Received: 15 August 2019 | Revised: 10 October 2019 | Accepted: 18 October 2019

DOI: 10.1002/cpe.5581

WILEY

RESEARCH ARTICLE

A job scheduling algorithm for delay and performance optimization in fog computing

Bushra Jamil[1] | Mohammad Shojafar[2] | Israr Ahmed[1] | Atta Ullah[3] | Kashif Munir[4] | Humaira Ijaz[1]

[1]Department of CS & IT, University of Sargodha, Sargodha, Pakistan
[2]5G Innovation Center, Institute for Communication Systems, University of Surrey, UK, Guildford, UK
[3]Technical consultant, Eurosoft (UK) Ltd, Bournemouth, England
[4]Department of Computer Science, NUCES, Islamabad, Pakistan

Correspondence
Mohammad Shojafar, 5G Innovation Center, Institute for Communication Systems, University of Surrey, Guildford GU2 7XH, UK.
Email: m.shojafar@surrey.ac.uk; m.shojafar@ieee.org

Summary

Due to an ever-increasing number of Internet of Everything (IoE) devices, massive amounts of data are produced daily. Cloud computing offers storage, processing, and analysis services for handling of such large quantities of data. The increased latency and bandwidth consumption is not acceptable to real-time applications like online gaming, smart health, video surveillance, etc. Fog computing has emerged to overcome the increase in latency and bandwidth consumption in Cloud computing. Fog Computing provides storage, processing, networking, and analytical services at the edge of a network. As Fog Computing is still in its infancy, its significant challenges include resource-allocation and job-scheduling. The Fog devices at the edge of the network are resource-constrained. Therefore, it is important to decide the assignment and scheduling of a job on a Fog node. An efficient job scheduling algorithm can reduce energy consumption and response time of an application request. In this paper, we propose a novel Fog computing scheduler that supports service-provisioning for Internet of Everything, which optimizes delay and network usage. We present a case study to optimally schedule the requests of Internet of Everything devices on Fog devices and efficiently address their demands on available resources on every Fog device. We consider delay and energy consumption as performance metrics and evaluate the proposed scheduling algorithm using iFogSim in comparison with existing approaches. The results show that the delay and network usage of the proposed scheduler improve by 32% and 16%, respectively, in comparison with FCFS approach.

KEYWORDS

delay, energy consumption, fog computing, Internet of Everything (IoE), job scheduling, sensors

1 | INTRODUCTION

The Internet of Things (IoT) represents a comprehensive environment that consists of various everyday objects such as home appliances, computing machines, animals, farms, factories, vehicles, etc, connected through heterogeneous networks. IoT has made these things "smart" to make them sense, process, and effectively communicate through a network to perform useful jobs without end-users interaction.[1] The main focus of IoT was to embed smartness in physical objects only, but later on, this concept was enhanced by Cisco as Internet of Everything (IoE), which emphasizes bringing people, process, data, and things together to make intelligent connections. IoE model helps to automate human activities by enabling Machine-to-Machine (M2M), People-to-Machine (P2M), and People-to-People (P2P) communications.

IoE systems like smart healthcare monitoring, smart traffic, smart farming, etc, follow the architecture of Cloud-centric Internet of Things (CIoT) for storage, analytics, and processing.[2] CIoT data centers usually reside at multi-hop distance from a source node, thus resulting in long delays. Furthermore, ubiquitous computing and the development of 4G/5G technologies have provided computing services everywhere by connecting billions of new IoT applications and devices. By 2020, there will be about 5.4 billion connected devices in the world[3] and about 1 trillion by 2025.[4] It will result in the generation of a huge amount of data that will not be efficiently handled by Cloud, consequently resulting in long latencies and network congestion. Even now, some of the real-time, latency-sensitive, and geo-spatially distributed IoT applications like smart healthcare monitoring, smart traffic surveillance, virtual reality, etc, cannot be efficiently served using Cloud computing as they need low latency.[5] To overcome these limitations of CIoT, several approaches like Edge Computing, Mobile Computing, Fog Computing, etc, have been proposed to provide computational, storage, decision making, and networking services at close proximity to the source node or end-user application.[6]

Among various proposed approaches, Fog Computing is the one that has recently gained the most attention. Fog Computing is a distributed computing paradigm that complements Cloud by providing computing, storage, networking, and decision making near the edge of the network. In Fog computing, mostly Sense Process Actuate Model (SPAM) is used. In SPAM, the sensors sense and collect data which is transmitted to Fog devices for processing. After processing, the results are sent to actuators for taking actions. Fog nodes forward some data to Cloud for managing storage and processing for long-term analytics. Smart phones, internal modems, smart gateways, routers, switches, cellular base stations, etc, are some devices that can serve as Fog nodes.[7] These Fog devices have different architectures and are resource-constrained in terms of processing power, storage capacity, bandwidth, and limited power. Real-time Fog Computing applications need faster response than that of delay-tolerant applications. Therefore, such applications have to compete for these limited resources.

Resource-constrained and latency-sensitive nature of Fog applications make resource management one of the key challenges in Fog Computing.[8] Therefore, the decision of resource allocation and job scheduling is of great importance. Efficient job scheduling can yield fast and timely response that is desirable in smart systems. For example, in a smart healthcare system, a patient's condition needs fast notification to save the patient's life.[9] Therefore, there is need to develop some efficient job scheduling algorithm to maximize utilization of these heterogeneous and resource-constrained Fog devices.[10] The overall objective is to minimize the response time and network usage without increasing energy consumption.[11]

Although some job scheduling algorithms like first come first served (FCFS), Concurrent, FCFS-based Round Robin, delay-priority, etc, have already been proposed, job scheduling in Fog Computing is still in its infancy stage.[12] Existing algorithms like FCFS and Round Robin execute the jobs according to their arrival sequence. Hence, they are unable to reduce the response time of latency-sensitive applications. There is a need to develop a scheduling algorithm that can minimize average response time and network usage for latency-sensitive applications along with optimal energy consumption.

