y	y	y	y	y	n	G	C	R

[a] Scope of system: G general IoT systems; I industrial IoT systems; C crowdsensing apps, [b] Fog Layer Architecture: C clustering; H hierarchical, [c] Used Data for Evaluation: R real data; N non-real data.

3. The System Modeling and Problem Formulation

This section presents the proposed policy for fog-assisted cloud-based service placement in a smart home application. In this work, we have assumed that each application has several services that can be deployed, upgraded, and executed independently. Each service has the resource requirement, including terms of CPU (Central Processing Unit), RAM (Random Access Memory), and BW (Band Width).

Besides, the Sense-Process-Actuate model [5] is used for IoT applications. According to this model, the sensors and IoT devices collect raw data from the environment, and then sensing tasks are created. The sensing task calls the client task. The resulting task execution is stored in the actuating task and displayed to the end-user. All of the applications have a sensing task and an actuating task that is deployed into fog cells. The rest of the services are processing services, placed, and executed on fog devices and cloud data centers based on service placement policy. The proposed architecture and formulation of SPP are discussed in detail within the following subsections.

3.1. Proposed Architecture for the SCATTER (System Model)

In fog computing, computation, storage, and networking resources are provided using fog devices, such as cell phones, proxy servers, routers, and access points. These devices are often heterogeneous and geographically located in different regions. In this work, as shown in Figure 1, a hierarchical architecture consisting of three layers is considered. The fog layer is located between the IoT devices and the cloud data centers.

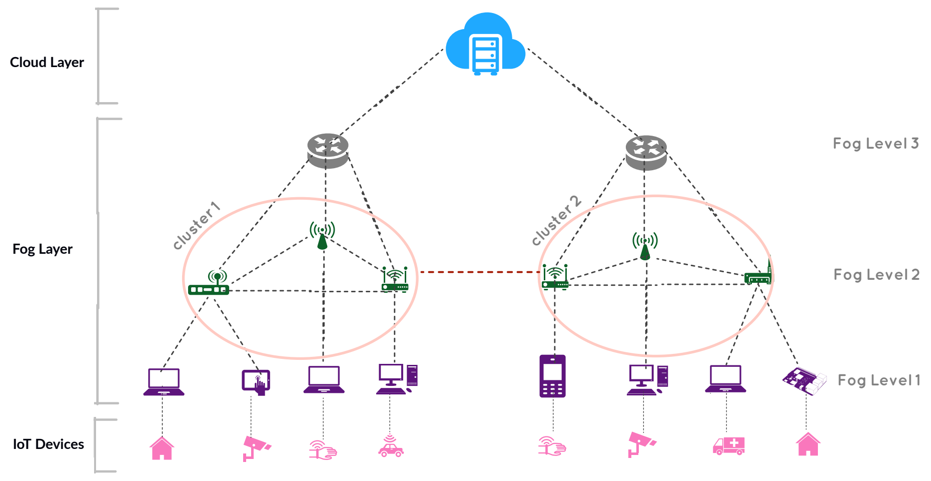

Figure 1. Proposed system architecture including 3 layers of Internet of Things (IoT), fog, and the cloud, while encompassing 3 fog levels.

In the fog layer, fog devices are located in a hierarchical level. Fog devices that are located in lower levels, have less computing, storage and networking capabilities. When moving from lower levels to higher levels, computing and storage capacity, as well as the communication delay of the devices, increase.

In traditional fog computing architectures, fog devices with the same level have no connection link to each other, and, if a fog device fails to run a service, it sends the service to a higher level or to the cloud, which increases latency. In our proposed architecture, each fog device, in addition to being connected to a higher level fog device, can also have connection links with other fog devices at the same level. Communication delay between two fog nodes of the same level is less than two fog nodes with two different levels. To better control the network and prevent high load density on a particular fog node, we cluster the devices in the fog layer. A set of fog devices with the same level form a cluster. A fog node can be placed in several clusters. In this work, it is assumed that each fog device, at a particular time, belongs to only one cluster. To organize and coordinate fog nodes in a cluster, we consider a head for each cluster called Fog Orchestrator node (FON). After entering the cluster, all services are first sent to the FON, and the FON decides which node to run the service based on its computational demands. If the computing resources in the fog layer cannot meet the demand of a service, the current service is sent to the highest layer (the cloud data centers), where the delay between the fog layer and the cloud is very high. In the following, the three layers considered in the proposed architecture are examined in more detail.

Moreover, we have considered that tasks have different computing demands as users may request different types of computing tasks. When tasks are offloaded to the fog nodes, they may introduce different computing demands, where these demands are not determined manually. Thus, we used a random function in the SCATTER implementation. In this way, the computing demands of each task in terms of RAM, CPU, and BW are selected randomly, and we have no involvement in selecting the values.

3.1.1. IoT Devices

The IoT devices are geographically distributed in different locations and include end-user devices, home appliances, wireless sensors, and actuators. These devices send services toward higher layer for processing and execution. In IoT devices, the sensors sense the environment and send requests to the gateways for processing and filtering, and the actuators provide the results of service execution for users.

3.1.2. Fog Layer

The fog layer is the middle layer and is divided into two sub-layers:

1. The fog cells (e.g., end devices, smartphones, wearable devices, tablets) [21].
2. The fog nodes (e.g., routers, switches, access points).

Fog cells represent virtual resources in IoT devices connected to sensors and actuators. These virtual resources are used for deploying and executing arbitrary IoT services.

The fog cells are virtual resources located at the lowest level of the fog layer and have less processing and storage capacity than fog devices. These virtual resources are used for deploying and executing of sensing and actuating services. Fog nodes are fog cells that implement processing services and act as access points for them. Fog nodes can contain several levels. At each level, several fog nodes are partitioned into multiple clusters. Based on the service placement algorithm, FON in each cluster decides which services should be deployed on fog nodes. Two fog nodes are in the same cluster if they satisfy the following conditions: (i) having same parent node in the hierarchical structure and (ii) having a connection link delay between two fog devices below a certain threshold (explained in Section 4).

3.1.3. Cloud Layer

The cloud layer is at the highest level of the hierarchical architecture, where cloud data centers are located there for providing more computing power than the fog devices. When the processing services require more computing resources than the fog layer's support, they are sent to the cloud data centers for deployment. If services are executed in the cloud, due to the significant distance and connection link delay between the end-user devices and cloud, the delay between services and then applications response time will increases.

3.2. Problem Statement

Consider a fog cluster with a FON, n fog nodes, and m fog cells. In each cluster, F_k denotes FON in the k-th cluster, $R_{fd} = \{fd_1^k, fd_2^k\ fd_3^k\ \ldots,\ fd_n^k\}$ signifies the set of fog devices, where fd_i^k is the i-th fog device in the k-th cluster, and $R_{fc} = \{fc_1^k\ \ fc_2^k\ \ fc_3^k,\ \ldots fc_m^k\}$ denotes the set of fog cells, where fc_i^k is the i-th fog cell in the k-th cluster. We also consider the closest neighbor cluster for the current cluster and use the notation N for neighbor FON and the notation C for the cloud.

