Received: 27 December 2021 | Revised: 28 August 2022 | Accepted: 1 September 2022

DOI: 10.1002/ett.4650

RESEARCH ARTICLE

WILEY

Reinforcement optimization for decentralized service placement policy in IoT-centric fog environment

Hamza Sulimani[1] | **Akbar Muhammad Sajjad**[1] | **Wael Y. Alghamdi**[2] | **Omprakash Kaiwartya**[3] | **Tony Jan**[4] | **Simeon Simoff**[5] | **Mukesh Prasad**[1]

[1]School of Computer Science, FEIT, University of Technology Sydney, Sydney, New South Wales, Australia

[2]Faculty of Computer Science and Information Technology, Ta'if University, Ta'if, Saudi Arabia

[3]School of Science and Technology, Nottingham Trent University, Nottingham, UK

[4]Artificial Intelligence and Optimization Research Centre, Faculty of Design and Creative Technology, Torrens University, Sydney, New South Wales, Australia

[5]School of Computer, Data and Mathematical Sciences, Western Sydney University, Sydney, New South Wales, Australia

Correspondence
Mukesh Prasad, School of Computer Science, FEIT, University of Technology Sydney, Sydney, New South Wales, Australia.
Email: mukesh.prasad@uts.edu.au

Abstract

A decentralized service placement policy plays a key role in distributed systems, such as fog computing, where sharing workloads fairly among active computing nodes is critical. A decentralized policy is an inherent feature of the service placement process that may improve load balancing among computers and can reduce the latency in many real-time Internet of Things (IoT) applications. This article proposes reinforcement optimization for a decentralized service placement policy, which attempts to mitigate some of the drawbacks of existing service placement policies. Matching task size with node specifications and the allocation of less popular but time-sensitive applications in the fog layer are the primary contributions of this study. Extensive experimental comparisons are made between the proposed algorithm and other well-known algorithms over service latency, network usage, and computing usage using the iFogSim simulator. A microservice-based application with varying sizes of computing requests are tested experimentally and show that the proposed algorithm effectively serves computing instances that are closer to users, reducing service latency and network usage. Compared to the existing models, the proposed modified algorithm reduces service latency by 24.1%, network usage by 4%, and computing usage by 20%, thus highlighting positive outcomes when using the proposed algorithm for fog analytics in future real-time IoT applications.

1 | INTRODUCTION

The popularity of smart cities, wearables, e-health, and smart vehicle environments has been increased by the emerging applications of the Internet of Things (IoT).[1-3] According to the report of McKinsey Global Institute,[4] the IoT applications will facilitate $11 trillion economy by 2025. Quality assurance in IoT applications is of paramount importance due to the rapidly increasing number of human users and their future service requirements. For the quality assurance of IoT applications, cloud servers were expected to integrate IoT technologies with human users.[2,5,6] The cloud-centric IoT network has been designed to alleviate the constraints of IoT devices, which are due to the limitations of computational and

memory resources, by providing centralized control of IoT devices.[5,7,8] Unfortunately, cloud servers are often located far from IoT edge devices, thus introducing latency and bandwidth challenges between users and computing nodes.[9-12] Latency in time-sensitive applications, such as smart grids or unmanned vehicle management can result in major problems.[13-16] A fog network can use the available local computing and storage resources near the IoT edge devices and can thus provide necessary computations to the IoT edge devices, providing an opportunity to reduce the latency and bandwidth requirements due to being closer to IoT edge devices than cloud servers.[17-20]

Cloud servers and fog networks provide alternative options for IoT edge devices to access computing resources. A fog data manager (FDM) can determine whether the data are executed either in the fog network or cloud servers. The FDM can transmit part of the data to the cloud servers, while the remaining data are retained on the fog network, as discussed by Puliafito.[21] Once the data are allocated to the fog network, the placement policy (PP) can optimize the resource allocation within the fog network to maximize the use of computing nodes within the fog network. Fog service orchestrators (FSOs) can also fairly allocate computing resources among the fog nodes to improve the quality of service (QoS), as presented by Baranwal.[22] However, both of these methods do not address scalability issues in an IoT network well, and Salaht and Desprez[23] addressed scalability in their fog service placement problem (FSPP). An efficient FSPP is vital to assure the QoS of IoT applications. A failed or suboptimal FSPP can results in delay in fog analytics causing major issues to the real-time IoT applications.

FSPP is important, but its current centralized practice have some disadvantages: (a) increased network overhead due to the periodic communications between the resource broker and fog node; (b) longer computing time due to the increased number of IoT edge devices to control; (c) reduced reliability to a single point of failure (SPOF); (d) management of heterogeneous fog devices; and (e) inherent latency generated by machine communications between brokers and fog devices. However, the decentralized scheduling algorithm (DSA) proposed by Desprez and Salaht[24] relies on the time vector for efficient operation. A poorly managed FSPP can cause real-time IoT applications to fail due to the "induced" high latency.[25]

This study reinforces the current placement policy FSPP using a proposed new policy to overcome some of aforementioned shortcomings. This study proposes an innovative FSPP that operates in a distributed manner to reduce the induced and inherent latencies in IoT applications that prioritizes delay-sensitive applications to be served with priority placement even with a low popularity rate. The father node, which receives the computing tasks from the end-user nodes, can forward the services or maintain the services if the father nodes are compatible. All tasks are managed locally by the father node and its subordinate nodes without global scheduling. Such an FSPP is more scalable and not affected by the increasing number of IoT edge devices or services. This study also performs several experiments to confirm the above hypothesis. The primary contribution of this study is a more scalable FSPP that also reduces computing requirements while reducing both inherent and induced delays in time-sensitive IoT applications. The new FSPP may limit global communication between fog nodes, therefore not guaranteeing global optimization; however, as the size of the IoT network increases, global optimization is often not necessary.[25,26] In this approach, the proposed method places the most popular or delay-sensitive compute tasks closer to the end-user nodes using the hop number as the reference. This method manages the workload for fog nodes to remain sustainable with locally optimized QoS, thereby satisfying the necessary computing requirements of IoT applications in the fog network. The primary objective of this study is to develop the decision criteria for service allocation, including when and where the services are to be allocated among the IoT fog nodes, and associated time-sensitive requirements (either closer or further from the end-user nodes).

2 | RELATED WORKS

Fog networks can use many optimization techniques, including greedy algorithms, heuristics, genetic algorithms, and linear programming. In References 27-31, several aspects of fog resource management are defined, including scheduling, placement, provisioning, allocation, and mapping for clients, virtual machines, services, and resources. These algorithms have also been investigated for real-time applications, such as smart cities, industrial IoT, and mobile microclouds.[32-35] Table 1 summarizes recent work, including application types (eg, eHealth IoT system, general IoT system, or embedded system) in the column *Scope*, types of brokers (*Broker*), proposed algorithms (*Alg.)*, target functions to optimize (*Objective functions*), components that the optimization algorithm can control to enhance the objective functions (*Decision variables*), validation tools used to test the algorithms, and types of computing environments (*Env.)*.

TABLE 1 Summary of approaches to service placement

Authors	Scope	Broker	Alg.	Objective function	Decision variables	Val.	Env.
Khoshroabadi et al[13]	S	C-FON	SCATTER	QoS, response times, network usage, energy consumption, application loop delays	App. sensitivity	F, E	F
Velasquez et al[22]	G	C(Os)	PRP	Latency, jitter of app., application profile	Popularity of app	Y	CF
Morkevicius et al[23]	G	H(o3)	TSM	QoS, security, CPU performance, storage, energy consumption	Integer multiobjective particle swarm optimization	M	CF
Hassan et al[24]	S	C-FRM	MinRE	Response time energy consumption, resource usage	App. sensitivity	O	CF
Salimian et al[2]	S	H	GWO	Execution cost, average waiting time	Matching the task with node in the colony	M	CF
Baranwal et al[25]	G	D(Os)	FONs	Network relaxation ratio, processing time reduction	Enhance the selection of fog orchestrator nodes	M	CF
Zeng et al[18]	E	Ds	H	Min and max response time	Minimize computation and I/O time	O	F
Jamil et al[32]	H	C	SJF	Energy efficiency delay	Computing time, network usage	F	CF
Maiti et al[28]	G	H	SGAP	Average make span average schedule length	Latency	M	F
Al-Tarawaneh et al[27]	G	Ds	Bi-Obj	Application performance, energy efficiency	Sensitivity and security	F	F
Taneja et al[36]	G	Ds	MM	QoS	Use the border nods	F	CF
Deng et al[37]	G	Ds	H	Energy consumption delay	Save communication bandwidth	Mt	CF
Wang et al[38]	G	D	H	Resource utilization	Use the boarder nodes	O	ME
Guerrero et al[33]	G	D	POP	QoS, cost, execution time	Popularity of service	F	CF
Yang et al[17]	G	Ds	CSPP	Hop count, CPU, network	Matching the task with node	E	ME
Arora et al[31]	S	C	HSMF	Network usage	Popularity of services	F	CF
Sami et al[14]	S	C	IFSP	QoS, cost, execution time	Proactivity service	O	CF
The proposed method	G	D	RODSPP	Hop count, network usage	Popularity of service, task sensitivity	F	CF

Abbreviations: C, IoT-camera; E, embedded systems; G, general IoT system; H, eHealth IoT system; S, IoT smart home application (***Scope***). C, centralized; D, decentralized; Ds, distributed; H, hierarchical, O, orchestrators (***Broker***). Bi-Obj, bi-objective; CSPP, cost-aware service placement policy; EPOP, enhancement of placement optimization policy; FONs, fog orchestrator nodes; H, heuristic algorithm; HSMF, heterogeneous shortest module first; IFSP, intelligent fog and service placement; MM, module mapping; PoP, placement optimization policy; PRP, popularity rank placement; SCATTER, clustering of fog devices and requirement-sensitive service first; SGAP, schedule GAP's; TSM, two stage method (***Algorithm*** (***Alg.***)). E, experiment; F, iFogSim; M, Matlab; Mt, mathematically; O, their own simulation program; Y, YAFS (***Validation*** (***Val.***)). CF, cloud/fog; F, fog computing; ME, mobile edge-clouds (***Environment*** (***Env.***)).