1.1 | Contribution and goal of this paper

In this paper, we propose a novel multi-objective Fog scheduler for latency-sensitive applications that supports IoE service-provisioning. The objective is to maximize utilization of Fog devices that are available at the edge of a network and minimize the latency and energy consumption. The major contributions of the proposed work are as follows.

- **Minimization of application loop delay**: In latency-sensitive applications, the most important criterion is to reduce service response time. The proposed algorithm executes the jobs on the Fog nodes according to their lengths. The execution of the smallest job first results in decreasing loop delays.
- **Minimization of network usage**: Increasing the number of IoE devices connected per application results in increased network-usage. Consequently, there is network congestion. Another objective of our work is to improve the application's performance by reducing network usage.
- **Optimize energy consumption of devices**: Energy is consumed by all the devices of network and the Fog devices are resource constrained, so the proposed algorithm is aimed to optimize energy consumed by Fog devices.

1.2 | Organization

The remainder of this paper is organized in the following order. Section 2 describes the related work. Section 3 explains the architectural model. In Section 4, we present the design and implementation of the proposed algorithm. Experimental results are discussed in Section 5. We conclude and describe the future work in Section 6.

2 | RELATED WORK

In this section, we review some existing resource management techniques that have been proposed for performance optimization in Fog Computing. We use the terms “task” and “job” interchangeably. Majority of these resource management optimizations deal with job allocation, whereas a few approaches focus on scheduling of jobs on these resource-constrained Fog devices. The objectives of both job allocation and scheduling are to minimize make-span, reduce network usage, provide better quality of service, minimize cost, decrease loop delay, minimize device energy consumption, increase network availability, better resource utilization, etc. The algorithms proposed so far only optimize one or two parameters of the above mentioned criteria. Following is a brief overview of some of these proposed job allocation approaches.

Pham and Huh[13] proposed a job allocation strategy of dependent jobs to Fog nodes. Dependent jobs depend on each other and need data communication with each other. Dependent jobs are represented by Directed Acyclic Graph (DAG) or workflows. The authors compute the priority of jobs by traversing DAG and assign these prioritized jobs to Fog nodes. If the Fog nodes are unable to handle a job due to resource-limitation, it is sent to Cloud. The authors do not consider the important parameters like deadline constraints of workflow execution and Fog provider's budget. Zeng et al[14] presented another job image placement and scheduling algorithm in Fog computing. Job image is placed on a storage server. Computations are performed by embedded clients and Fog nodes and they both can share the storage servers. Jobs are scheduled to achieve minimum completion time along with better user experience. Ni et al[15] presented a dynamic resource allocation strategy on the basis of job's completion time and the credibility of Fog nodes by using Priced Timed Petri Nets (PTPNs) to improve resource utilization and user's QoS requirements. Another job allocation algorithm was proposed by Pooranian et al[16] in which they presented a heuristic-based algorithm for resource allocation to optimize energy consumption. They consider resource allocation as "bin packing penalty"-aware problem. Fog Servers are considered as bins, while VMs are packs that are to be served on the basis of time and frequency limitations. To optimize energy consumption, "penalty and reward policy" is used. In case of penalty, the server is not allocated for some iterations. Ni et al[15] proposed an adaptive double fitness Genetic Task Scheduling algorithm for smart cities to optimize task make-span and communication cost. The algorithm considers computing capability, communication cost and delay requirements of Fog devices to achieve better performance. Another heuristic based job scheduling algorithm was designed by Hoang and Dang[17] to achieve low latency and better performance. They propose a Fog-based region architecture to assign jobs to Fog regions and Cloud. Sun et al[18] proposed a two-level resource scheduling scheme. Resources are allocated to various Fog clusters and also to Fog nodes in the same cluster. They use the theory of improved Non-Dominated Sorted Genetic Algorithm II for resource scheduling among Fog nodes of same cluster for multi-objective optimization. The authors claim to achieve short delays and more solidity of job execution.

Liu et al[10] used a priori algorithm for mining association rules. The generated rules are then combined with minimum completion time of a job to schedule it to a Fog device. They claim to reduce execution time and average waiting time. Fog computing aims to serve applications of diverse nature. These applications can either be latency-critical or delay-tolerant. For latency-critical applications, the most important criterion is to reduce loop delay. In another solution reported in the work of Gupta et al[4] FCFS scheduling algorithm was used for job scheduling on a latency-critical application (EEG tractor game). They compute loop delay, energy consumption, and network usage using FCFS algorithm on Fog nodes. They prove that placement of modules on Edge is much better than their placement on Cloud.

Bittencourt et al[12] discussed resource allocation in Fog computing by analyzing different types of applications using three scheduling policies, ie, FCFS, concurrent, and delay-priority. The algorithms are applied on two different types of applications, viz, video surveillance (VSOT) and EEG tractor beam game (EEGTBG). First application is delay-tolerant and the later one is near real-time. Loop delay and network usage were computed to prove that resources should be allocated on the basis of mobility requirements of the application. Choudhari et al[19] proposed a prioritized job scheduling algorithm in Fog computing to reduce overall response time and decrease cost. Upon arrival of a job, its priority is computed on the basis of its deadline. Using this computed priority of a job, it is placed in Fog layer. In each Fog layer, there are multiple micro data centers and Fog nodes that are able to communicate with each other. In case all the data centers in a Fog layer are saturated, the job is moved to Cloud. Other important objectives like energy consumption and network usage are ignored. Moreover, Bitam et al[20] used Bees Life Algorithm (BLA) that is bio-inspired optimization approach for job scheduling. The algorithm was used to optimally distribute jobs on Fog nodes. The proposed algorithm was found to be better in terms of run-time and memory allocation values as compared to Particle Swarm Optimization (PSO) algorithm. Moreover, Gai and Qiu[21] used Reinforcement Learning (RL) to optimize resource allocation and Quality of Experience (QoE). They proposed two RL algorithms for creation of cost mapping tables and optimized resource allocation. Against every user request, a clusters of Fog nodes were setup to achieve particular target like delay.