Each device has different computational and storage resources. U_F^k, R_F^k, and B_F^k are used to indicate the total amount of CPU, RAM, and BW allocated to FON F in the k-th cluster, respectively. Similarly, the number of resources allocated to other devices in the network can be defined. The communication link delay between a particular fog cell fc_i^k, and F is denoted by $d_{fc_i^k}$. Analogously, $d_{fd_i^k}$, d_N, and d_C denote the communication link delay between F and fog device fd_i^k in the same cluster, F and neighbor FON, F, and cloud, respectively. Let the set of $D = \{F,\ R_{fd},\ R_{fc},\ N,\ C\}$ consist of all computational resources. Table 2 shows the notations and descriptions of terms used for service placement modeling.

Let $A = \{a_1,\ a_2,\ a_3, \ldots,\ a_{|A|}\}$ denote the set of applications that can be requested by IoT devices. Each application $a_p \in A$ is composed of independent tasks, $a_p = \{\tau_{p,1},\ \tau_{p,2},\ \ \tau_{p,3}, \ldots,\ \tau_{p,n}\}$, where application a_p has n tasks, and $\tau_{p,i}$ is the i-th task of application a_p. Each task $\tau_{p,i}$ requires the number of resources, such as CPU, RAM, and BW, that are denoted as $U_{\tau_{p,i}}$, $R_{\tau_{p,i}}$, and $B_{\tau_{p,i}}$, respectively.

Table 2. Notations and description of terms in the proposed algorithm.

Parameter	Notation	Description
Computational Resources	F_k	FON in the k-th cluster
	R_{fd}	The set of fog devices
	fd_i^k	The i-th fog device in the k-th cluster
	R_{fc}	The set of fog cells
	fc_i^k	The i-th fog cell in the k-th cluster
	N	Neighbor FON
	C	Cloud
	U_F^k	The total amount of CPU allocated to FON in the k-th cluster
	R_F^k	The total amount of RAM in FON in the k-th cluster.
	B_F^k	The total amount of BW in FON in the k-th cluster
	$d_{fc_i^k}$	The communication link delay between a particular fog cell fc_i^k and FON in the same cluster
	$d_{fd_i^k}$	The communication link delay between fog device fd_i^k and FON in the same cluster
	d_N	The communication link delay between neighbor FON and FON
	d_C	The communication link delay between cloud and FON
	D	The set of all computational resources
Applications	$H_{\tau_{p,i}}$	The devices that can host task $\tau_{p,i}$
	A	The set of applications that can be requested by IoT devices
	a_p	An application
	$\tau_{p,i}$	The i-th task of application a_p
	$U_{\tau_{p,i}}$	CPU demand of a task
	$R_{\tau_{p,i}}$	RAM demand of a task
	$B_{\tau_{p,i}}$	BW demand of a task
	D_{a_p}	Application deadline
	R_{a_p}	The response time of a_p
	$Req(a_p, f)$	The number of tasks for application a_p that device f can handle together
	$C_f(a_p)$	The computation capability of device f that can execute application a_p
	$t_{installing}(a_p)$	The requirement times for downloading and installing of application a_p
	$\delta(i,j)$	The distance of computational requirements between two sequential tasks

3.3. Problem Formulation

Consider each application a_p has a deadline D_{a_p} that determines by application users. In a_p, tasks must be placed on the resources in the set D for deployment. We assume that sensing and actuating tasks are placed on fog cells, and processing tasks are transmitted FON. Based on the task placement algorithm, FON will decide to transfer tasks on which device in the set D.

Let the set $H_{\tau_{p,i}}$ include devices that can host task $\tau_{p,i}$, where $H_{\tau_{p,i}} \subseteq D$. We consider $\{x^{F}_{\tau_{p,i}}\ x^{fd_j^k}_{\tau_{p,i}},\ x^{fc_h^k}_{\tau_{p,i}},\ x^{N}_{\tau_{p,i}},\ x^{C}_{\tau_{p,i}}\} \in \{0, 1\}$, where, if task $\tau_{p,i}$ can be placed on fog device fd_j^k, therefore, $x^{fd_j^k}_{\tau_{p,i}} = 1$, and, otherwise, $x^{fd_j^k}_{\tau_i} = 0$. Such notations can be defined for other devices within the set D. Equation (1) shows a service $\tau_{p,i}$ can be deployed on only one device.

$$x^{F}_{\tau_{p,i}} + x^{fd_j^k}_{\tau_{p,i}} + x^{fc_h^k}_{\tau_{p,i}} + x^{N}_{\tau_{p,i}} + x^{C}_{\tau_{p,i}} = 1 \quad , \qquad \forall \tau_{p,i}. \tag{1}$$

Each application a_i that is delay-sensitive should follow Equation (2), where R_{a_i} is the response time of a_i, and D_{a_i} is the deadline of a_i that determinate by application users.

$$R_{a_i} \leq D_{a_i} \quad , \qquad \forall a_i \in A. \tag{2}$$

The response time of a_i is composed of three parts: (1) the communication link delays between the end-user and the devices that tasks of application a_i are placed on them, (2) the delay of computation, and (3) the delay of application installation and deployment. Therefore, the response time of a_i can be written as Equation (3). $delay_{communication}$, $delay_{computation}$, and $delay_{deployment}$ are the communication link delays, application computation delay, and application deployment delay, respectively.

$$R_{a_i} = delay_{communication}(a_i) + delay_{computation}(a_i) + delay_{deployment} \quad \forall a_i \in A. \tag{3}$$

The communication link delays of application a_i can be written as Equation (4), where is a sum of communication link delays of placed tasks on a particular device.

$$delay_{communication}(a_i) = \sum_{\tau_j}^{a_i} \left(\sum_{f}^{H_{\tau_j}} \left(\left(d_{fc_m^k} \times x_{\tau_j}^{fc_h^k} \right) + \left(d_{fd_n^k} \times x_{\tau_j}^{fd_n^k} \right) + \left(d_N \times x_{\tau_j}^{\mathrm{N}} \right) + \left(d_C \times x_{\tau_j}^{\mathrm{C}} \right) \right) \right)$$
$$\forall a_i \in A,\ f \in H_{\tau_j},\ \tau_j \in a_i. \tag{4}$$

Let $Req(a_i, f)$ is the number of tasks for application a_i that device f can handle, where $a_i \in A$ and $f \in D$. The computation delay can be calculated by considering the computation capability of device f that can execute application a_i, defined by $C_f(a_i)$, and $Req(a_i, f)$ that is given by Equation (5).

$$delay_{computation}(a_i) = Req(a_i, f) \times C_f(a_i) \quad \forall a_i \in A, f \in D. \tag{5}$$

The deployment time of application a_i is considered as download time of packages, installation time of application, and time passed before each task is entirely placed on a computational resource, defined by T. It can be formulated as Equation (6), where $t_{installing}(a_i)$ is requirement times for downloading and installing application a_i.

$$delay_{deployment}(a_i) = t_{installing}(a_i) \times \sum_{\tau_j}^{a_i} \mathrm{T} \left(\sum_{f}^{H_{\tau_j}} x_{\tau_j}^{f} \right) \quad \forall a_i \in A,\ f \in H_{\tau_j},\ \tau_j \in a_i. \tag{6}$$

We consider the dependency of tasks in an application as a directed acyclic graph (DAG). In DAG, the vertices represent tasks, and edges represent data flows between tasks [5]. To improve QoS metrics and reduce waiting time in queues for tasks, we can cluster tasks according to expected CPU, expected storage space, and expected BW. In DAG, both tasks are interdependent, and the requirements of a task are requirements for task execution and requirements for transferring data via edges to the next task. Equation (7), Equation (8), and Equation (9) show normalization of task requirements. The distance of computational requirements between two sequential tasks, defined by $\delta(i, j)$, is calculated using Equation (10), where two tasks τ_i and τ_j are exactly sequential, and application a_k has n tasks. After the calculation of distances between tasks, tasks are sorted based on distances in ascending order. It means tasks that require fewer resources are deployed and executed earlier.