Khosroabadi and Fotouhi-Ghazvini[5] proposed "a clustering of fog devices and requirement-sensitive services first" (SCATTER) algorithm to allocate the fog-edge border computing resources to delay-sensitive applications but ignores the other tasks in demand, which could be a concern in many real-life applications.[39] introduced popularity ranked placement (PRP) based on graph partitions and optimization using genetic algorithms, and showed that PRP yielded improvements compared to the generic genetic algorithm or first fit (FF) algorithm but could still be improved. Morkevicius and Venčkauskas[40] presented two stages of multiobjective optimization that included multiple metrics that we use in this study to evaluate the proposed method: security performance, compute usage performance, and storage utilization. This technique was designed to maintain the QoS level with the minimum usage of available resources.

QoS and energy consumption metrics were used to evaluate the algorithms proposed by Hassan et al[38]; Kamel[19]; and Puliafito.[21] Those authors attempted to enhance the service placement policy in a cloud/fog environment, ultimately improving QoS in delay-sensitive applications. Hassan and Azizi[38] proposed a MinRE placement policy to enhance QoS and energy consumption in a cloud/fog ecosystem. MinRE consists of two algorithms, MinRes and MinEng, where each focuses on different types of workload. During experimentation, MinRE showed promising results, even with a fixed number of fog nodes, while increasing computing loads. Baranwal and Vidyarthi[36] used distributive fog orchestrator nodes (FONs) to place services on a fog-integrated cloud. The FON, a mediator entity, was proposed by Al-Tarawneh to assign the arrival workload to the fog computational node (FCN). Al-Tarawneh[41] proposed a biobjective placement algorithm to enhance the placement policy for interoperated services in a fog network. That study considered the security requirements and criticality of the application as optimization variables, and improved the satisfaction metrics compared to other policies, including edge-affinity and cloud-only.

Maiti[42] designed a service placement policy to use schedule gaps. Their proposed policy aimed to minimize the makespan to meet the task deadlines with less utilization of communication resources. In Reference 43, an intelligent fog and service placement (IFSP) was proposed to proactively place services on demand using deep reinforcement learning (DRL) hosted in the cloud to predict the system's load expectations with the entire database. The IFSP can thus prepare nodes predictively before bearing workloads. Salimian[37] proposed autonomous IoT service placement to reduce the execution costs of a distributed fog system using the gray wolf optimization (GWO) scheme, which outperformed comparable policies in finding suitable fog nodes to allocate computations. For heterogeneous cloud/fog environments,[44] presented a heterogeneous shortest module first (HSMF) placement policy. The HSMF was based on finding the shortest module to serve first, which yielded improved performance compared to the other policies. Wang[38] introduced a job allocation methodology for interdependent services in a fog network, where interdependent jobs rely on internode data communication and were described using a directed acyclic graph (DAG). The Wang model[38] was limited because it did not consider the deadline constraint and workflow execution. A linear approach (first comes, first serviced) has been widely used, but task completion in ascending order is more time-efficient.[34] However, sorting can be resource-intensive in terms of time and computing.

The decentralized service provisioning introduced by Guerrero[45] produced promising outcomes, such that all latencies in data transmission, decision processing, and return response were reduced, as well as achieving fair load balancing. However, this system still has a challenge due to its high running cost. Random service allocation was discussed by Souza[46] but resulted in a high service delay due to poor load balancing. Huang[47] proposed an energy-efficient approach to the idea of co-locating service placement, which was adequate for a small network but achieved poor service placement in a larger network. A similar approach by Wang et al[48] reduced execution costs by placing predictive modeling tasks. The proposed solution performed well, but the cost of the system was high, making it unaffordable to a small company. Taneja and Davy[49] proposed a resource-aware service placement solution, a good approach for QoS optimization, that first considered the highest-demand application tasks. Thus, the mechanism resulted in a swamped network with idle and waiting states with low-capacity devices/nodes.

3 | PROPOSED ARCHITECTURE AND ALGORITHM

3.1 | Architecture

In real applications, a fog network is an extension of a cloud network. The extension of the decentralized, remote distribution of the cloud network can be considered a fog network. The computing nodes within the fog network are equipped with limited storage capacity and computing power for host instances, and can thus be considered cloudlets.[50] Thus, a

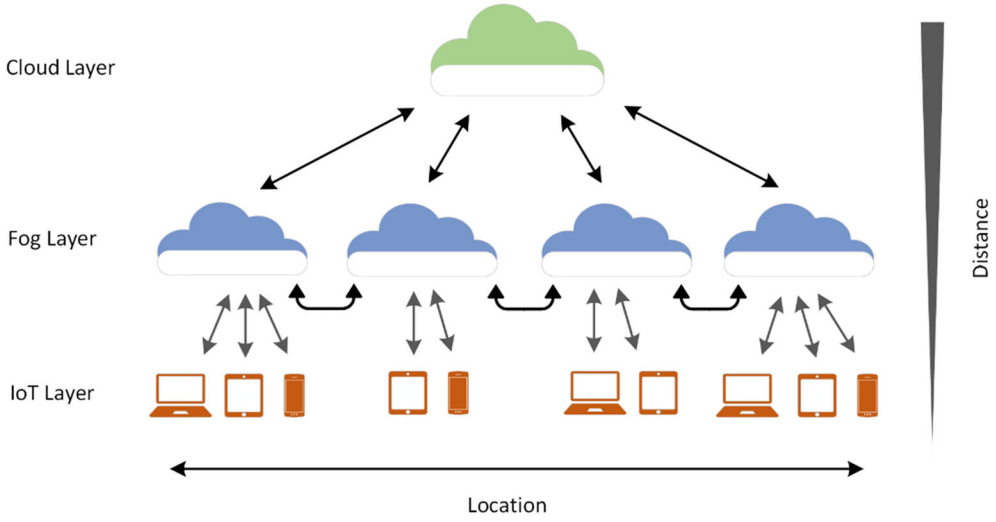

FIGURE 1 Fog architecture

resource management policy for the IoT network can determine where and when to place computing instances among the different layers of computing nodes (eg, fog or cloud). This article proposes an innovative service allocation architecture that alleviates the drawbacks of FSPP, particularly for time-sensitive applications.

Given the three layers of the IoT network (cloud, fog, and edge) as shown in Figure 1, the top layer is the cloud layer that contains the primary cloud servers. The access layer is the edge layer, where end devices manage requests for IoT applications. Edge devices can include computers, smart cars, sensors, and smart homes.[51,52] The fog nodes connect to both the edge devices and cloud servers. This study assumes that all applications in an IoT network use microservices.[53] These applications were configured as a group of stateless and trivial services. These small services complete a complex task once executed in a sequence. For example, app1 requires s_1, s_4, and s_6 to perform its complex task, while app_2 requires s_1, s_2, s_3, and s_5, where the system consists of N services, as shown in Figure 2. Both applications must complete a specific number of jumps to perform app_1 and app_2. Thus, each computing node can be scaled up and down in their instances to improve the QoS of the distributed fog nodes. The scaling process can use service codes from the cloud servers.

This article proposes an optimization of service placement by (1) allocating the most demanded services and all time-sensitive applications in a particular area in the fog nodes that are closer to the clients (edge); (2) migrating the services that are not used regularly to reserve the available resources for other priority services; and (3) managing service allocations to prevent system overloads. For example, if a large computing task is allocated to a less powerful device, an overload will occur in the system. Due to resource limitations in fog nodes, the service placement policy must select services that are to be migrated. The proposed algorithm allocates priority services to the nearest nodes in the shortest path and migrates the lowest nonsensitive requests to the upper levels away from the users and closer to the cloud.

System latency is inherent in all IoT applications. Although the service placement policy can reduce latency in some fog applications, a delay-sensitive application demands further reduced latency to perform adequately in a time-critical application. For example, a device for a stroke patient must respond within near-zero seconds, where any delay in the reporting system can have marked consequences for the patient.[54] The metadata of service requests must be evaluated in priority assignment; thus, such critical tasks can be allocated near edge nodes to support real-time requirements.

Many IoT applications perform interrelated services, where the cloud server becomes essential as a long-term solution. The migration of such services along the shortest path to the cloud is essential. The migration of some services can increase the total execution time (and thus latency) if some essential services do not migrate through the shortest path.[27] For example, an algorithm finds a service S_x with the highest demand for device D_1. Unfortunately, due to the limitation of $D_1's$ resources, s_2, which is selected with the lowest popularity in D_1 must migrate to neighboring devices as shown in Figure 3. Although D_3 has a short distance from the source device, the number of hops increases by two for D_2, which is located on the shortest path to the cloud. To accomplish this task, s_4 was required.