To reduce energy consumption in edge computing, Zhang et al[22] proposed a double deep Q-learning model (DDQ). The model computed Q-value for each dynamic voltage and frequency scaling (DVFS) algorithm using DDQ. Rectified Linear Units were used as activation function instead of Sigmoid function to prevent gradient vanishing. Nguyen et al[23] proposed a genetic algorithm for optimization of job scheduling in Cloud-Fog environment. Their aim was to reduce execution time of jobs. Each chromosome represents a job assignment to a node. Mutation and crossover were used to generate new population. The authors claimed to get better results as compared to Bee Life Algorithm (BLA) and Modified Particle Swarm Optimization (MPSO). Recently, in 2019, Rahbari and Nickray implemented greedy knapsack-based scheduling (GKS) algorithm for job scheduling using iFogSim simulator.[24] They applied their algorithm on video surveillance (VSOT) and EEG tractor beam game (EEGTBG). They compared their execution cost and energy consumption with FCFS, concurrent and delay priority scheduling, and claimed to get better results. However, the authors ignored network usage that is an important parameter.

Deep reinforcement learning has also been used by the work of Mai et al[25] for Real-Time Task Assignment (RTTA) using a neural network, trained by reinforcement learning approach using evolutionary strategies for scheduling real-time jobs.

As Fog computing is still in its infancy stage, only a few algorithms have considered multi-objective optimization for job scheduling. Compared to the proposed approach, the presented methods suffer from lack of job scheduling efficiency in jointly preserving energy and delay in Fog computing. To achieve this purpose, we implement a job scheduling algorithm for delay and performance optimization in fog computing for latency-sensitive applications.

3 | ARCHITECTURAL MODEL

Fog computing extends Cloud computing as it works as a middle layer between Cloud and source nodes. Fog computing architecture is hierarchical, bi-directional, and distributed that is composed of multiple layers, as shown in Figure 1.

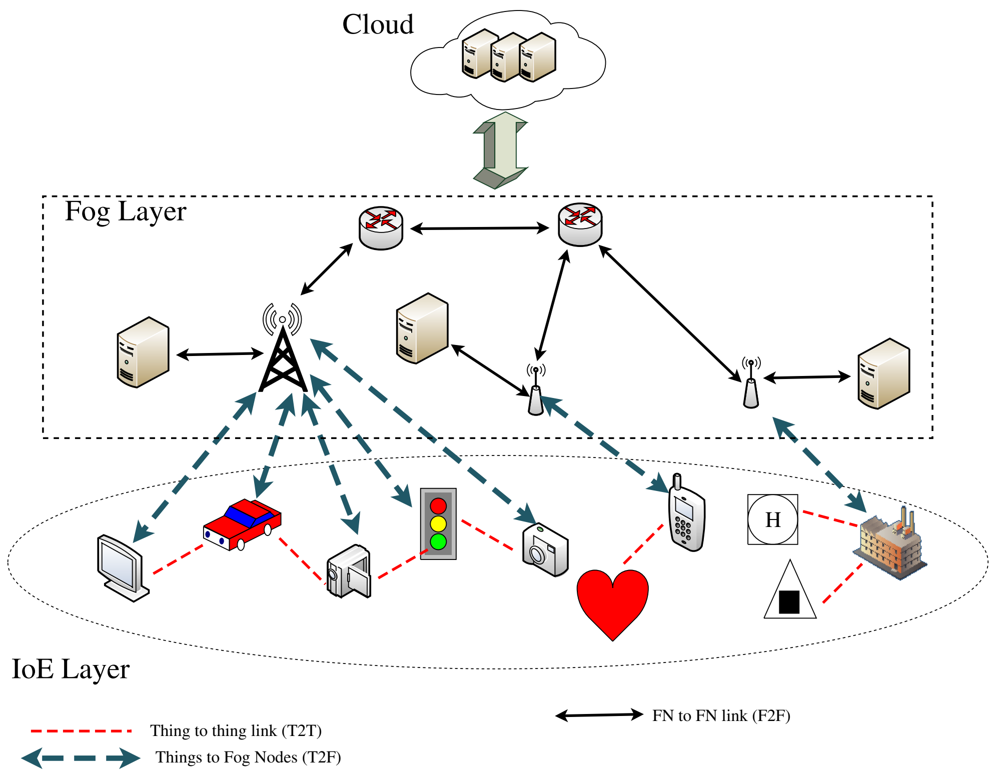

FIGURE 1 Fog architecture: in the figure, FN stands for Fog node

The top most layer is Cloud layer, the intermediate layers consist of Fog devices, while the bottom most IoE layer consists of sensors and actuators.

- **IoE Layer.** IoE layer consists of sensors and actuators. Sensors like cameras, heartbeat sensor, humidity sensors, temperature sensors, GPS sensors, etc, collect raw data from external environment, convert it into signals, and transmit them to Fog nodes for further processing. After processing, Fog nodes send the results back to actuators that work as controllers to perform action accordingly. There are two types of communication links for communication between devices in the IoE layer, ie, Thing-to-Thing (T2T) and Thing-to-Fog (T2F) link, as shown in Figure 1. The nearby IoE devices can interact with Bluetooth, Zigbee, and WiFi through the Thing-to-thing (TtoT) communication link. Using Thing-to-Fog (TtoF) communication link, these IoE devices can communicate with upper level Fog nodes.
- **Fog Layer.** Fog layer acts as an intermediate layer between Cloud and end devices. It loosely integrates both Cloud and IoE layers associated to the Fog, thereby enabling independent evolution and high degree of interaction between multiple layers. The Fog layer acts as a backbone to the underlying IoE platform, while efficiently utilizing the Cloud services above.