$$U^*_{\tau_{k,j}} = \frac{U_{\tau_{k,j}}}{\sum_{j=1}^{n} U_{\tau_{k,j}}} \tag{7}$$

$$R^*_{\tau_{k,j}} = \frac{R_{\tau_{k,j}}}{\sum_{j=1}^{n} R_{\tau_{k,j}}} \tag{8}$$

$$B^*_{\tau_{k,j}} = \frac{B_{\tau_{k,j}}}{\sum_{j=1}^{n} B_{\tau_{k,j}}} \tag{9}$$

$$\delta(i,j) = \sqrt{\left(U^*_{\tau_{k,i}} - U^*_{\tau_{k,j}}\right)^2 + \left(R^*_{\tau_{k,i}} - R^*_{\tau_{k,j}}\right)^2 + \left(B^*_{\tau_{k,i}} - B^*_{\tau_{k,j}}\right)^2}$$
$$\forall\, i,j \in \mathbb{N}, 1 \le i < n\,,\ 1 < j \le n,\ \ i = j-1. \quad (10)$$

According to Equation (11), in latency-sensitive applications, most services must be deployed in the fog layer as much as possible and maximize the number of placed services in fog devices. In this paper, we propose a fog-Assisted Cloud-Based Service Placement Algorithm (SCATTER) for smart home IoT devices addressing the challenges mentioned above.

It is worth mentioning that the formulas and mathematical relations expressed in Section 3.3 are matched in all solutions (SCATTER, edge-ward, and cloud-only). The SCATTER approach, like edge-ward and cloud-only, is implemented in the iFogSim simulator. We re-implemented all three algorithms with the same default parameters and considered new evaluation criteria, such as application response time, delay communication, delay computation, delay deployment, and average application loop delays, in the new implementation. So, the formulas are matched for three algorithms.

$$Maximize \sum_{i=1}^{n}\sum_{j=1}^{|A_i|}\left(\sum_{k=1}^{m} x^{F}_{\tau_i} + x^{fd^k_j}_{\tau_i} + x^{fc^k_j}_{\tau_i} + x^{N}_{\tau_i} + x^{C}_{\tau_i}\right). \quad (11)$$

4. Proposed Solution (SCATTER)

To solve SPP in the fog-cloud environment, we propose an efficient heuristic placement algorithm called clustering of fog devices and requirement-sensitive service first (SCATTER). The main ideas behind the SCATTER algorithm are: (i) placement of latency-critical applications as much as possible on the fog nodes that are closer to the user, (ii) placement of tasks that have fewer demands earlier, (iii) clustering fog nodes as multiple groups, and (iv) considering horizontal scalability for fog nodes.

4.1. Service Placement

We consider application deadlines as a factor in choosing applications to be executed. Each application that has fewer deadlines is selected to transmit its services to higher layers earlier than other applications. After selecting the application, its services are sorted according to demands in ascending order. Based on the assumed architecture, each IoT device is connected to a fog cell, and the end-user transmits the task data to the FON for processing. We assume that the sensing and actuating tasks are placed on the fog cells, and the processing tasks are submitted to the FON in the corresponding cluster.

The pseudo-code of the proposed approach is presented in Algorithms 1–3. The SCATTER approach considers latency-sensitive applications. In Algorithm 1, applications are sorted in ascending order based on deadlines. Application a_p has the shortest deadline, so it is the first application whose services are selected to execute.

An application contains a set of interconnected services. Each service has demands including CPU, RAM, and BW. The services are interconnected in a directional graph in which the edges are the flow of data transmitted between the services. The data generated as output by service $\tau_{p,j}$ is as input to service $\tau_{p,j+1}$. To locate a service on a fog device, the fog device must have the computing resources required by that service.

Algorithm 1 Service Placement Algorithm

```
Input: set of computational resources D, sets of Applications A, and sets of application tasks
Output: Placing of tasks in the computational resources
function TaskPlacement (A)
1:  Sort application list A based on deadline in ascending order
2:  Map <task name, distance> TaskDistanceMap
3:  for each application $a_p \in A$ do
4:      for each processing task $\tau_{p,j} \in a_p$ do
5:          TaskDistanceMap.put ($\tau_{p,j}$ , distanceCompute ($\tau_{p,j}$ , $\tau_{p,j+1}$))
6:      end for each
7:      Sort TaskDistanceMap based on distance in ascending order
8:      $FogDevicesList \leftarrow clusteringNodes\ (D, F)$
9:      counter = 0        ▷ counter is used to count the number of fog devices
10:     while TaskDistanceMap.size( ) > 0 do
11:         $task \leftarrow TaskDistanceMap.get(0)$   ▷ The first member of TaskDistanceMap is in task variable
12:         while $counter \leq FogDevicesList.size()$ do
13:             $currentDev \leftarrow FogDevicesList.get(counter)$   ▷ The first Fog Device is in currentDev
14:             if Compare (currentDev , service) is True then
15:                 place service on currentDev
16:                 $U_{currentDev} \leftarrow U_{currentDev} - U_{task}$
17:                 $R_{currentDev} \leftarrow R_{currentDev} - R_{task}$
18:                 $B_{currentDev} \leftarrow B_{currentDev} - B_{task}$
19:                 TaskDistanceMap.remove(0)
20:                 break
21:             else $counter \leftarrow counter + 1$
22:             end if
23:         end while
24:         if service is not placed then
25:             place service on cloud
26:             TaskDistanceMap.remove(0)
27:         end if
28:     end while
29: end for each
```

Algorithm 2 Compute distance of tasks

```
    Input: Application a, Task$\tau_1$, Task$\tau_2$
    Output: calculated distance between two tasks $\tau_1$ and $\tau_2$
    function distanceCompute
1:  Map < task name, distance > CurrentDistanceMap
2:  for each edge in a.getEdge() do
3:      if edge.get_source is equal to $\tau_1$ && edge.get_distination is equal to $\tau_2$ then
4:          $task\_cpu \leftarrow edge.get\_Tuple\_cpu + \tau_1.get\_cpu()$
5:          $task\_storage \leftarrow edge.get\_Tuple\_storage + \tau_1.get\_storage()$
6:          $task\_bandwidth \leftarrow edge.get\_Tuple\_bandwidth + \tau_1.get\_bandwidth()$
7:      end if
8:  end for each
9:  calculate distance according to Equation (7)–(10)
10: CurrentDistanceMap.put ($\tau_1$ , distance )
11: return CurrentDistanceMap. distance
```

The demands of a service are: (i) the necessary demands to execute the service, and (ii) the demands of the data flow that is transmitted from other services and Applied as input to the service. In the SCATTER approach, the policy of executing an application is to offload services that have lower demands first. After selecting application a_p with the least deadline, the demands of each of its services should be calculated. Then, the services should be arranged in ascending order according to their demands, and services with less demands should be offloaded sooner. In line 5 of Algorithm 1, in the *distanceCompute* function (Algorithm 2), the demands of all services of Application a_p are calculated in pairs.