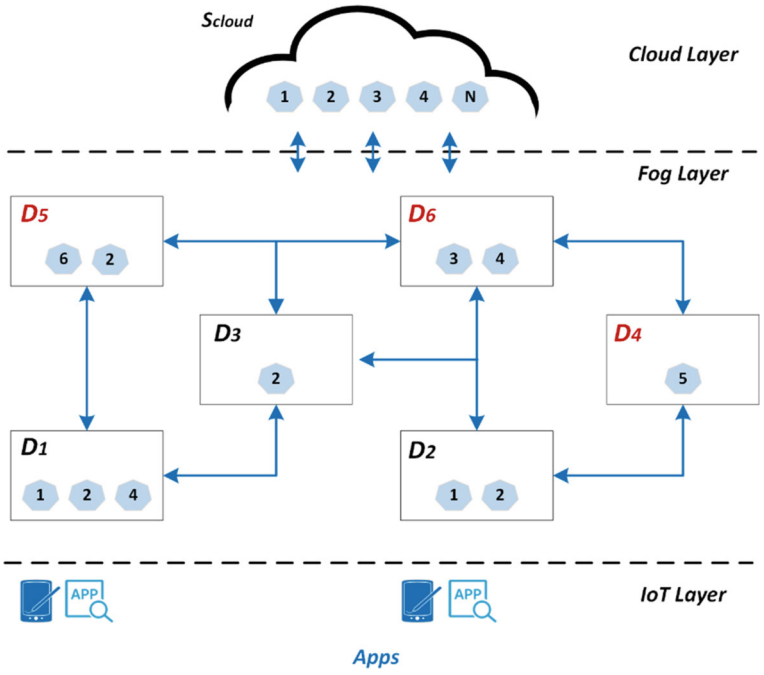

FIGURE 2 Interoperated services and applications

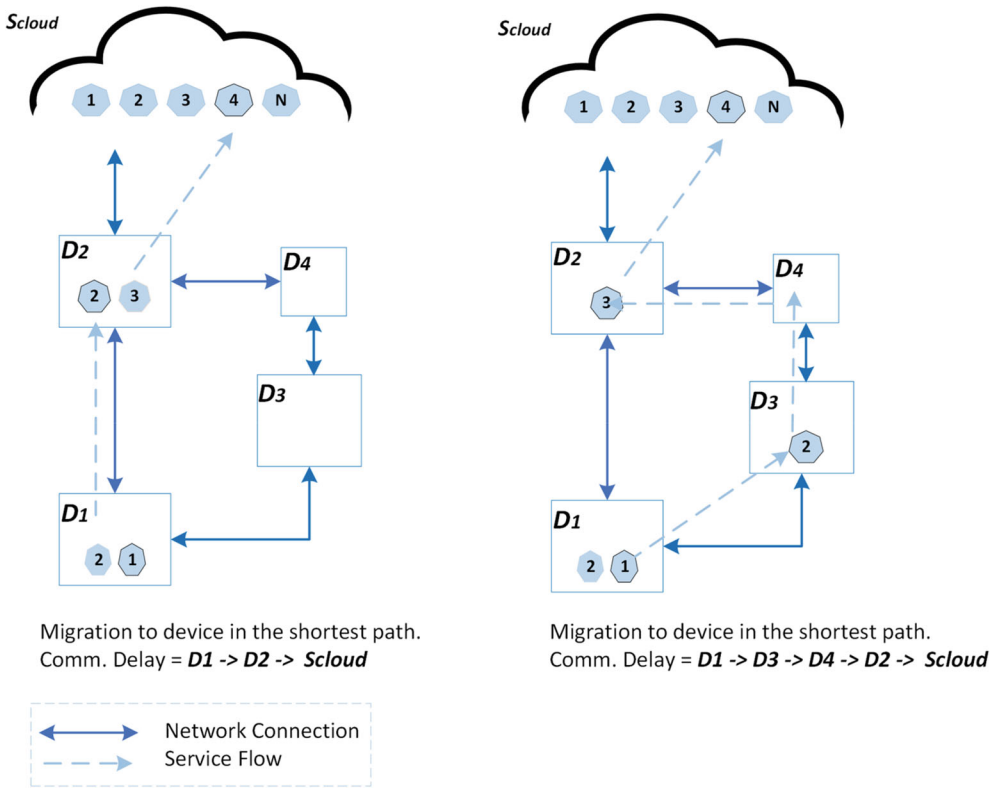

FIGURE 3 Example of service migration within different paths

Some services are in high demand (ie, popular), while others are in low demand. The popularity rule works by migrating high-demand services to fog nodes that are closer to the client (edge), while other low-demand services are allocated to the upper nodes with the shortest path to the cloud servers.[3]

The interoperated service is where a clump of semi-related microservices is combined to accomplish a complex task. Thus, it is critical to migrate all interoperated services (if possible) if one of them must be migrated; this action tends to minimize the number of hops. Figure 4 shows the idea of partially migrating the interoperated services. We assume that the service execution flow for an application is $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5$. Due to the limitations of D_1's computing resources, s_3, which has the lowest request rate in D_1, must be migrated to D_2, which resides at the upper level closer to the cloud and away from the edge. Two additional hops will be added to the application if the placement management system (PMS) does not migrate to the next interoperated services, S_4 and S_5. Thus, it is essential to verify the migration of S_{n+1} and higher, and keep S_{n-1} at the same node. If we assume that S_n is the migrated service, as shown in Equation (1), the PMS must maintain the migration process for the $(n+1)$th interoperated service along the shortest path to the cloud while keeping the $(n-1)$th in the initial node D_1:

$$migration_{s_n} = \bigwedge_{x=n}^{x \rightarrow \infty} s_x \forall x \in interoperated\ services. \tag{1}$$

This strategy must be activated if the initial node has limited resources. The computing node uses a decentralized service broker in the proposed system, which has proven beneficial against a centralized management system.[55]

A decentralized broker is used to implement the proposed strategy. The modeling and characterization parameters were obtained from Reference 3 but with modifications. The modification is made using the service placement request manager (SPRM). A matching code is used in the SPRM to add more restrictions to accept the tasks or migrate them to the higher layer. Therefore, the computing nodes accept workloads by matching their sizes with node specifications. Conversely, Figure 5 shows the proposed broker, which keeps the remaining components of the broker having similar functions as service manager (SM), service usage monitor (SUM), and service popularity monitor (SPM).[33]

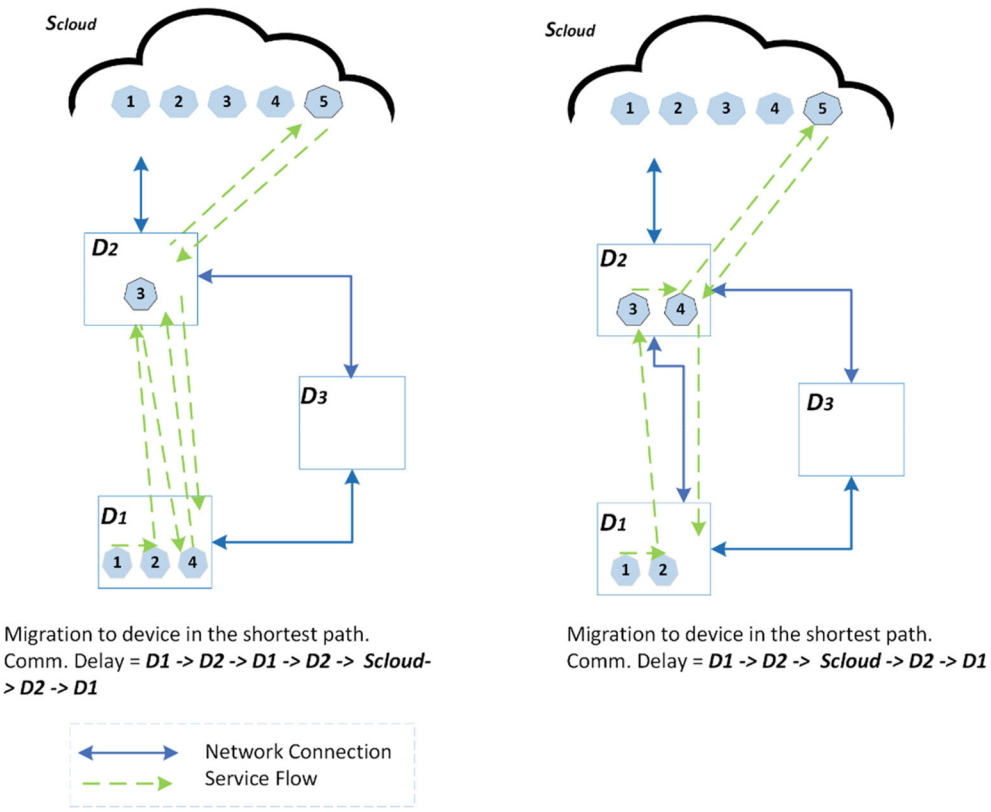

FIGURE 4 Example of interoperated service migration within the shortest path

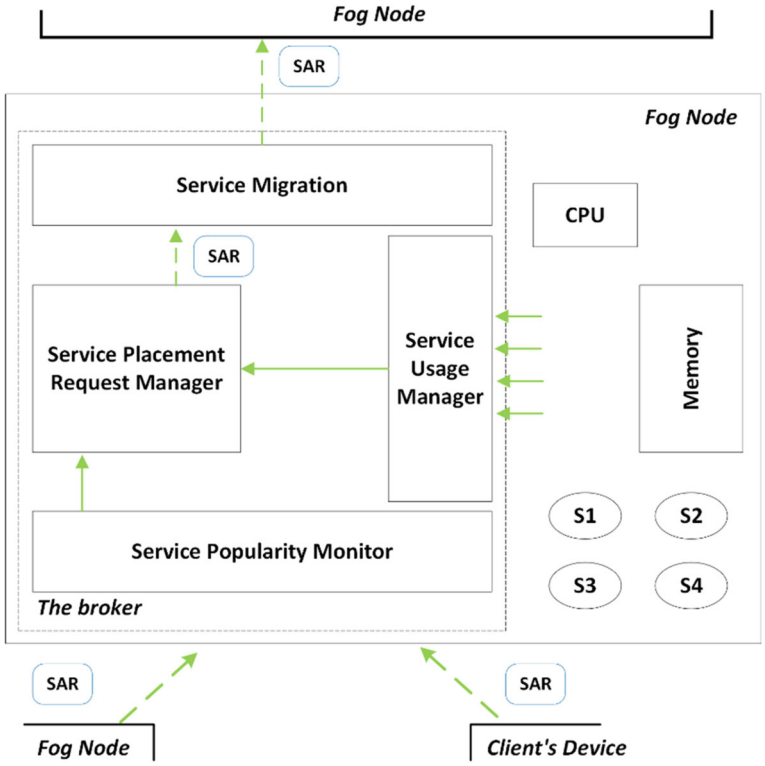

FIGURE 5 Decentralized service placement, the broker

Fog clients must be connected to one leaf device or gateway in the fog network to start using available services. Each time the client requests a specific service, a service allocation request (SAR) is generated to obtain permission to reserve the computing resources. The SPRM decides to accept the allocation of this service or forward the SAR to the next upper fog device (see Figures 3 and 4). The SPRM decides after analyzing the local device information gathered from the SPM and SUM. Although the SUM aims to collect internal data about fog devices, the SPM is designed to measure the service request rate, which the SPRM uses in its calculations. Each node either decides to host the services or migrates them to the next available node in the upper layer.