  Fog layer is comprised of heterogeneous Fog devices having limited computing, storage, and networking capability, eg, routers, switches, proxy servers, and cellular base stations. A Fog node can either be a low-level or a high-level Fog device. The low-level Fog devices are edge devices with poor resources in terms of processing power, RAM, and storage, whereas high-level Fog devices are Fog servers that are attached to edge devices and have rich resources like processor, RAM, and storage. These servers are connected with a Cloud server. Fog Nodes in Fog layer communicate with IoE devices using Thing-to-Fog (T2F) link, while they communicate with other Fog nodes using Fog-to-Fog (F2F) link, as shown in Figure 1.
- **Cloud Layer.** Cloud is the top most layer of the architecture. It gets data from networking devices for long term behavior and data analysis and returns the results back to Fog devices for further necessary actions.

In Fog computing, applications mostly use Sense-Process-Actuate-Model (SPAM), where sensors are used to sense and gather data form IoE devices. The data is forwarded to Fog devices for processing and then results are sent to actuators for taking actions.[4] These applications can use Peer-to-Peer (P2P), client-server, or cluster techniques for nodal collaboration.[26] These applications are developed as Directed Acyclic Graph (DAG) that contains different modules following Distributed Data Flow (DDF) model. In DAG, a module takes input, processes it, and sends output to another module as input.

4 | DESIGN AND IMPLEMENTATION

Some applications of Fog computing are latency-sensitive, while other ones are delay-tolerant. The workloads generated by these applications are dynamic, variable length, and sometimes require priority execution at both Edge and Cloud. Applications compete for limited resources of heterogeneous devices in heterogeneous environment. These workloads are assigned and executed at various Fog nodes. For job scheduling in Fog Computing, if simple Round Robin (RR) algorithm employing First Come First Served (FCFS) technique is used, it gives equal priority to all the jobs, which results in increased response time for jobs with small burst times. However, the aim of Fog Computing paradigm is to minimize waiting time, response time, and network traffic.

Therefore, there is a need to design and implement a job scheduling algorithm in Fog with the following objectives:

1. Minimize application loop delay (latency);
2. Efficiently utilize the resources of Fog devices (energy, RAM, processor, etc);
3. Minimize the network usage.

4.1 | Case study

Ubiquitous nature of IoE devices allows monitoring of different activities of healthcare system. Fog computing is emerging as an important architectural ingredient for Ubiquitous computing.[27] In e-healthcare, Cloud-based solutions result in longer latency that is unacceptable in emergency situations. Using Fog computing, a significant number of healthcare computing jobs can be performed by nearby Fog nodes, resulting in smaller delays and more availability.[28]

In smart Health care applications, there are distinct use case classes, some are critical while the others are delay-tolerant.[29] For example, a patient's data is collected and saved so that a doctor can examine it later. Such kind of data saving and retrieval is delay-tolerant and not too critical. While in some activities, like critical situation of patient, faster data analysis is required to generate emergency alerts. Such activities are more latency critical as a late response in emergency alert can be threatening to patient's life.

Our smart healthcare case study consists of three type of application use cases as follows.

1. **Emergency Response System**. Emergency Response System is a system to process patient's critical data like blood glucose, blood pressure, heartbeat, and body temperature from various body sensors in case of emergency. This data contains critical information about patient's condition that is to be processed under emergency circumstances, eg, blood glucose greater than 400 mg/dl or blood pressure greater than 140/90 mmHg. Afterward, this data is processed and analyzed to generate timely notifications are required to save a patient's life.
2. **Patient's Appointment Management System**. Patient's Appointment System is a means for e-health care that makes it possible to book and manage patient appointments quickly while eliminating the chance of repeating the same time slot for distinct patients, therefore is less critical.
3. **Patient's Record Management System**. Patient's Record Management System is a database that contains patient record. This record contains the patient's personal information, doctor, visits, treatment, and lab results. The receptionist creates new record for every patient since the patient enters the hospital to store it in Patient Record Database stored on cloud.

For realization of this smart healthcare case study, we have used three application modules as follows.

1. **DCPB**. DCPB module is placed on low level fog device that receives data from all three application modules. It receives critical data from body sensors, processes, and analyzes it to generate emergency notifications to be displayed. It also receives data for patient's appointment and patient's records but forwards it to organizer module.
2. **Organizer**. Organizer module is placed on high level fog device that receives data from DCPB. It generates time slot that are assigned to patients against appointment requests. This appointment schedule is conveyed to patients. It forwards patient's record and some of the critical information to Patients Record Database.
3. **Patients Record Database**. This module is placed on cloud. It receives data from organizer for storage and long term analysis purpose. It generates patterns of patient's health situations, their hospital visits, and their information and sends these to organizer.

The modules of the case study are shown in Figure 2.

The maximum CPU-lengths of tuples for communication between modules are given in Table 1. As the CRITICAL tuple contains the most critical information, it is needed to be processed on priority basis. The proposed algorithm sorts all the arrived jobs in ascending order on the basis of the MIPS required and selects the first job from this list for execution. The proposed algorithm executes the shortest tuples first, so the delay for "CRITICAL" tuple is minimized. Let us consider, at a particular time, CRITICAL (1), SCHEDULE (2), and RECORD (3) tuples arrive at DCPB according to the sequence given in Table 2.

The average waiting times for CRITICAL, SCHEDULE, and RECORD tuples using FCFS are given in Table 3.

Table 3 shows that, by using the proposed algorithm, the waiting time of the most critical tuple "CRITICAL" is smallest, after that "SCHEDULE" tuple is processed, and the highest waiting time is of "RECORD" tuple that is delay-tolerant.

The symbols used in the proposed algorithm are shown in Table 4. Algorithm 1 lists the steps of the proposed job scheduling algorithm.