Next, for maximum use of the fog layer and effective control of the fog nodes, they are grouped into multiple clusters, which is done in Algorithm 3. The fog nodes in a cluster have three features in common: (i) they are on the same level of the fog layer, (ii) the delay between each node and the FON is less than threshold delay, (iii) the fog nodes can be connected horizontally, and (iv) in hierarchical structure, the parent of all fog nodes in a cluster is the same (line 3 of Algorithm 3). These four conditions prevent fog nodes that are far apart from being in the same cluster. In fact, the goal of fog node clustering is to reduce latency and execute services faster.

After clustering the fog nodes, application a_p services are sent to the corresponding FON cluster. FON examines the service demands that need to be implemented and selects the most appropriate fog node to deploy. So, the service placed on the fog node and as stated in lines 16–18 of Algorithm 1, the amount of computational service demands is subtracted from the amount of computing resources available on the selected fog device. If the fog nodes in the fog layer do not have sufficient computing resources to place a service, the service is sent to the cloud data centers (lines 24–26 Algorithm 1). The stated actions are repeated for all applications and continue until all application services are executed.

Algorithm 3 Clustering of Nodes

```
   Input: set of computational resources D, Fog Orchestrator Node F
   Output: clustering of fog nodes
   function clusteringFogNodes (FogNode F)
1: List < FogNodeId > clusteringList
2: for each $FogDevice \in D$ do                         ▷ say fog device fd
3:     if (fd $\neq$ F) && (delayBetween(F,fd) < Cluster_Threshold_Delay) && (fd.getLevel( )
   == F.getLevel( ) ) && (fd.getParent( ) == F.getParent ( ) ) then
4:         clusteringList.add (F)
5:         clusteringList.add (fd)
6:     end if
7: end for each
8: return clusteringList
```

4.2. Time Complexity Analysis

The time complexity of our proposed algorithm consists of two parts: (i) the time complexity of calculating the distance of services from each other and sorting them, and placement request processing on fog devices that are handled in Algorithms 1 and 2, and (ii) the time complexity of the clustering of fog devices covered in Algorithm 3.

If number of applications is A and each application consists of S services (services represent vertices in the DAG model), and number of communication edges between services is E, the time complexity of calculating the distance of services and sorting them is equal to $O(|A| \times |S| \times |E|)$. If number of clusters is C, and each fog node cluster consists of F fog devices, the time complexity of placement request processing and comparing the service demands with the available computing resources is equal to $O(|S| \times |C| \times |F|)$. In Algorithm 3, worst case time complexity of fog devices clustering is equal to $O(|C| \times |F|)$.

5. Performance Evaluation

We validate the SCATTER approach by a small test-bed representing a smart home application domain. The application tasks in an IoT-based smart home are simulated in the iFogSim simulator to place tasks in fog nodes according to SCATTER policy.

5.1. Case Study: IoT-Based Smart Home

In this work, we design and implement a smart home case study using multiple IoT devices and sensors that are connected wirelessly to evaluate the SCATTER algorithm with real data. The sensors transmit data to fog cells (here, we have used a smartphone as a fog cell).

It is possible to consider various scenarios for the experiment. For example, during the daytime, especially when no one is at home, according to predetermined instructions, various events, such as turning off extra lights, covering the security system, turning on the facade lights, turning off the electrical equipment, and reducing the activity level of the engine room, can be executed with just one command. We consider a scenario consisting of a home with two rooms (living room and bedroom), so that sensors are located in different parts of the rooms.

In the implemented smart home, we have used a Raspberry Pi3 as FON, five sensors, four Arduino boards, and nRF240L01 transceiver. The sensors are, for example, temperature sensor (DHT11) to measure air temperature of the home, humidity sensor (YL69) to measure air humidity of home, Ultrasonic sensor for control of entry and exit to the room, light sensor for room lighting control, and motion sensor (PIR) for motion control. The temperature, humidity, and Ultrasonic sensors are located in the living room, and light and motion sensors are embedded in the bedroom.

The sensors are interconnected with each other wirelessly using the nRF24L01 transceiver that enables ultra-low-power and high-performance communication with low-cost host micro-controllers. Figure 2 depicts the interaction of the sensors and modules in the assumptive smart home scenario. Figure 2a,b show sensors in the living room, and Figure 2c show sensors in the bedroom. The communications between different parts of the system are depicted in Figure 3. Details on communication between different components of the system are provided below.

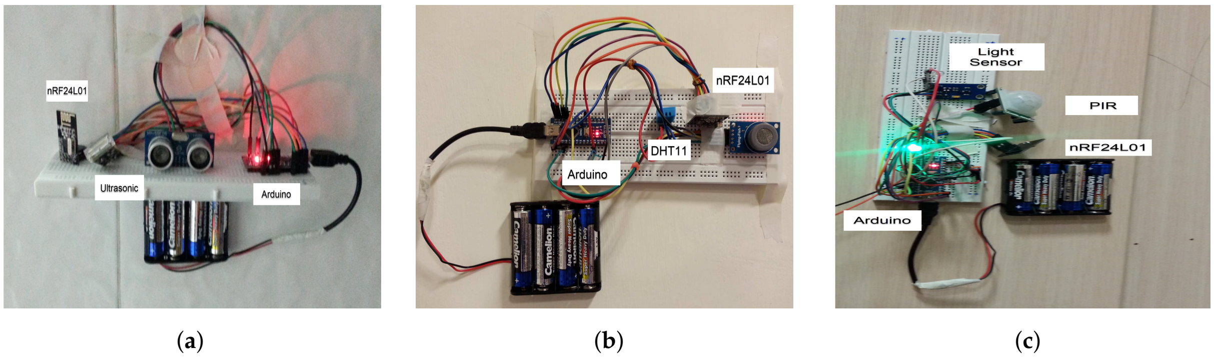

Figure 2. Illustration of different sensors in our smart home application: (**a**) Ultrasonic sensor, nRF24L01 transceiver, and Arduino board, (**b**) temperature and humidity sensor, nRF24L01 transceiver, and Arduino board, and (**c**) motion sensor, light sensor, nRF24L01 module, and Arduino board.

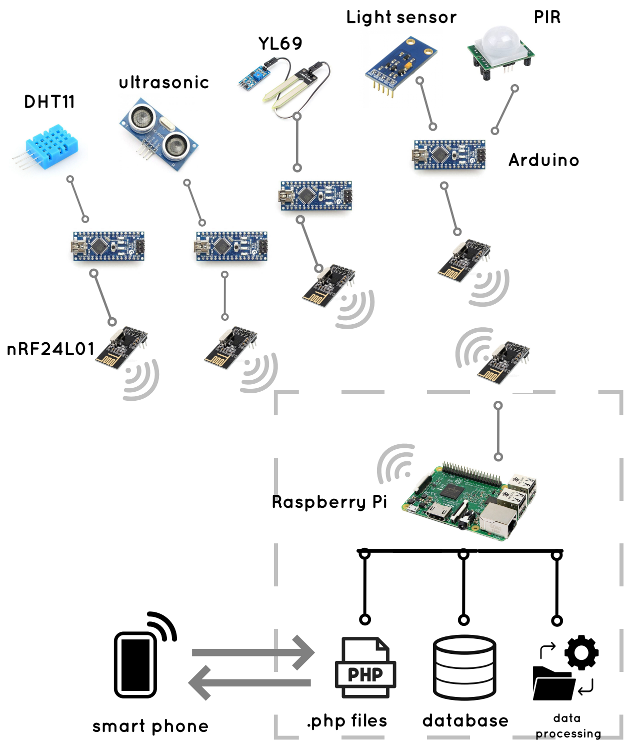

Figure 3. The communication between the components and sensors.