3.2 | System model

In the proposed model, which follows the principles of FSPP, all applications requested by a group of clients, C_n, are initially allocated to cloud servers, S_{cloud}. Each application comprises a set of (interrelated) service units. Therefore, a sequence of service units must be executed to perform a specific application. Applications in the cloud are called by clients in the lower layer (closer to the edge) with a group of interconnected fog devices D in between. These fog nodes have constrained but available processing power that can complete the sequence of service units without relying on cloud servers. SP_S^{Cloud} is a term used to define the shortest path between service tasks and the cloud with the minimum number of hops. The father (D_1) of the device defines the first device that receives service requests.

The decentralized approach allocates services to local (neighboring) fog devices. Various instances (S_x^y, where x is the service number and y is the code of the device that hosts the services) are allocated across the fog nodes. Thus, considering a given instance, we can formulate the relation as a many-to-one relationship $alloc : [S_x^y] \rightarrow [D_x]$. Conversely, many-to-many is the relationship if we consider [Sx] → [Dx].

To complete an application, every client C_x must be connected to the system through a leaf device. The leaf device can communicate with one or more devices simultaneously. Thus, the relationship in this model was defined as a many-to-one relationship $conn.[C_n] \rightarrow [D_x]$. The service request rate $\gamma_{S_x}^{C_n}$ for each client must be considered in this study to categorize

services. Each device D_i connected to the system must analyze its behavior by monitoring the request rate $\gamma_{S_x}^{D_i}$ that it receives for each task. To calculate the request rate for D_x, for example, we must sum all the requests of clients who are in the range covering D_x as follows:

$$\gamma_{S_x}^{D_i} = \sum^{C_n} \gamma_{S_x}^{C_n} \forall C_n \in \textit{The coverage area of } D_i. \tag{2}$$

To determine the performance of each device in the system, we consider the computational capacity of a fog device. Therefore, the resource capacity is introduced as $R_{D_i}^{Cap} = \left[r_{cpu}\right]$, which is constant for each device and where $R_{D_i}^{Consp}$ is the power consumption of Di depending on the allocation process in each device, which must be variable. Therefore, it is essential to calculate the total resource usage $R_{D_i}^{utz}$ for each computing node. The total resource usage can be calculated as follows:

$$R_{D_i}^{utz} = \sum^{S_x} R_{D_i}^{Cap} \times \gamma_{S_x}^{D_i} \forall S_x \in D_i. \tag{3}$$

3.3 | Optimization model

The proposed algorithm allocates the most popular (in-demand) computing loads closer to the client layer and allows all nodes to accept computer loads if resources are available with the correct specifications. Thus, the service request rate is part of the acceptance metric in the SPRM module. The decision for each fog node is to locally analyze the service request rate prior to migrating less popular services to the cloud. The heavy tasks that overload the local nodes also migrate to the upper level.[56] Notes that heavier tasks should be kept closer to the cloud provider. In both cases, the interoperated services migrated to the cloud once classified. The migration of the interoperated services with the undesirable service decreases the number of unwanted hops in the network, as previously discussed in reference to Figure 4. Table 2 summaries the functions and variables that are used in the proposed model.

Algorithm 1 shows the pseudocode for the proposed enhanced service placement policy. The placement algorithm is invoked when a particular service cannot serve appropriately with the local fog nodes and/or when the available capacity of a fog device is inadequate to satisfy the maximum service requirements. (line 1). The latter condition ensures that all fog nodes in the proposed policy have sufficient capacity to run the most popular services. If these criteria are met, the

TABLE 2 Summary of the functions and variables used in the proposed model

	Notation	Description
Variable	S_x	Service S_x in the system
	S_x^y	An instance y of service x
	D_i	Specific fog device in the system
	S_{cloud}	The cloud server
	$R_{D_i}^{cap}$	Resource capacity of fog device
	$R_{D_i}^{consp}$ Rdi cons	Resource consumption of dog device
Function	AvaS (D_i)	Function that return the list of the variable instances in device D_i
	Install (S_x^y)	Function that download and install specifc instance on specific device
	Father (S_x)	Function that identify father device for service S_x
	Overh (D_i)	Function that return 30% of current resource for device D_i
	ShrtP(D_i)	Function that return the list of fog nodes in the shortest path to cloud
	TopRSev($\gamma_{S_x}^{D_i}$)	Function the service which record the lowest request rate in device D_i
	DAS (S_x)	Function that return true if the service is delay-sensitive application.
	nDSA (S_x)	Function that return the list of non-delay-sensitive application.
	Migrate (S_x^y)	Function that deactivate the instance S_x or several instances in D_i and send SAR request to next device to start migration process

gateway or leaf device D_i is the father of route D_{father} (line 2). The father device is the closest computing node to the client's gadget. Thus, it is critical to maintain the highest demand instances in a leaf device.[57]

Additionally, once the father device is selected, the first-child device D_{Child_1} must be selected (line 4) by identifying the shortest path to the cloud (line 3). Thus, the first-child device D_{Child_1} receives SAR messages if D_i begins straggling. The service placement request manager (SPRM) in the child device decides to either host the service or migrate it again by generating another SAR to its own child while recalculating the shortest path. The decision of the child depends on the popularity of the service in D_{Child_1}. Typically, SPRM gathers essential data by SUM and SPM, where both are located within the same local broker. The reinforcement optimization for a decentralized service placement policy (RODSPP) algorithm also ignores the remaining child in the route to the cloud, where each fog node recomputes the shortest path once the previous criteria are satisfied. This strategy reduces the dependency on operational conditions for the remaining distant fog nodes.

Because this algorithm aims to improve service availability in the fog nodes, the system downloads the services from the cloud in some cases. To download service S_x^y in the candidate fog device D_i (line 6), the candidate node must have adequate resources to serve the requested tasks; the device overload must also be within the acceptable threshold, and the tasks must be recognized as part of delay-sensitive applications (line 5). Eventually, the algorithm uniquely guarantees the maintenance of the overload of all computing nodes, while the POP algorithm focuses only on the guaranteed availability of services at the nodes.[35] The proposed algorithm then installs services with the highest service rate (line 8); otherwise, the candidate service is migrated to the upper level (line 17). The migration process extends the computing capacity by alleviating current loads. If the service is recognized as a high priority, the interoperated services for the lowest service rate must be migrated together (line 11). To create the required computation space to perform the top services, the number of spaces must be identified (line 9). The placement algorithm begins to sequentially migrate the lowest services and its interoperated services to its child (lines 12-14). Thus, the node begins by sending $SAR_{S_x}^{D_{Child}}$ to its child. Every child identifies his or her own child to communicate directly. Once the migrated services are deallocated from the father device, the required services are downloaded (line 15).

Algorithm 1. Enhancement of placement optimization policy algorithm

```
Result: Data Output to Actuator
Input: D_i, S_x^y
Start;
Generate tuples;                                   /*Data transfer from IoT layer to Fog layer*/

while S_x^y ∉ AvaS(D_i) || (R_{D_i}^{Consp} - R_{D_i}^{Cap}) < TopRSev(Y_{S_x}^{D_i}).size do
    D_i ← father(S_x)                                          /*Activation of the Algorithm*/
    SP_{D_i}^{Cloud} ← ShrtP(D_i)                              /*Identify the shortest path*/
    D_{child1} ← SP_{D_i}^{Cloud}                              /*Identify the 1st child*/

    if (R_{S_x}^{Consp} × Y_{S_x}^{D_i} < (R_{D_i}^{Consp} - R_{D_i}^{Cap})) & ((R_{S_x}^{Consp} × Y_{S_x}^{D_i} < OverH(D_i)) || (DSA(S_x)))
    then
        D_i ← Install(S_x^y)                                   /*Install the required instance from cloud*/
    else
        if TopRSev(Y_{S_x}^{D_i}) then
            R_{D_i}^{ToFree} ← (R_{S_x}^{Consp} × Y_{S_x}^{D_i} < (R_{D_i}^{Consp} - R_{D_i}^{Cap}))
            S_n ← (low(Y_S^{D_i}) ∩ nDSA(S_x))      /*Select the non-sensitive service & low request
            rate*/
            interS[s_n^y, s_n^{y+1}, ..] ← interS(s_n^y);          /*Identify the interoperated services*/

            while R_{D_i}^{ToFree} ≤ (R_{S_x}^{Consp} × Y_{S_x}^{D_i}) do
                D_{child1} ← migrate(∧_{k=n}^{k→∞} S_k^{D_i} ∀ k ∈ interS[ ])

            D_i ← Install(S_x^y)
        else
            D_{child1} ← migrate(S_x^y)
```

We now consider that the case study device D_i is currently allocated s_1, s_2, and s_4 in Figure 4. Due to emerging popular services and limited resources, SPRM may decide to migrate s_2 as the lowest service usage. This action had several consequences. First, D_1 is assigned as the father device, and D_2 is assigned as the child, where the shortest route $D_1 \rightarrow D_2 \rightarrow S_{Cloud}$ has two hops, and the other route has one more step $D_1 \rightarrow D_3 \rightarrow D_2 \rightarrow S_{Cloud}$. Second, the most popular service requirements must be less than the currently available computing resources, and the node configurations must satisfy computing requirements; the latter condition avoids overloading the computing nodes. Then, we are obligated to migrate s_2 and its interoperated services s_4, and the first migration process occurred for s_2, followed by s_4. Finally, the required service triggers the download, while the migrated services commence finding another child host in the shortest path.

Algorithm 1 shows the pseudocode of the proposed resource optimization algorithm. The request for migration is activated only when the requested service exceeds the currently available resources. We have provided a service for data classification using a lightweight module.[56] The service allocation request is sent to the gateway and then onto the fog layer. In this layer, the controller estimates the service requirements and sorts the devices accordingly. Thus, time and complexity are limited in sorting and job allocation. Devices that have the required capacity will be identified and allocated for computing in a predictive manner; thus, the tasks will be assigned to those nodes that have the capacity and are available for operation. Also, complex tasks will be assigned to nodes that are not busy and have sufficient power to perform effectively. Simple tasks are thus assigned to nodes that are busy and have less capacity. This approach minimizes latency and power consumption, and a reduction in power consumption generally reduces overall operational costs.