As described in the proposed algorithm, when a Fog node receives a tuple T_i from a sensor or another Fog node, the scheduler checks the length of the waiting list of tuples L_w. If L_w is empty, the scheduler allocates processing resources to incoming tuple and it is executed until completion. Upon completion, T_i is set as T_f and is added to the finished list of tuples L_f. If waiting list L_w is not empty, T_i is inserted in it and this

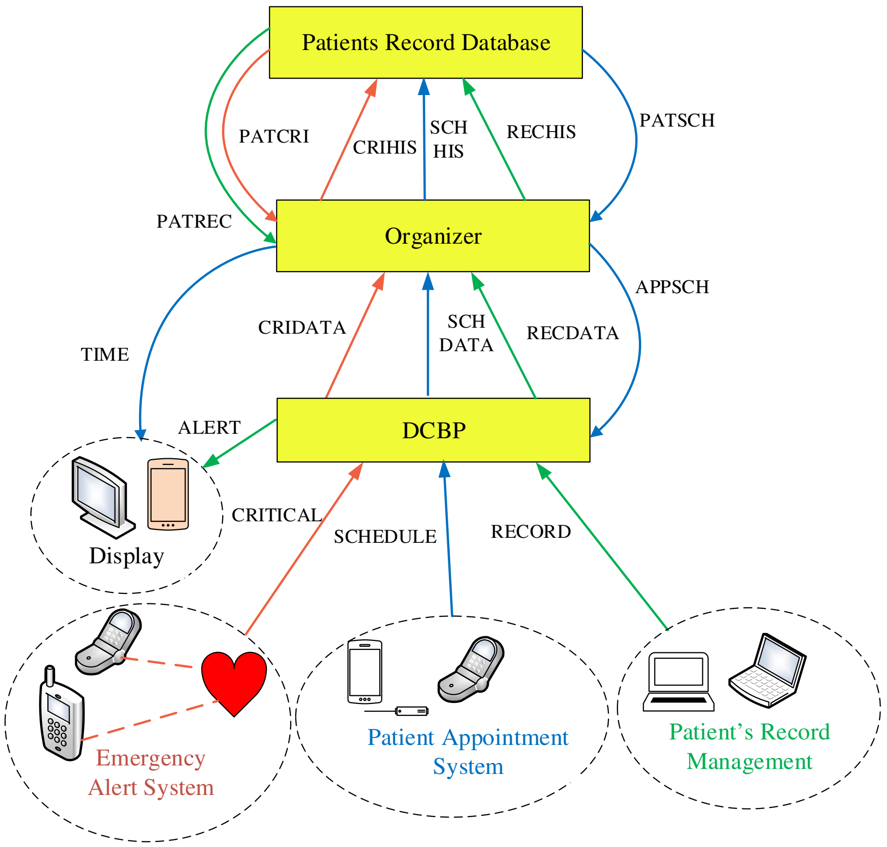

FIGURE 2 Example case study and its modules

TABLE 1 MIPS required by each module

Tuple Type	CPU Length	Tuple Type	CPU Length
CRITICAL	800	SCHEDULE	1000
SCHEDULE	1200	RECORD	3500
CRIDATA	400	SCHDATA	400
RECDATA	1000	CRIHIS	600
SCHHIS	800	RECHIS	800
PATCRI	400	PATSCH	400
PATREC	600	APPSCH	500
TIME	100	ALERT	100

TABLE 2 Tuples arrival

3	2	1	2	1	3	1	3	2

TABLE 3 Waiting time of tuples

Tuple	FCFS	Proposed Algorithm
CRITICAL	7466.6	2400
SCHEDULE	8100	3600
RECORD	6433.3	9500

TABLE 4 Summary of the symbols used in the proposed algorithm

Symbol	Meaning
L_w	Waiting List of Tuples
L_f	Finish List of Tuples
T_i	Incoming Tuple
T_f	Finished Tuple
$T_{\min}$	Tuple with shortest MIPS

list is sorted in increasing order of the length of tuples. Upon the completion of a tuple, scheduler selects shortest tuple T_min from the head of the waiting list L_w, allocates VM to it till it is finished. When finished, $T_{\min}$ is set to T_f and is added to the finished list of tuples L_f.

Complexity Analysis. Time complexity of the proposed algorithm depends on the length of the waiting list (L_w) that is N. As the scheduler sorts this list; therefore, the time complexity of the proposed algorithm is $O(nlogn)$ and the space complexity is $O(n)$. The functionality of the proposed algorithm is explained in Figure 3.

Algorithm 1 Proposed algorithm for job scheduling

```
INPUT: Tuple List ($T1, T2, \dots, TN$)
OUTPUT: Finished Tuple $T_f$
1:  do
2:      if new Tuple arrived then
3:          $T_i \Longleftarrow$ received tuple
4:          if $L_w = \emptyset$ then
5:              Allocate vm to $T_i$
6:              Execute $T_i$
7:              $T_f \Longleftarrow T_i$
8:              $L_f \Longleftarrow T_f$
9:          else
10:             $L_w \Longleftarrow T_i$
11:             Sort ($L_w$) in ascending order of MIPS
12:         end if
13:     end if
14:     if $T_f$ then
15:         $T_{min} \Longleftarrow$ Select $T_i$ with minimum MIPS from $L_w$ return Allocate VM to $T_{min}$
16:         Execute $T_{min}$
17:         $T_f \Longleftarrow T_{min}$
18:         $L_f \Longleftarrow T_f$
19:     end if
20: while Not end of tuples OR $L_w \neq \emptyset$
21: return tuple
```

When a tuple arrives at a Fog device from either sensor node or another Fog device, it is sent to SJF job scheduler. The Scheduler checks whether it is first tuple on the Fog device. If so, the received tuple is executed. Otherwise, it is added to the queue that is separately maintained for every Fog device. This queue is sorted by scheduler whenever a tuple is inserted. Once the execution of tuple is completed, it is added to finished tuple queue. After that, the scheduler selects shortest tuple for execution, from the head of waiting list(L_w) and this process continues.