5.1.1. Sensors to Arduino

The sensors cannot inherently transmit tasks. A microcontroller must be connected to each sensor. We have used the Arduino microcontroller that is linked to each sensor via the wire. Therefore, tasks will transmit from the sensor and Arduino to higher layers. In each room, a set of sensors with Arduino boards are placed based on location of appliances, as shown in Figure 2.

5.1.2. Arduino to nRF24L01

For wireless communication between different sensors, Arduino boards, and FON, employing a transceiver module is necessary. We have used nRF240L wireless transceiver that connects to the Arduino via wire and exchanges data between fog cells and fog nodes. The nFR24L01 is a transceiver that is used in an open space and with lower baud that its range can reach up to 100 m. It can send 1 to 25 bytes of raw data at the transmission rate of 1 MB and connects to the Arduino through the SPI communication.

5.1.3. Raspberry Pi to nRF240L

Generally, fog nodes have more computing and storage power than fog cells. For the FON, a more robust component is needed that is more powerful than Arduino. We used Raspberry Pi board as FON because it has more computing and storage power; hence, all tasks received from sensors are sent to it. A typical wireless communication module is required to communicate between Arduino boards (connected to sensors) and Raspberry Pi that we have used the nRF240L transceiver.

5.1.4. Raspberry Pi to Smartphone

At the beginning of the connection, an application installed in a smartphone transmits a signal to Raspberry Pi (FON). This signal is sent wirelessly to receive the required data from the smart home since the Raspberry Pi has a built-in WiFi module. Thus, smartphone and Raspberry Pi communicate via Wi-Fi. We have stored some `.php` files in the Raspberry Pi to create the local host and save and processed data.

5.2. SCATTER Implementation in iFogSim

We have implemented our proposed approach in the iFogSim simulator [5]. The fog nodes are modeled according to architecture mentioned earlier in Figure 1. Figure 4 shows the mapping of different smart home case study parts on the iFogSim simulator's classes. In `sensor` class in iFogSim, we defined IoT devices, sensors, actuators, edges, and their properties. In the `FogDevice` class, we have described hardware features, architecture, and connections in computational resources, such as fog devices, fog cells, and cloud. We also consider a fog device as FON that has a database and a local host for storing the processing tasks. The `Controller` class is responsible for the management of resources so that it calculates QoS metrics results, such as network usage, cost of execution in the cloud, the delay between services, and energy consumption of devices, during the simulation.

To evaluate the SCATTER algorithm, we consider five applications (proportional to the number of sensors) with different deadlines that are specified by application users. Table 3 depicts the type, name, and deadline of applications. The configuration of used computational resources in the smart home, such as CPU, RAM, up-level BW, down-level BW, and power, is shown in Table 4. After collecting measurements during a two-week period, we have calculated latencies between various resources (from source to destination), which are shown in Table 5.

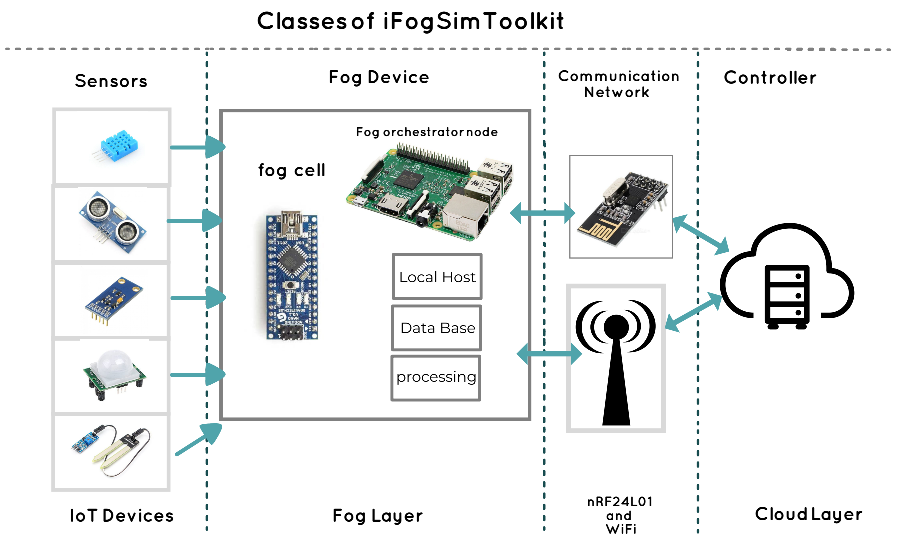

Figure 4. The mapping of different parts of smart home use-case within the iFogSim environment.

Table 3. Application deadlines after a 2-week period experiment.

Application Type	Application Name	D_{A_k} (ms)
Security	a_0	1000
Motion	a_1	600
Light	a_2	4000
Temperature	a_3	8000
Humidity	a_4	5000

Table 4. Configuration of different computational devices.

Device Type	CPU (MIPS)	RAM (MB)	BW to Up-Level (Gbps)	BW to Down-Level (Gbps)	POWER (W)
cloud	20,000	20,000	-	100	107.339
fog node	1000	2048	100	10	107.339
fog cell	250	256	0.15	0.002	107.339

Table 5. Description of network links for Smart Home.

Source	Destination	Latency (msec)
sensor/actuator	fog cell	36
fog cell	FON	124
FON	neighbor FON	829
fog device/FON/ neighbor FON	cloud	3241

To evaluate the performance of the proposed service placement algorithm, we compared the SCATTER approach with two policies: (i) edge-ward and (ii) cloud-only implemented in iFogSim. In the edge-ward approach, placement of the services is near the edge, and, if the devices at the edge of the network do not have enough computational resources to execute the services, searching for situation devices will be hierarchical. Higher-level devices are checked in terms of computational resources to find the appropriate devices for hosting the services. In the edge-ward approach, the priority of applications and tasks, the horizontal connection between same level devices, and clustering fog nodes are not considered, and only fog devices of level n and level $n + 1$ are communicated to each other. In the cloud-only approach, the sensing and actuating services are placed in fog cells, and processing services are executed in the cloud.

6. Evaluation Results

By execution of task placement approaches, we gain QoS metrics, such as application response times, the execution time of simulation, network usage, average application loop delays, and energy consumption. This section aims to show the performance of three algorithms, i.e., SCATTER, edge-ward, and cloud-only, for solving the SPP.

There are some other related works that we were unable to include them in our evaluations due to the lack of source codes and not being able to reproduce the algorithm and results in iFogSim simulator. In Reference [34], the authors proposed a multi-fog based architecture for reducing the latency and network usage. Their algorithm is compared with the edge-ward approach, and the iFogSim simulator is used for evaluation. The simulation results show that latency and network usage in their algorithm are 15% and 9% better than edge-ward approach, while the SCATTER algorithm has improved the delay and network usage parameters by 45% and 22% compared to the edge-ward policy. One of the drawbacks of the algorithm introduced in Reference [34] is the use of traditional fog-cloud hierarchical structure and the fog nodes at the same level have no connection links with each other, which makes most services execute in the cloud, thus reducing algorithm efficiency.