4 | EVALUATION AND EXPERIMENTAL RESULTS

4.1 | Evaluation

We used iFogSim simulator[58] for the microservice-based simulation. The module placement class has been modified to evaluate the proposed model. This evaluation compared the POP[5] with the edge based on iFogSim's built-in placement policy. Different scenarios were considered to evaluate the proposed algorithm by varying the parameters of the number of applications, the number of fog devices, and the number of users/clients. The configurations of the devices in the simulation environment are shown in Table 3, and the experiments followed the same configuration parameters in the POP study.[45] The experiments used a tree-based network topology to manipulate the number of devices in a system. Each fog device interacted with another fog device at the next level via the shortest path to the cloud. This forwarding process represents the migration activity for the lowest-priority task. For simplicity, the network device's behavior was not changed and was fixed with the shortest path once identified. This proposal identifies the number of children at each level, as shown in Figure 6.

As discussed in Section 4.1, cloud servers are assumed to provide unlimited computational resources. The memory in fog devices was sufficiently large in these experiments; thus, we can ignore memory's immediate influence, which is the current trend in fog devices: memory is rarely a limitation.[59] Although computational capacity is the prime evaluation vector, network bandwidth was also considered to affect migration, which relies on the exchange of data between devices. As the number of migrations increased, the number of hops also increased. Thus, we considered the number of hops as the metric to evaluate network usage.

Microservice-based applications are common in IoT domains. In microservice-based applications, the number of service placement requests describes service popularity. System latency is a critical factor for real-time services because delays may not be tolerated in some cases. The case considered in these experiments was an online store application, as shown in Figure 7. A similar benchmarking experiment is discussed in Reference 45.

This microservice-based online store application is widely used in IoT modeling and is called a shock shop.[1] The configurations for this application in the proposed container followed the benchmark.[45]

4.2 | Experimental results

The experimental setup for the proposed system followed existing recommendations. A different set of models was used in the experiment to compare and evaluate performance. The execution of simulation-based applications varies from

TABLE 3 The summary of configurations of experiments

Device	Parameters	Units	Value
Cloud	CPU	MIPS	40 000
	RAM	GB	40
	Bandwidth	Byte/ms	10 000
	Link latency	ms	100
	Power	Watt	107
	Request rate	Req/ms	0.01
Gateway	CPU	MIPS	10 000
	RAM	GB	10
	Bandwith	Byte/ms	1000
	Link latency	ms	50
	Power	Watt	100
Fog device	CPU	MIPS	20 000
	RAM	GB	10
	Bandwidth	Byte/ms	2000
	Link latency	ms	100
	Power	Watt	100
	Request date	Req/ms	0.01

TABLE 4 System specification

Device	Model	OS	RAM	HDD	Processor
1	HP	Win-10	6 GB	500 GB	AMD 2.9 GHz
2	Dell	Win-10	8 GB	1 TB	Intel 3.0 GHz
3	Lenovo	Win-7	4 GB	500 GB	Intel 2.56 GHz

machine to machine, depending on its scalability level. The details of the device model, operating system (OS), random access memory (RAM), hard disk drive (HDD), and processors that participated in the experiment are shown in Table 4.

CloudSim[60] is a requirement for iFogSim for cloud-based operations, and datacenter is managed by this tool. Java and JavaScript object notation (JSON) used to program the proposed algorithm, while common math is the library used for complex mathematical computations in the simulations.

The results of the experimental evaluation are presented in terms of CPU usage, system latency, and network usage. The equations used to calculate these two metrics were programmed in the simulator. By analyzing the iFogSim source code, we obtained the equation used to calculate them, as shown in Equations (4) and (5), respectively:

$$Net_{Usg} = \frac{\sum^{Req_u(D_x,D_y)} \left(T^{ltsy}_{D_x \to D_y} \times Req_{Size} \right)}{Total\ simulation\ time}, \tag{4}$$

where $T^{ltsy}_{D_x \to D_y}$ is the time consumed to migrate the request from one device to another and Req_{Size} is the size of the request that travels through the network. Network usage is the amount of data transferred from the original device or the loaded device to the destination device, which enriches free resources at a specific time.

An increase in the number of devices leads to more complex network usage scenarios, which can create network congestion, which indicates low system performance. A network usage comparison indicates the low or high performance of a system. Compared to the edge policy, the proposed solution reduces network usage by executing most of the tasks

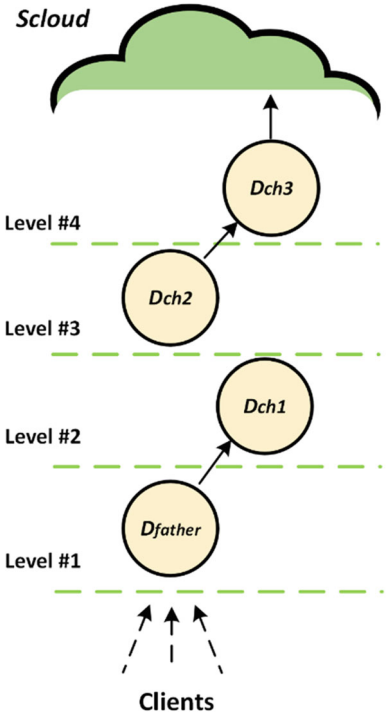

FIGURE 6 Proposed network

closer to the clients and by a well-formed job allocation mechanism at the fog layer. This strategy minimizes cloud usage, which eliminates the use of data transfer/communication networks by default.

System latency is the time required to perform a set of interoperated services to achieve the required application. iFogSim measures system latency by determining the average time required to execute the complete path of the interoperated services. Latency is the most important characteristic of the computing paradigm: the lower the system latency is, the more reliable the system. Equation 6 was used to calculate the system latency $\left(Lty_{sys}\right)$ as configured in the simulation tool:

$$Ltsy_{sys} = \frac{\sum^{C_n} T_n^{Srt} - T_n^{Fnl}}{Req_u} \forall Req_u \in \text{interoperated task}, \tag{5}$$

where T_n^{Srt} and T_n^{Fnl} are the start and final times required by the nth service, respectively, and Req_u is the total number of requests in the interoperated service list.

The following figures show a comparison of the three algorithms. The results of the proposed algorithm are labeled as RODSPP; those for Guerrero and Lera[45] are labeled as POP; and the edge is the label for the base policy of iFogSim. Figures 8 to 10 include two subfigures that show the effects of variation in the setting of execution: (A) variations in the number of users connected to one leaf device or gateway to examine varying levels of workloads; and (B) variations in the number of devices in the fog environment to study the influence of the route length on the network performance.

Figure 8 shows the hop count outcomes and plots the weighted average hope count proposed in the POP manuscript,[45] representing how the most popular services are closer to customers. Figure 9 shows the latency results for a representative loop of the application. At their highest rates, the experiment configured most of the parameters, such as accounts, orders, frontend, and edges. The simulator calculates the time taken by the edge server to complete the requested tasks.

CPU usage is an important performance metric of a system because the processing quality of a system in a scalable manner is measured in terms of the CPU. If CPU usage is too high, delays occur during processing.

The results of CPU usage, UCPU, are determined by the Equation (6) as follow:

$$U_{CPU} = \frac{(T_b - T_i) \times R_t}{T_{mips} - F_{mips}} \times 100, \tag{6}$$

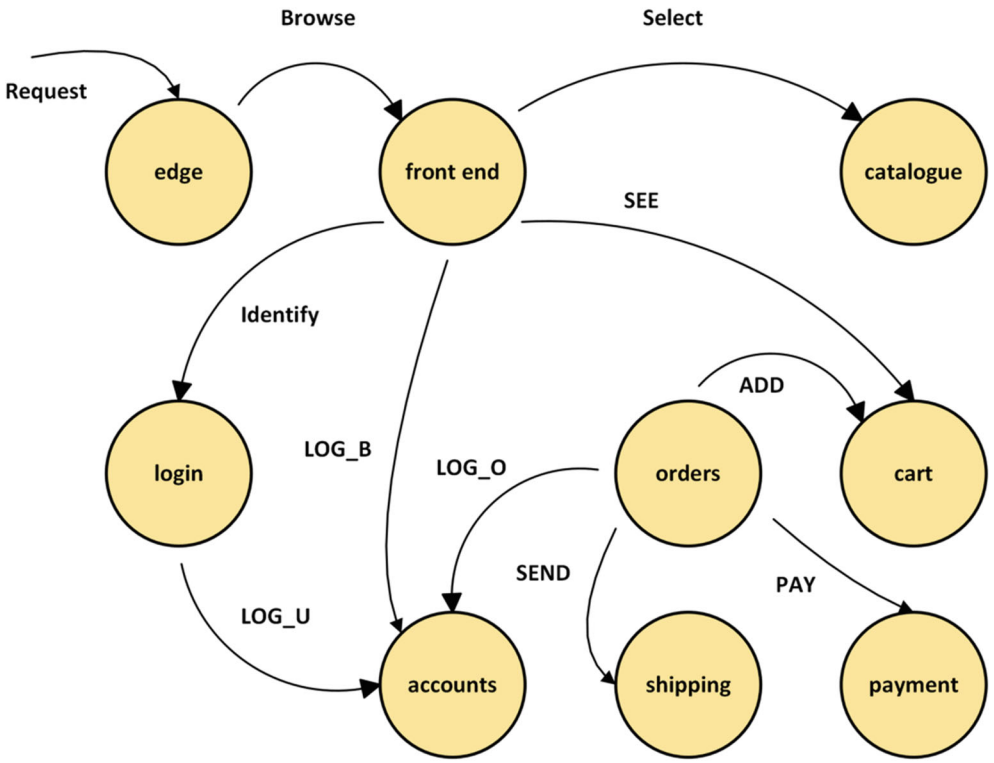

FIGURE 7 Application edges for online store-based case study

where T_b is the busy task rate, T_i is the idle task rate, R_t is the average tuple time, T_{mips} is the total number of mips (million instructions per second) in the host, and F_{mips} is the final mips in the control of the CPU.