For the implementation of the proposed algorithm, we have used iFogSim toolkit that is an enhancement of iCloudSim. iFogSim is quite efficient for simulating resource management techniques in IoE and Fog computing environments.[4] We modify some classes of iFogSim and also add some classes to implement the scheduler. A brief introduction of the used classes is given as follows.

- **Sensors.** This class can be used to simulate IoT sensors. The data from sensor instances can be transmitted to Fog devices in the form of tuples. We use this class to generate variable-sized tuples.
- **Fog Device.** Instances of this class are used to simulate different Fog devices. For each Fog device, its memory, processor, storage size, and uplink and downlink bandwidths are specified. There can be multiple-level Fog nodes. Each Fog node can communicate with other upper-level

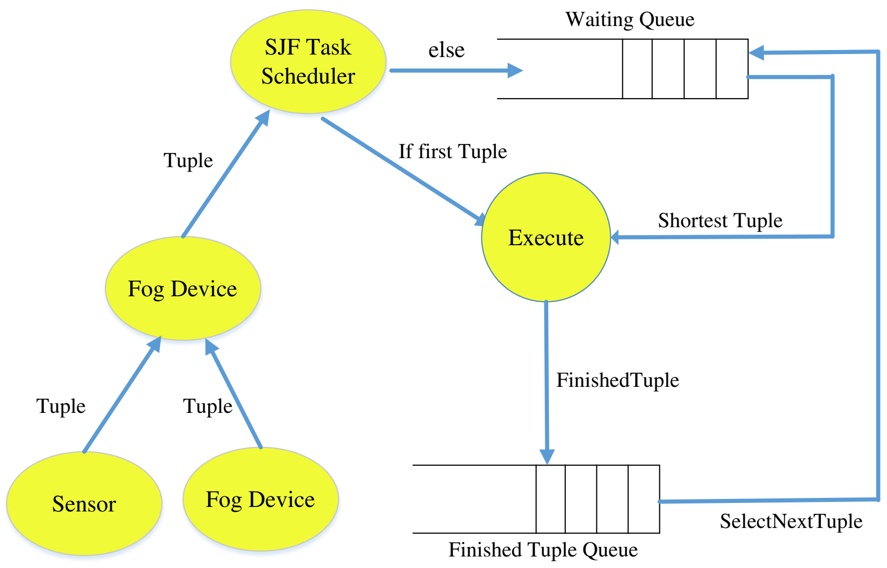

FIGURE 3 Block diagram of scheduling

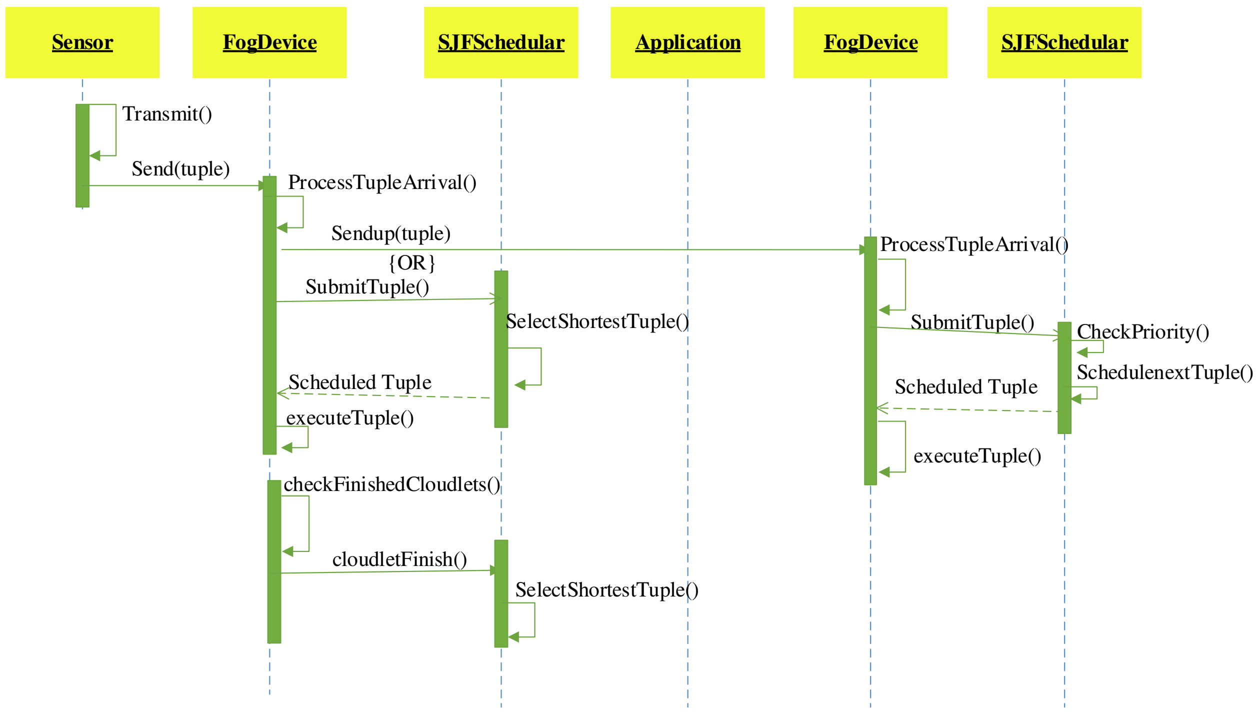

FIGURE 4 Sequence diagram of tuple scheduling and execution

Fog nodes and devices in IoE layer using the tuples. Each Fog node can process arrived tuples selected by the scheduler on the basis of increasing order of MIPS.

- **Tuples.** Communication between all the entities in Fog is done via instances of the tuple class. Each tuple contains information about source, destination, and its processing requirements as MIPS.
- **SJFScheduler.** This class is extended from *CloudletScheduler* that maintains two lists, ie, waiting list (L_w) and finished list (L_f). Waiting list contains all the tuples waiting for execution, while finish list consists of all the tuples that have completed their execution. When a tuple is finished, a smallest tuple is selected from the waiting list.