The authors in Reference [35] proposed an energy-aware strategy in order to solve SPP for healthcare systems, and compared their work with edge-ward and cloud-only approaches. Their algorithm places the application services on a fog device that ensures a minimum increment of energy consumption. The policy selects energy-efficient fog devices more frequently than non-energy-efficient ones. The results show that their approach reduces the energy consumed by 1.61% and 2.72% compared to the edge-ward and cloud-only strategies, respectively. Moreover, it minimizes the delay, undercutting the edge-ward and cloud-only strategies by 0.53% and 17.73%. But the network usage in their proposed strategy is approximately equal to that of the edge-ward, while, in the SCATTER, we improved the energy consumption by 2.33% and 3.2%, the delay by 45% and 65%, and the network usage by 22% and 27.8% compared to the edge-ward and cloud-only approaches, respectively. One of the limitations of this work is neglecting different demands of the services. Therefore, only the CPU required for the services is considered. In addition to the CPU, the services also need RAM, BW, and storage.

Ref. [21] proposed a genetic algorithm as a problem resolution heuristic, where its results show that 9 out of 25 services are executed in the cloud. In other words, approximately 36% of services execute in the cloud, while, in the SCATTER, 15% (3 out of 20) of the services execute in the cloud. This fact shows that SCATTER performs better in terms of QoS parameters than the proposed algorithm in Reference [21]. One of limitations of Reference [21] is that it does not prioritize applications and services. In addition, because the SPP problem is solved as an ILP, it is very difficult to implement the algorithm on a large scale.

6.1. Applications Response Times

In real-time applications, the response time must be less than the deadline. The difference between application response time and deadline ($D_{(a_i)} - R_{(a_i)}$) determines the performance of a service placement algorithm. In Table 6, the results of algorithms performance based on $D_{(a_i)} - R_{(a_i)}$ is shown. In addition, Figure 5 depicts the response time of applications and the amount of application deadline violations in three policies. The edge-ward scenario results in deadline violations for the application a_2 at 8457 ms, application a_3 at 9501, and application a_4 at 9240 ms. In the edge-ward, first, the tasks are placed in the corresponding fog node connected to the current fog cell. If the fog node is not able to host a task, it is transmitted to the cloud. Therefore, the response time of applications that their tasks are placed in the cloud will be increased. In the cloud-only scenario, the deadlines for all applications are violated in 7257, 6991, 8457, 9501, and 9240 ms. But, in the SCATTER algorithm, any application deadlines are not broken because most tasks are placed in the fog layer instead of the cloud, which would result in a more extensive communication delay. In the SCATTER algorithm, the average value of response time is 42.1% and 60.21% less than edge-ward and cloud-only, respectively. In SCATTER, only 3 out of 20 tasks are executed in the cloud, and the rest are placed in the fog layer. But, in edge-ward policy, 6 out of 20 tasks are propagated to the cloud.

Table 6. Algorithms performance comparison.

Algorithms	$D_{(a_i)} - R_{(a_i)}$ (ms)					Network Usage (kb)	Execution Time (ms)
	a_0	a_1	a_2	a_3	a_4		
edge-ward	225	91	4460	−1500	−4240	2.73×10^7	5720
cloud-only	−6260	−6390	−4460	−1500	−4240	2.95×10^7	4880
SCATTER	225	91	367	841	584	2.13×10^7	5550

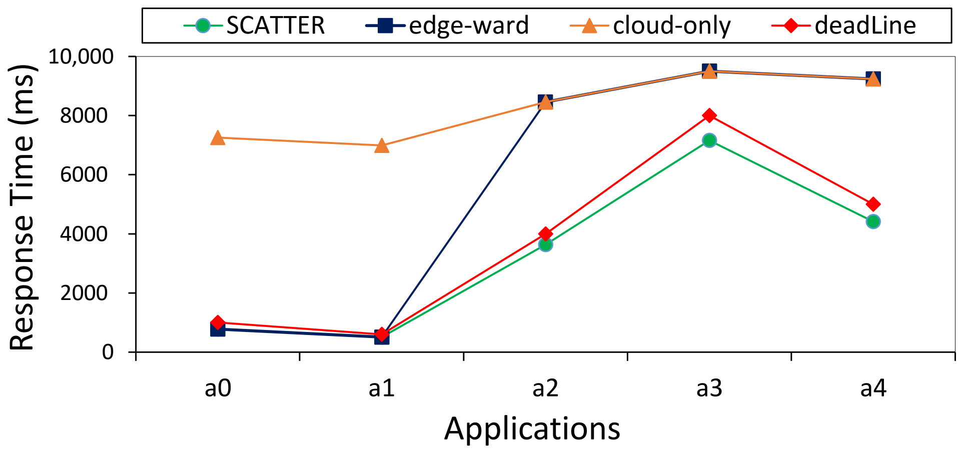

Figure 5. Response times of applications.

6.2. Network Usage

Each IoT application sends a request from the end-user device to higher layers for the execution of tasks. The submitted requests have a total size in bytes and are transmitted to a target device. The network usage of each request is related to the request's overall size and latency between the first device and the target device of the request. The total network usage is calculated as the sum of the network usage of each request during the simulation time. As shown in Table 6, total network usage in the SCATTER algorithm is 21.98% and 27.8% less than edge-ward and cloud-only policies.

6.3. Average Application Loop Delays

The application loop delay is delay time to execute the path of independent tasks in an application. We analyzed the average loop delay for each application in three algorithms. As shown in Figure 6, it is observable that SCATTER has a lower average delay in contrast to both edge-ward and cloud-only. In cloud-only, all processing tasks are executed in cloud data centers. Therefore, loop delays in all applications are more than other both policies. In both edge-ward and SCATTER, the applications a_0 and a_1 are placed in the fog layer, and their average loop delays are low. In edge-ward, applications a_2, a_3, and a_4 are executed in the cloud layer, and delays are increased. In SCATTER, only three tasks are placed in the cloud, and the rest of the tasks are hosted in the fog layer; therefore, loop delays in applications are lower than both edge-ward and cloud-only.

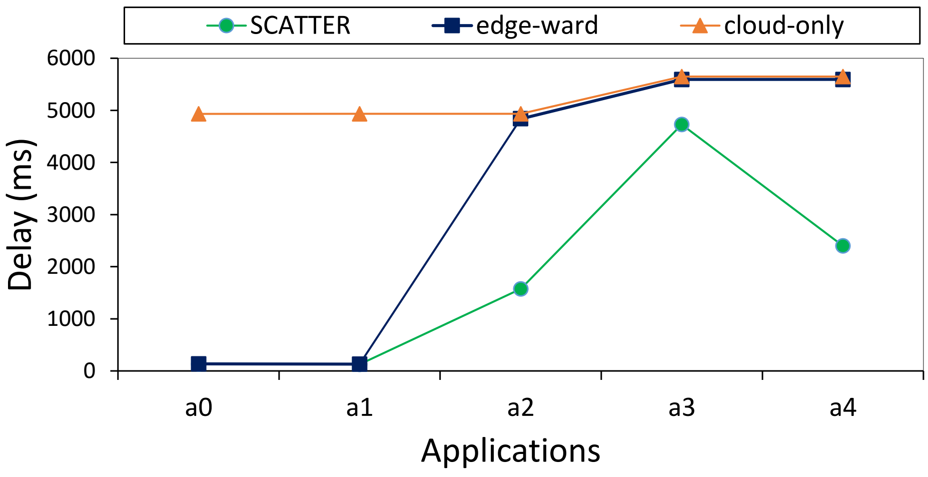

Figure 6. Average Applications Loop Delays.