Figure 10 shows the CPU usage of the devices by varying the number of users in each device. The experimental setup was configured as follows: two applications, up to five users per gateway, one child per device, and two fog devices per level. This setup is similar to the experimental benchmarking setup.

5 | DISCUSSION

System latency is an important performance metric in fog computing. The weighted average hops effectively measures the proximity between the services and users, therefore providing some sense of system complexity. Thus, we used both metrics to answer the first research question. The series labeled edge, POP, and RODSPP in Figure 8 were thus analyzed. In Figure 8A, the graph shows a marked increase in the number of hops for all three approaches when the number of users increases, implying that the system would slow down once the number of users increases, and the resources conflict. The weighted average hops indicated that RODSPP consumed 4% more hops than POP and 23.1% fewer hops than edges. The increase in hops in the RODSPP was due to migrating the interoperated services, which have low request rates and matching the size of the task with the node resources to avoid overload. RODSPP prevented overload for the computing nodes in all layers by migrating excessive tasks from the overloaded ones.

POP and edge were not affected by the changes in the number of fog devices; RODSPP tended to migrate more services to the upper levels closer to the cloud. Due to the addition of two more service categories to activate the RODSPP algorithm, such as delay-sensitive applications, even with a low request rate, the migration process increased with an increasing number of fog levels in the system, as shown in Figure 8B. Therefore, RODSPP did not decrease the number of hops while maintaining the most usable and delay-sensitive services in the edge-fog network. This behavior is acceptable if we try to solve many issues with limited resources in leaf nodes. In Figure 9, after we split the tasks into high and low request rates, Figure 9A shows that there has been a marked rise in the time latency incurred by the highest popularity applications. What is striking in the chart is the outperformance of RODSPP by 24.1%, which is due to matching the workloads with

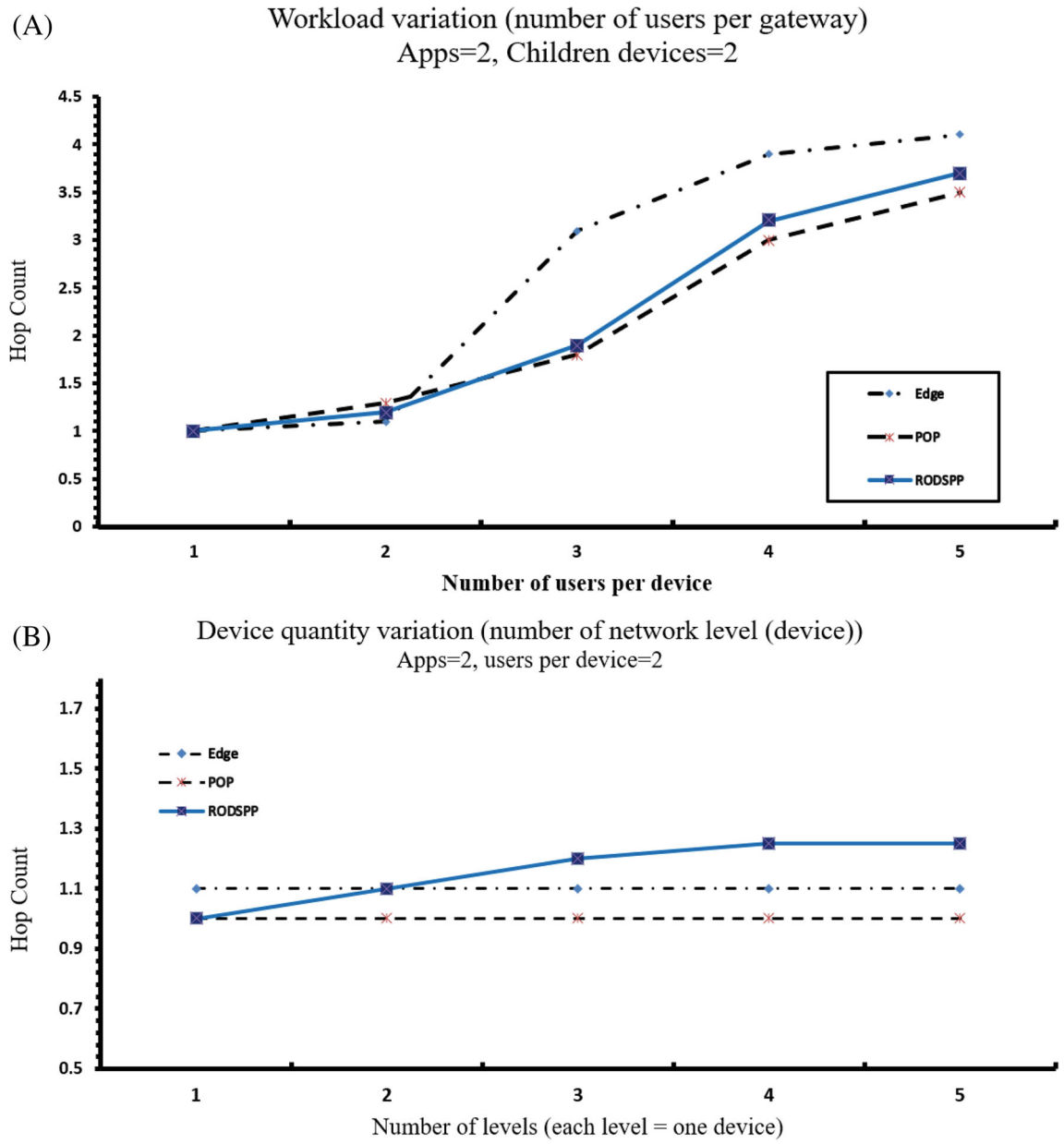

FIGURE 8 Hops count with different situations. Experiment with two applications, two users, and two levels of fog devices: (A) changing the number of devices associated with the edge device; and (B) changing the number of proposed fog levels

the capability of the computing nodes. Although RODSPP recorded a noticeable increase in hops, it achieved fair results in the overall scheme of the performance.

POP ignores low-requested services without considering their sensitivities. We added a delay-sensitive application with a low request rate to the proposed experiment. RODSPP allocates low-popularity applications on a leaf device if it is a delay-sensitive application. To answer the second research question, we consider Figure 9B, which shows the performance of the POP and RODSPP for applications with low popularity but high time sensitivity. The chart shows the comparable behaviors of both the RODSPP and POP policies in non-time-sensitivity applications. The chart shows that there has been a sharp decline in system latency for RODSPP in delay-sensitive applications. Thus, RODSPP is recognized as a valid policy for delay-sensitive applications even with low popularity. Generally, even though the system places all services at the border in the case of one user, the gateway still engages the cloud to accomplish complex tasks in real life. The third question in this study addresses the performance of RODSPP, among other models, in CPU, network usage, and system latency. The system latency parameter has already been discussed in the second question. The RODSPP showed

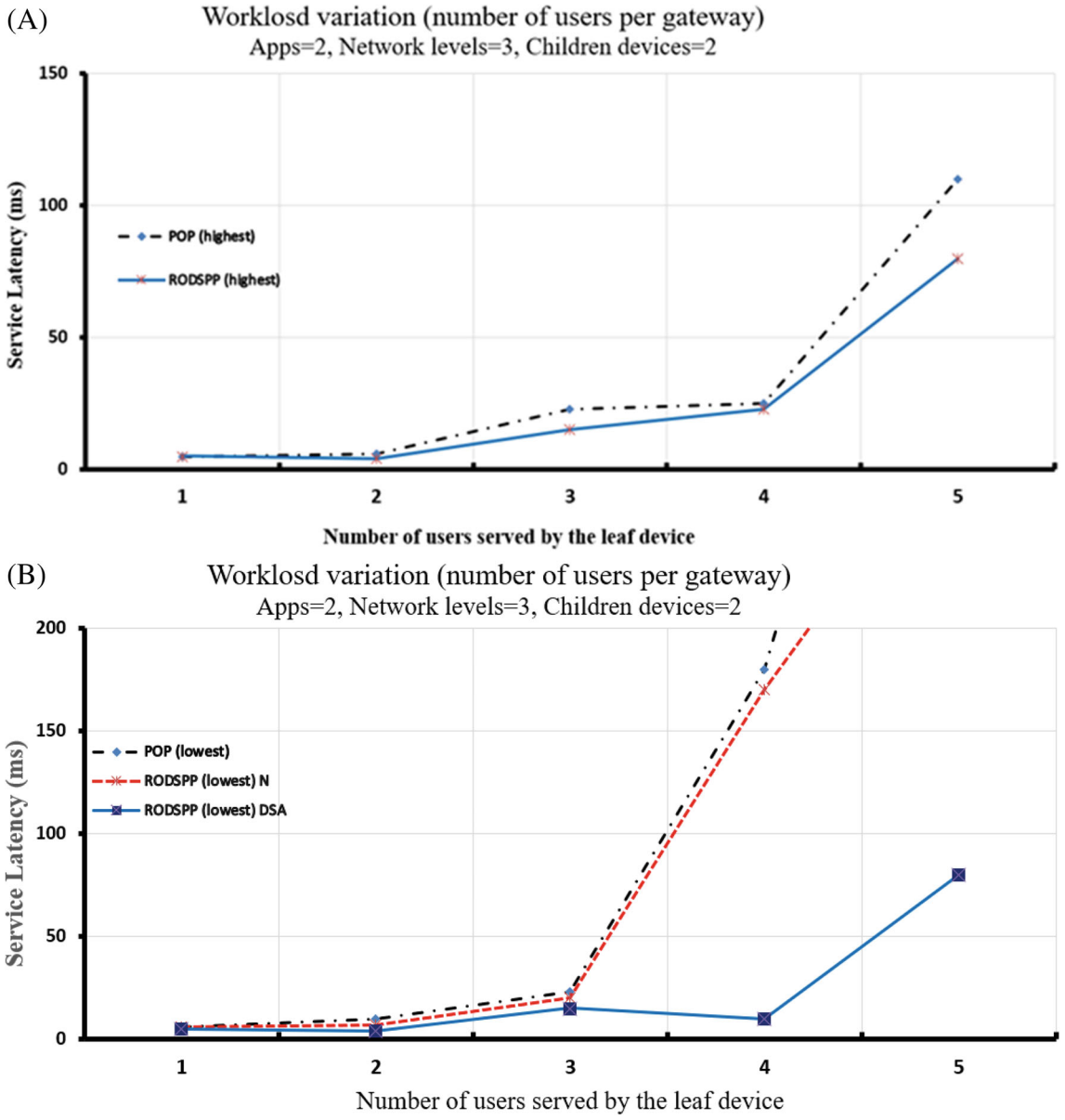

FIGURE 9 Service latency in different situations. Experiment with two applications, two users, and two levels of fog devices: (A) changing the number of devices associated with the edge device; and (B) changing the number of proposed fog levels

outstanding outcomes for delay-sensitive applications with low request rates, as shown in Figure 9. Thus, RODSPP can be highly recommended for time-sensitive applications.