We show the tuple emission, scheduling, and execution in Figure 4.

As shown in Figure 4, when a tuple is emitted using *Transmit()* method from Sensor, it is sent to low-level connected Fog device via *Send(tuple)* method. A callback function *processTupleArrival()* is called on Fog device upon tuple arrival. This method checks either the tuple is to be processed at Fog device or it should be routed upwards to some other high-level Fog device. If the tuple is to be processed by a Fog device, it is sent to *SJFScheduler* via *SubmitTuple()* method. This method places the tuple in waiting list L_w and sorts this list on the basis of tuple length. After that, *SchedulenextTuple()* selects the smallest tuple and the scheduled tuple is sent to Fog device for execution. After tuple is completely executed, the scheduler is requested for next tuple to be executed via *CloudletFinish()* method. This method places the finished tuple in the finished tuple list L_f, and then a shortest tuple is selected by scheduler from the head of waiting list L_w.

5 | PERFORMANCE EVALUATION

In this section, we report the results of the proposed SJF algorithm in different scenarios against the state-of-the-art FCFS algorithm. We select three metrics, which are *loop delay*, *energy consumption*, and *network usage*. We present the notations used in various equations in Table 5.

TABLE 5 Notations used in simulation setup

Symbol	Meaning
ST_i	Execution start time of Tuple
ET_i	Execution end time of Tuple
$\mathcal{E}_c$	Current energy consumption
$\mathcal{T}_c$	Current Time
$\mathcal{T}_l$	Update Time of last utilization
$\mathcal{P}_H$	Host power in last utilization
L_i	Latency of *i*th tuple
$\mathcal{N}_i$	Network size of *i*th tuple.

TABLE 6 Configuration of Fog nodes

Name	MIPS	RAM	UpBw	DnBw	Level	Rate per MIPS	Busy Power	Idle Power
Cloud	44800	40000	100	10000	0	0.01	1648	1332
Proxy server	5600	4000	10000	10000	1	0.0	107.339	83.4333
2nd level Fog Device	5600	4000	10000	10000	2	0.0	107.339	83.4333
1st level Fog Device	800	1000	10000	270	3	0	87.53	82.44

5.1 | Simulation setup

For the evaluation of the proposed algorithm, we use iFogsim library as the simulation environment.[4] To illustrate our algorithm, we simulate a Smart Healthcare case study that is explained in Section 4.1. This case study contains both latency-sensitive and delay tolerant tuples.

We generate tuples of variable length from three modules, namely, Emergency Response System, Patient's Appointment System, and Patient's Record Management System. The "critical" tuples are generated from sensors that contains patient's condition data to be processed and analyzed and processed at the earliest. Patient's Appointment System generates "schedule" tuple containing an appointment request and after processing Appointment time is displayed, so it is less critical. "Record" tuple from Patient's Record Management System contains patient data to be stored on Cloud for later usage; hence, it is delay-tolerant. As "critical" tuples are the most critical, we require the fastest alert. "Schedule" tuples are less critical, while "record" tuples are delay-tolerant, so we compute delays for four end-to-end modules.

1. Emergency Alert Loop: DCPB->Organizer->display.
2. Patient's Appointment Loop: DCPB->Organizer->DCPB->display.
3. Patient's Record Management Loop: DCPB->Organizer->Patients Record Database.
4. History: Organizer->Patients Record Database->Organizer->DCPB.

Loop number 1 is the most critical, as it is an emergency condition, so it is the least delay-tolerant. Loops number 2 and 3 are less critical as they are dealing with the appointment time, while the loop number 4 is delay-tolerant because the patient information is needed to be stored on Cloud for long-term analysis.

5.2 | Configurations

We consider five configurations in which we use five high-level Fog devices (serving as Organizer) at level 2. A detailed evaluation is done by varying the number of Fog nodes (serving as DCPB) at level 3. At a first-level Fog device, three types of tuples are arriving from three module of type Emergency Alert, patient's Appointment, and patient's record management. For different configurations, against a second-level Fog node, 20, 40, 60, 80, and 100 third-level Fog nodes are attached. The simulation time is 400 units for every setup. The parameters for Fog nodes on every level are shown in Table 6.

5.3 | Average loop delay

We use a control loop to measure the end-to-end latency of all modules in the loop. To compute the loop delay, we compute the average CPU time, $\overline{\mathcal{T}_{cpu}}$, utilized by all tuples of a particular type of tuple. We compute this average using Equation (1)

$$\overline{\mathcal{T}_{cpu}} = \begin{cases} \frac{ST_i \times N + ET_i - ST_i}{N+1}, & \text{if the average CPU time for a particular type of tuple is already computed} \\ ET_i - ST_i & \text{otherwise,} \end{cases} \quad (1)$$

where ST_i is the starting execution time, ET_i is the ending execution time of i^{th} tuple, and N is the total number of executed tuples of a particular type. We compute the execution delay of each tuple using Equation (2)

$$Delay_i = ST_i - ET_i, \quad \forall i \in \mathbb{T}, \quad (2)$$

where $\mathbb{T}$ is the existing tuple set. We compare the application loop delays produced by the proposed scheduler with the results of FCFS (First Come First Served) as it is the only implemented scheduling algorithm in iFogSim. Figure 5 shows the application loop-delay computed in milliseconds using FCFS and the purposed Tuple scheduling algorithms. Along x-axis, the number of nodes are presented; while along y-axis, the average application loop delays are presented for 400 simulation time. The triangle-marked curve indicates the average loop delay using the proposed algorithm while the circle-marked curve shows the same value using FCFS algorithm.