6.4. Energy Consumption

Energy consumption is the sum of energy consumed by fog nodes, fog cells, and the cloud for the task placement. Figure 7 depicts the total energy consumption for all computational resources in edge-ward, cloud-only, and SCATTER algorithms. In three algorithms, sensing and actuating tasks are placed in fog cells; therefore, energy consumption in fog cells in three algorithms is the same value. In the SCATTER algorithm, the fog devices' energy consumption is more than in edge-ward and cloud-only algorithms, because most processing tasks are executed in the fog layer. Although in edge-ward and cloud-only approaches, the energy consumption in the cloud is more than the SCATTER algorithm. Because most processing tasks are propagated in the cloud, the SCATTER algorithm's total energy consumption is 2.33% and 3.20% less than edge-ward and cloud-only approaches, respectively.

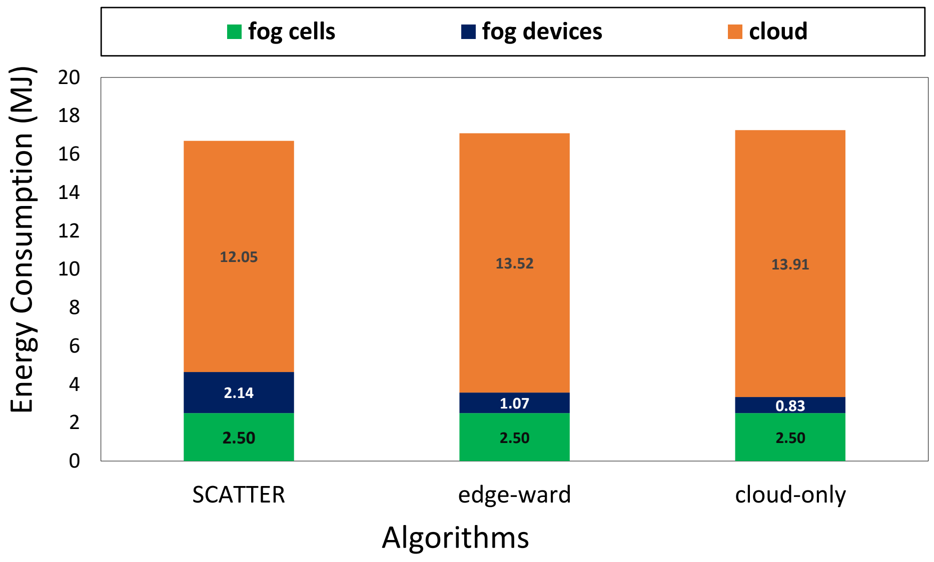

Figure 7. Energy consumption of devices in three algorithms.

7. Conclusions and Future Works

In this paper, we have proposed SCATTER algorithm based on integrated fog-cloud environments to solve the SPP. In many application domains, it is mandatory to guarantee time bounds to support real-time applications. It means that, if the response time of the application exceeds a certain threshold, a failure will happen in the system. In the proposed algorithm, we have focused on latency-sensitive applications and more processing tasks are hosted by fog nodes. To achieve this goal, we clustered the fog nodes and considered horizontal connections between fog nodes instead of traditional hierarchical architecture. Our approach has been validated by a case study of smart home and evaluated in fog-cloud computing environment using iFogSim toolkit. We compared the SCATTER algorithm with two policies, which were previously built-in the simulator (edge-ward and cloud-only) in terms of QoS metrics. Experimental results demonstrated that our approach compared to edge-ward and cloud-only has 42.1% and 60.21% less response time, 21.98% and 27.8% less network usage, 45.01% and 65.67% less loop delay, and 2.33% and 3.20% less energy consumption.

We intend to expand the SCATTER algorithm and consider mobility of devices in the network for our future work. In this work, we assumed that all devices have no movement, while, in the real world, there is a high probability of employing mobile nodes. We are also aiming to consider device battery lifetime in our future work. Each computational resource consumes battery when it hosts and executes the tasks and even in the idle mode. If many tasks are performed on a computational device, and the other computational device is idle most of the time, then the first device may have a problem with the lack of battery and even being shut down. Due to the lack of computational resources, we considered a small scale experiment with a small number of fog devices for our tests. The networks in the real-world have a more complex structure; thus, nodes may be distributed in a rather larger area with heterogeneous network topology. In our future work, we intend to implement an optimization algorithm for large number of fog nodes and services in a large-scale environment.

Author Contributions: Conceptualization, F.F.-G. and H.F.; Data curation, F.K.; Methodology, F.K. and F.F.-G.; Project administration, H.F.; Software, F.K.; Writing–original draft, F.K.; Writing–review & editing, F.F.-G. and H.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Swedish Research Council (Vetenskapsrådet), through the MobiFog starting grant, and by the Swedish Knowledge Foundation (KKS) through the FlexiHealth Prospekt, and the EU Celtic_Next/Vinnova project, Health5G (Future eHealth powered by 5G).

Data Availability Statement: The data presented in this study are available on request from the corresponding authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Aazam, M.; Khan, I.; Alsaffar, A.A.; Huh, E.N. Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In Proceedings of the 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST), Islamabad, Pakistan, 14–18 January 2014; pp. 414–419.
2. Nan, Y.; Li, W.; Bao, W.; Delicato, F.C.; Pires, P.F.; Zomaya, A.Y. Cost-effective processing for delay-sensitive applications in cloud of things systems. In Proceedings of the 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 31 October–2 November 2016; pp. 162–169.
3. Tran, M.Q.; Nguyen, D.T.; Le, V.A.; Nguyen, D.H.; Pham, T.V. Task placement on fog computing made efficient for iot application provision. *Wirel. Commun. Mob. Comput.* **2019**, *2019*, 6215454.
4. Ren, C.; Lyu, X.; Ni, W.; Tian, H.; Song, W.; Liu, R.P. Distributed Online Optimization of Fog Computing for Internet-of-Things under Finite Device Buffers. *IEEE Internet Things J.* **2020**, *7*, 5434–5448.
5. Gupta, H.; Vahid Dastjerdi, A.; Ghosh, S.K.; Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Softw. Pract. Exp.* **2017**, *47*, 1275–1296.
6. Stojmenovic, I. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In Proceedings of the 2014 Australasian Telecommunication Networks and Applications Conference (ATNAC), Southbank, VIC, Australia, 26–28 November 2014; pp. 117–122.
7. Yangui, S.; Ravindran, P.; Bibani, O.; Glitho, R.H.; Hadj-Alouane, N.B.; Morrow, M.J.; Polakos, P.A. A platform as-a-service for hybrid cloud/fog environments. In Proceedings of the 2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Rome, Italy, 13–15 June 2016; pp. 1–7.