Figure 10 shows the relation between the CPU usage in different layers and the network usage. Figure 10A shows the CPU usage of the father nodes in the proposed algorithm, which outperformed POP by matching the workloads with the available resources. This feature has improved the use of border CPUs. The proposed algorithm tends to decrease the load of children's CPUs once a new layer appears, which increases network usage. Although RODSPP consumes 4.5% more network bandwidth than POP, it decreases all fog resource usage by 20%. The primary contribution of this study was to synthesize well-disciplined resource allocation approaches using them in local optimization through a divide-and-conquer approach in a carefully articulated realistic IoT fog networking environment. This study demonstrates successful IoT fog resource allocations in terms of controlled response time and constrained computing resource usage, thus providing important insights and guidelines for the community to refer to and seek further enhancements in real-time IoT fog applications.

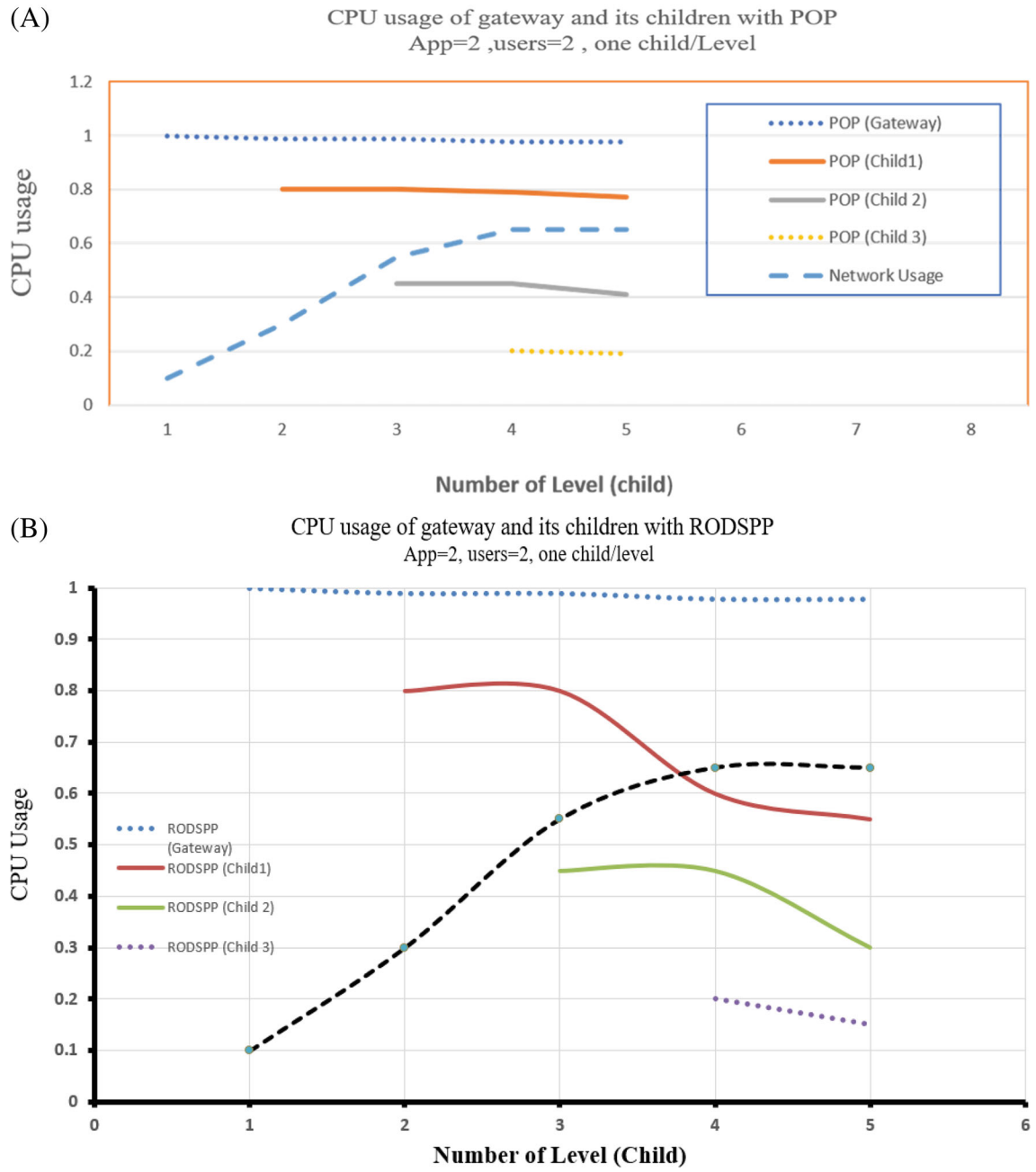

FIGURE 10 CPU usage of the devices with regard to their topology distribution. Experiment with two applications, two users, and two levels of fog devices: (A) CPU performance with the POP algorithm; and (B) CPU performance with the RODSPP algorithm

6 | CONCLUSION AND FUTURE WORK

6.1 | Conclusion

The span of IoT devices is growing day by day, generating large amounts of data. However, the cloud cannot manage and process an increasing number of IoT devices for real-time processing, which requires low latency and reduced resource consumption due to its centralized and distant architecture. Server mobility and decentralization are requirements for IoT devices for real-time data processing. A fog computing paradigm is thus proposed in this study to meet the requirements of future IoT networks. In this regard, this research proposed reinforcement optimization for a decentralized service placement policy (RODSPP), which attempts to mitigate the drawbacks of existing placement policies by decentralizing

FSPP intelligently. To verify the performance of the proposed RODSPP, extensive experimental comparisons were made against the other well-known algorithms including edge and POP using the evaluation parameters of service latency, network usage and computing usage. The experimental analysis showed that the proposed load-balancing algorithm better utilized the local fog computing resources and supported well the real-time performance requirements of the IoT applications.

6.2 | Future work

Although the proposed algorithm (RODSPP) achieved the improved performance within the parameters considered in this study, we plan to improve upon these results further by considering more performance dimensions relevant to the IoT applications in the industry. Thus, we aim to support the real-time industry IoT application with the proposed innovative FSPP scheme.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ACKNOWLEDGMENT

Open access publishing facilitated by University of Technology Sydney, as part of the Wiley - University of Technology Sydney agreement via the Council of Australian University Librarians.

ORCID

Tony Jan https://orcid.org/0000-0002-3114-8978
Mukesh Prasad https://orcid.org/0000-0002-7745-9667

REFERENCES

1. El-Sayed H, Sankar S, Prasad M, et al. Edge of things: the big picture on the integration of edge, IoT and the cloud in a distributed computing environment. *IEEE Access*. 2017;6:1706-1717.
2. Catruc I, Iosifescu D. IoT integration with Mobile and cloud solutions. *Inform Econom*. 2020;24(2):63-74.
3. Srivastava G, Lin JC-W, Pirouz M, Li Y, Yun U. A pre-large weighted-fusion system of sensed high-utility patterns. *IEEE Sens J*. 2020;21(14):15626-15634.
4. Manyika J, Chui M. By 2025, internet of things applications could have $11 trillion impact. Insight publications. 2015.
5. Khosroabadi F, Fotouhi-Ghazvini F, Fotouhi H. SCATTER: service placement in real-time fog-assisted IoT networks. *J Sens Actuator Netw*. 2021;10(2):26.
6. Lenka RK, Rath AK, Tan Z, et al. Building scalable cyber-physical-social networking infrastructure using IoT and low power sensors. *IEEE Access*. 2018;6:30162-30173.
7. Yazdinejad A, Srivastava G, Parizi RM, Dehghantanha A, Karimipour H, Karizno SR. SLPoW: secure and low latency proof of work protocol for blockchain in green IoT networks. Paper presented at: 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring); 2020: 1-5; IEEE.
8. Puthal D, Obaidat MS, Nanda P, Prasad M, Mohanty SP, Zomaya AY. Secure and sustainable load balancing of edge data centers in fog computing. *IEEE Commun Mag*. 2018;56(5):60-65.
9. Kim D-Y, Jung M. Data transmission and network architecture in long range low power sensor networks for IoT. *Wirel Pers Commun*. 2017;93(1):119-129.
10. Jaiswal A, Kumar S, Kaiwartya O, Prasad M, Kumar N, Song H. Green computing in IoT: time slotted simultaneous wireless information and power transfer. *Comput Commun*. 2021;168:155-169.
11. Kaiwartya O, Abdullah AH, Cao Y, et al. Virtualization in wireless sensor networks: fault tolerant embedding for internet of things. *IEEE Internet Things J*. 2017;5(2):571-580.
12. Tiwary M, Sharma S, Mishra P, Sayad HE, Prasad M, Puthal D. Building scalable Mobile edge computing by enhancing quality of services. Paper presented at 13th International Conference on Innovations in Information Technology (IIT'18); 2018.
13. Kumar P, Kumar R, Srivastava G, et al. PPSF: a privacy-preserving and secure framework using blockchain-based machine-learning for IoT-driven smart cities. *IEEE Trans Netw Sci Eng*. 2021;8(3):2326-2341.
14. Vögler M, Schleicher JM, Inzinger C, Dustdar S. A scalable framework for provisioning large-scale IoT deployments. *ACM Trans Internet Technol*. 2016;16(2):1-20.
15. Sulimani H, Alghamdi WY, Jan T, Bharathy G, Prasad M. Sustainability of load balancing techniques in fog computing environment. *Procedia Comput Sci*. 2021;191:93-101.