8. Maleki, N.; Loni, M.; Daneshtalab, M.; Conti, M.; Fotouhi, H. Sofa: A spark-oriented fog architecture. In Proceedings of the IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society, Lisbon, Portugal, 14–17 October 2019; Volume 1, pp. 2792–2799.
9. Mahmud, R.; Kotagiri, R.; Buyya, R. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 103–130.
10. Mahmud, R.; Buyya, R. Modelling and simulation of fog and edge computing environments using iFogSim toolkit. In *Fog and Edge Computing: Principles and Paradigms*; Wiley: Hoboken, NJ, USA, 2019; pp. 1–35.
11. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16.
12. Yi, S.; Qin, Z.; Li, Q. Security and privacy issues of fog computing: A survey. In Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications, Qufu, China, 10–12 August 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 685–695.
13. OpenFog Consortium Architecture Working Group. OpenFog reference architecture for fog computing. In *OPFRA001*; OpenFog Consortium: Fremont, CA, USA, 2017; pp. 1–162.
14. Pallewatta, S.; Kostakos, V.; Buyya, R. Microservices-based IoT Application Placement within Heterogeneous and Resource Constrained Fog Computing Environments. In Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, Auckland, New Zealand, 2–5 December 2019; pp. 71–81.
15. He, J.; Wei, J.; Chen, K.; Tang, Z.; Zhou, Y.; Zhang, Y. Multitier fog computing with large-scale iot data analytics for smart cities. *IEEE Internet Things J.* **2017**, *5*, 677–686.
16. Silva, R.; Silva, J.S.; Boavida, F. Opportunistic fog computing: Feasibility assessment and architectural proposal. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; pp. 510–516.
17. 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*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 169–186.
18. Gill, S.S.; Garraghan, P.; Buyya, R. ROUTER: Fog enabled cloud based intelligent resource management approach for smart home IoT devices. *J. Syst. Softw.* **2019**, *154*, 125–138.
19. Rahman, G.; Chuah, C.W. Fog computing, applications, security and challenges, review. *Int. J. Eng. Technol.* **2018**, *7*, 1615–1621.
20. Puthal, D.; Mohanty, S.P.; Bhavake, S.A.; Morgan, G.; Ranjan, R. Fog computing security challenges and future directions [energy and security]. *IEEE Consum. Electron. Mag.* **2019**, *8*, 92–96.
21. Skarlat, O.; Nardelli, M.; Schulte, S.; Borkowski, M.; Leitner, P. Optimized IoT service placement in the fog. *Serv. Oriented Comput. Appl.* **2017**, *11*, 427–443.
22. 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 Things J.* **2019**, *6*, 5080–5096.
23. Al Faruque, M.A.; Vatanparvar, K. Energy management-as-a-service over fog computing platform. *IEEE Internet Things J.* **2015**, *3*, 161–169.
24. Rahbari, D.; Nickray, M. Task offloading in mobile fog computing by classification and regression tree. *Peer-to-Peer Netw. Appl.* **2020**, *13*, 104–122.
25. Skarlat, O.; Karagiannis, V.; Rausch, T.; Bachmann, K.; Schulte, S. A framework for optimization, service placement, and runtime operation in the fog. In Proceedings of the 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), Zurich, Switzerland, 17–20 December 2018; pp. 164–173.
26. Bala, M.I.; Chishti, M.A. Optimizing the Computational Offloading Decision in Cloud-Fog Environment. In Proceedings of the 2020 International Conference on Innovative Trends in Information Technology (ICITIIT), Kottayam, India, 13–14 February 2020; pp. 1–5.
27. Lu, H.; Gu, C.; Luo, F.; Ding, W.; Liu, X. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Gener. Comput. Syst.* **2020**, *102*, 847–861.
28. Cai, P.; Yang, F.; Wang, J.; Wu, X.; Yang, Y.; Luo, X. JOTE: Joint Offloading of Tasks and Energy in Fog-Enabled IoT Networks. *IEEE Internet Things J.* **2020**, *7*, 3067–3082.
29. Wei, H.; Luo, H.; Sun, Y. Mobility-Aware Service Caching in Mobile Edge Computing for Internet of Things. *Sensors* **2020**, *20*, 610.
30. Misra, S.; Bera, S. Soft-VAN: Mobility-Aware Task Offloading in Software-Defined Vehicular Network. *IEEE Trans. Veh. Technol.* **2019**, *69*, 2071–2078.
31. Rejiba, Z.; Masip-Bruin, X.; Marin-Tordera, E. A user-centric mobility management scheme for high-density fog computing deployments. In Proceedings of the 2019 28th International Conference on Computer Communication and Networks (ICCCN), Valencia, Spain, 29 July–1 August 2019; pp. 1–8.
32. Mseddi, A.; Jaafar, W.; Elbiaze, H.; Ajib, W. Joint container placement and task provisioning in dynamic fog computing. *IEEE Internet Things J.* **2019**, *6*, 10028–10040.
33. Optimizer, I. Available online: http://https://www.ibm.com/nl-en/analytics/cplex-optimizer (accessed on 17 February 2021).
34. Gavaber, M.D.; Rajabzadeh, A. MFP: An approach to delay and energy-efficient module placement in IoT applications based on multi-fog. *J. Ambient. Intell. Humaniz. Comput.* **2020**, *12*, 1–17.

35. Mahmoud, M.M.; Rodrigues, J.J.; Saleem, K.; Al-Muhtadi, J.; Kumar, N.; Korotaev, V. Towards energy-aware fog-enabled cloud of things for healthcare. *Comput. Electr. Eng.* **2018**, *67*, 58–69.
36. Mirghani, S.; Hajjdiab, H. Comparison between Amazon S3 and Google Cloud Drive. In Proceedings of the 2017 2nd International Conference on Communication and Information Systems, Wuhan, China, 7–9 November 2017; pp. 250–255.
37. Varghese, B.; Wang, N.; Nikolopoulos, D.S.; Buyya, R. Feasibility of fog computing. In *Handbook of Integration of Cloud Computing, Cyber Physical Systems and Internet of Things*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 127–146.
38. Dang, T.D.; Hoang, D. A data protection model for fog computing. In Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, Spain, 8–11 May 2017; pp. 32–38.
39. Mechalikh, C.; Taktak, H.; Moussa, F. A Scalable and Adaptive Tasks Orchestration Platform for IoT. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 1557–1563.
40. Mechalikh, C.; Taktak, H.; Moussa, F. PureEdgeSim: A simulation toolkit for performance evaluation of cloud, fog, and pure edge computing environments. In Proceedings of the 2019 International Conference on High Performance Computing & Simulation, Dublin, Ireland, 15–19 July 2019; pp. 700–707.
41. Li, C.; Zhuang, H.; Wang, Q.; Zhou, X. SSLB: Self-similarity-based load balancing for large-scale fog computing. *Arab. J. Sci. Eng.* **2018**, *43*, 7487–7498.
42. Yang, Y.; Liu, Z.; Yang, X.; Wang, K.; Hong, X.; Ge, X. POMT: Paired offloading of multiple tasks in heterogeneous fog networks. *IEEE Internet Things J.* **2019**, *6*, 8658–8669.
43. Goudarzi, M.; Wu, H.; Palaniswami, M.S.; Buyya, R. An application placement technique for concurrent iot applications in edge and fog computing environments. *IEEE Trans. Mob. Comput.* **2020**, *20*, 1298–1311.
44. Wang, D.; Liu, Z.; Wang, X.; Lan, Y. Mobility-Aware Task Offloading and Migration Schemes in Fog Computing Networks. *IEEE Access* **2019**, *7*, 43356–43368.
45. Yang, Y.; Luo, X.; Chu, X.; Zhou, M.T., Fog-Enabled Smart Home and User Behavior Recognition. In *Fog-Enabled Intelligent IoT Systems*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 185–210.
46. Guerrero, C.; Lera, I.; Juiz, C. A lightweight decentralized service placement policy for performance optimization in fog computing. *J. Ambient. Intell. Humaniz. Comput.* **2019**, *10*, 2435–2452.