16. Sharma S, Puthal D, Tazeen S, Prasad M, Zomaya AY. MSGR: a mode-switched grid-based sustainable routing protocol for wireless sensor networks. *IEEE Access*. 2017;5:19864-19875.
17. Alam T, Benaida M. The role of cloud-MANET framework in the internet of things (IoT). arXiv Preprint arXiv:190209436; 2019.
18. Kaur H, Sood SK. Fog-assisted IoT-enabled scalable network infrastructure for wildfire surveillance. *Journal of Network and Computer Applications*. 2019;144:171-183.
19. Kamel MA, Yu X, Zhang Y. Formation control and coordination of multiple unmanned ground vehicles in normal and faulty situations: a review. *Annu Rev Control*. 2020;49:128-144.
20. Puthal D, Wilson S, Nanda A, et al. Decision tree based user-centric security solution for critical IoT infrastructure. *Comput Electr Eng*. 2022;99:107754.
21. Puliafito C, Vallati C, Mingozzi E, Merlino G, Longo F. Design and evaluation of a fog platform supporting device mobility through container migration. *Pervasive Mob Comput*. 2021;74:101415.
22. Baranwal G, Vidyarthi DP. FONS: a fog orchestrator node selection model to improve application placement in fog computing. *J Supercomput*. 2021;77(9):10562-10589.
23. Salaht FA, Desprez F, Lebre A. An overview of service placement problem in fog and edge computing. *ACM Comput Surv*. 2020;53(3):1-35.
24. Mubarakali A, Durai AD, Alshehri M, AlFarraj O, Ramakrishnan J, Mavaluru D. Fog-based delay-sensitive data transmission algorithm for data forwarding and storage in cloud environment for multimedia applications. *Big Data*. 2020. doi:10.1089/big.2020.0090
25. Brogi A, Forti S, Guerrero C, Lera I. How to place your apps in the fog: state of the art and open challenges. *Softw Pract Exp*. 2020;50(5):719-740.
26. Huang C, Wang H, Zeng L, Li T. Resource scheduling and energy consumption optimization based on Lyapunov optimization in fog computing. *Sensors*. 2022;22(9):3527. doi:10.3390/s22093527
27. Fan Q, Bai J, Zhang H, Yi Y, Liu L. Delay-aware resource allocation in fog-assisted IoT networks through reinforcement learning. *IEEE Internet Things J*. 2021;9:5189-5199.
28. Sami H, Mourad A, Otrok H, Bentahar J. Demand-driven deep reinforcement learning for scalable fog and service placement. Paper presented at: IEEE Transactions on Services Computing; 2021.
29. Wei H, Weng H, Zhai M. Research on the application of 5G edge computing Technology in the Power Internet of things. Paper presented at: 2021 IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC); 2021:600-605; IEEE.
30. Wang Z, Zhong Z, Ni M. Application-Aware offloading policy using SMDP in vehicular fog computing systems. IEEE International Conference on Communications Workshops (ICC Workshops); 2018:1-6.
31. Ghosh S, Ghosh SK. Mobility driven cloud-fog-edge framework for location-aware services: a comprehensive review. In: Mukherjee A, De D, Ghosh SK, Buyya R, eds. *Mobile Edge Computing*. Berlin, Heidelberg: Springer; 2021:229-249.
32. Salimian M, Ghobaei-Arani M, Shahidinejad A. Toward an autonomic approach for internet of things service placement using gray wolf optimization in the fog computing environment. *Softw Pract Exp*. 2021;51(8):1745-1772.
33. Arora U, Singh N. IoT application modules placement in heterogeneous fog–cloud infrastructure. *Int J Inf Technol*. 2021;13(5):1975-1982.
34. Jamil B, Ijaz H, Shojafar M, Munir K, Buyya R. Resource allocation and task scheduling in fog computing and internet of everything environments: a taxonomy, review, and future directions. *ACM Comput Surv*. 2022. doi:10.1145/3513002
35. Ghobaei-Arani M, Shahidinejad A. A cost-efficient IoT service placement approach using whale optimization algorithm in fog computing environment. *Expert Syst Appl*. 2022;200:117012.
36. Etemadi M, Ghobaei-Arani M, Shahidinejad A. A learning-based resource provisioning approach in the fog computing environment. *J Exp Theor Artif Intell*. 2021;33(6):1033-1056.
37. Abd Elaziz M, Abualigah L, Ibrahim RA, Attiya I. IoT workflow scheduling using intelligent arithmetic optimization algorithm in fog computing. *Comput Intell Neurosci*. 2021;2021:1-14.
38. Farzin P, Azizi S, Shojafar M, Rana O, Singhal M. FLEX: a platform for scalable service placement in multi-fog and multi-cloud environments. Paper presented at: Australasian Computer Science Week 2022 (ACSW 2022); 2022:106-114.
39. Velasquez K, Abreu DP, Curado M, Monteiro E. Service placement for latency reduction in the fog using application profiles. *IEEE Access*. 2021;9:80821-80834.
40. Morkevicius N, Venčkauskas A, Šatkauskas N, Toldinas J. Method for dynamic service orchestration in fog computing. *Electronics*. 2021;10(15):1796.
41. Malukani SP, Bhensdadia C. Fog computing algorithms: a survey and research opportunities. *Appl Comput Syst*. 2021;26(2):139-149.
42. Maiti P, Sahoo B, Turuk AK, Kumar A, Choi BJ. Internet of things applications placement to minimize latency in multi-tier fog computing framework. *ICT Express*. 2021;8:166-173.
43. Jazayeri F, Shahidinejad A, Ghobaei-Arani M. Autonomous computation offloading and auto-scaling the in the mobile fog computing: a deep reinforcement learning-based approach. *J Ambient Intell Humaniz Comput*. 2021;12(8):8265-8284.
44. Sriraghavendra M, Chawla P, Wu H, Gill SS, DoSP BR. A deadline-aware dynamic service placement algorithm for workflow-oriented IoT applications in fog-cloud computing environments. In: Tiwari R, Mittal M, Goyal LM, eds. *Energy Conservation Solutions for Fog-Edge Computing Paradigms*. Berlin, Heidelberg: Springer; 2022:21-47.
45. Forti S, Lera I, Guerrero C, Brogi A. Osmotic management of distributed complex systems: a declarative decentralised approach. *J Softw Evol Process*. 2021;e2405.
46. Kumar B, Dhurandher SK, Obaidat MS. Performance analysis of the Spectrum access strategies in cognitive radio networks. Paper presented at: 2021 International Conference on Computer, Information and Telecommunication Systems (CITS); 2021:1-5; IEEE.

47. Hu F, Huang H, Guo Z. Review on service composition. Paper presented at: 2021 IEEE 6th International Conference on Smart Cloud (SmartCloud); 2021:117-124; IEEE.
48. Sedghani H, Filippini F, Ardagna D. A randomized greedy method for ai applications component placement and resource selection in computing continua. Paper presented at: 2021 IEEE International Conference on Joint Cloud Computing (JCC); 2021:65-70; IEEE.
49. Apat HK, Bhaisare K, Sahoo B, Maiti P. A nature-inspired-based multi-objective service placement in fog computing environment. In: Udgata SK, Sethi S, Srirama SN, eds. *Intelligent Systems*. Berlin, Heidelberg: Springer; 2021:293-304.
50. Muniswamaiah M, Agerwala T, Tappert CC. A survey on cloudlets, Mobile edge, and fog computing. Paper presented at: 2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom); 2021:139-142; IEEE.
51. Sabireen H, Neelanarayanan V. A review on fog computing: architecture, fog with IoT, algorithms and research challenges. *ICT Express*. 2021;7(2):162-176.
52. Sikandar A, Agrawal R, Tyagi MK, Rao ALN, Prasad M, Binsawad M. Toward green computing in wireless sensor networks: prediction-oriented distributed clustering for non-uniform node distribution. *EURASIP J Wirel Commun Netw*. 2020;2020:183.
53. Burhan HM, Abbas MN, Bara'a AA. A genetic algorithm for task allocation problem in the internet of things. *Iraqi J Sci*. 2021;62:1376-1385.
54. Pareek K, Tiwari PK, Bhatnagar V. Fog computing in healthcare. *IOP Conf Ser Mater Sci Eng*. 2021;1099:012025.
55. Apat HK, Sahoo B, Mohanty S. A quality of service (QoS) aware fog computing model for intelligent (IoT) applications. Paper presented at: 2021 19th OITS International Conference on Information Technology (OCIT); 2021:267-272; IEEE.
56. Al-Tarawneh MA. Bi-objective optimization of application placement in fog computing environments. *J Ambient Intell Humaniz Comput*. 2022;13(1):445-468.
57. Roig PJ, Alcaraz S, Gilly K, Juiz C. Modelling a plain N-hypercube topology for migration in fog computing. In: Thampi SM, Gelenbe E, Atiquzzaman M, Chaudhary V, Li KC, eds. *Advances in Computing and Network Communications*. Berlin, Heidelberg: Springer; 2021:595-608.
58. Gupta H, Vahid Dastjerdi A, Ghosh SK, 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(9):1275-1296.
59. Barros CL, Rocio V, Sousa A, Paredes H. Scheduling in cloud and fog architecture: identification of limitations and suggestion of improvement perspectives. *J Inf Syst Eng Manag*. 2020;5(3):1-12.
60. Buyya R, Pandey S, Vecchiola C. Cloudbus toolkit for market-oriented cloud computing. In: Jaatun MG, Zhao G, Rong C, eds. *Cloud Computing*. Berlin, Heidelberg: Springer; 2009:24-44.

How to cite this article: Sulimani H, Sajjad AM, Alghamdi WY, et al. Reinforcement optimization for decentralized service placement policy in IoT-centric fog environment. *Trans Emerging Tel Tech*. 2022;e4650. doi: 10.1002/ett.4650