Figure 5A shows the average loop delay for the most critical loop, ie, emergency alert. In this case, the delays using SJF are remarkably lesser than those of FCFS. Using SJF, the delay does not increase with increasing the number of nodes. Figure 5B indicates the loop delay for patient's

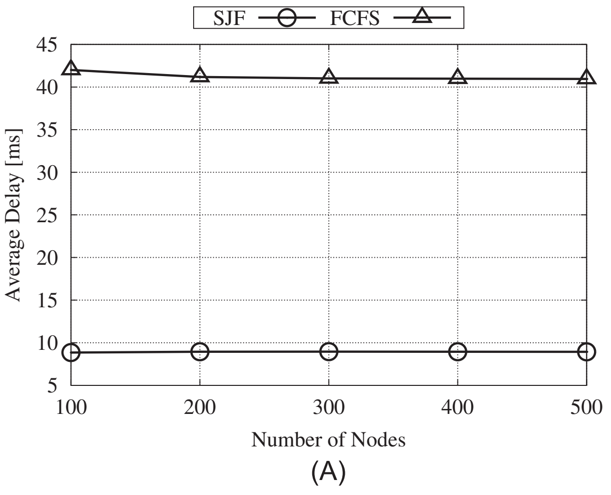

FIGURE 5 Average loop delay in [ms] for (a) emergency alert, (b) patient's appointment, (c) record management, and (d) history for various number of IoE nodes

appointment that is not too critical. Here, FCFS performs better than SJF because SJF gives more priority to the most critical loop. Figure 5C shows the loop delay for Patient Record Management, where again the proposed algorithm results in smaller delays than those of FCFS. The delay using FCFS sharply increases as the number of nodes becomes greater than 400. Figure 5D shows the delay for patient's history, where loop delay using SJF is smaller than that of FCFS. After 400 nodes, the delay using SJF decreases.

5.4 | Energy consumption

We compute the energy consumed by a Fog device, $\mathcal{E}_{FN}$, using Equation (3)

$$\mathcal{E}_{FN} = \mathcal{E}_c + (\mathcal{T}_c - \mathcal{T}_l) \times \mathcal{P}_H. \tag{3}$$

The energy of any Fog device can be computed by the power of all the hosts in a set period of time for execution, where $\mathcal{E}_c$ is the current energy consumption, $\mathcal{T}_c$ is the current time, $\mathcal{T}_l$ is update time of last utilization, and $\mathcal{P}_H$ is the host power in last utilization.

In this part, we present the cost of applying SJF method against FCFS algorithm on Cloud system.

Total execution cost can be computed using Equation (4)

$$Cost = \sum_{i=1}^{FN} \left[\mathcal{C}_c + (\mathcal{T}_c - \mathcal{T}_l) \times \mathcal{R}_{mips} \times \mathcal{U}_l \times \mathcal{M}_{host}\right]. \tag{4}$$

In Equation (4), $\mathcal{C}_c$ is current cost, $\mathcal{T}_c$ is the current time, $\mathcal{T}_l$ is update time of last utilization, $\mathcal{R}_{mips}$ is rate per mips, $\mathcal{U}_l$ is last utilization, and $\mathcal{M}_{host}$ is total mips of all hosts. Last utilization can be computed as $\mathcal{U}_l = min(1, \mathcal{AM}_{host}/\mathcal{M}_{host})$, where $\mathcal{AM}_{host}$ is total allocated mips of host.

Figure 6 shows the average energy consumed by Cloud and Fog nodes. Along x-axis, the number of nodes are represented; while along y-axis, the energy consumed is represented. Specifically, Figure 6A shows the average energy consumption of SJF versus FCFS for Cloud component

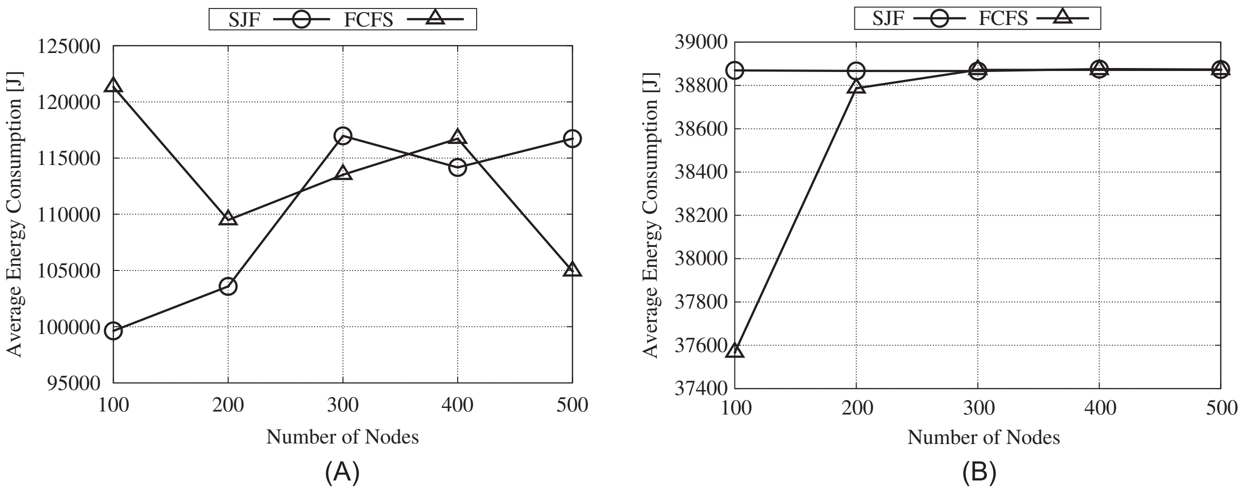

FIGURE 6 Average consumed energy in [J] by varying the number of nodes: (a) cloud devices and (b) fog devices

used in the architecture presented in Figure 1. In this figure, we have two observations. First, the average energy consumption of SJF algorithm for a small number of participated IoE nodes (below 300 nodes) is less than that of FCFS. Second, this value increases when the size of the problem increases. Therefore, it leads us to apply FN (Energy consumed by device) to provide adaptive load balancing for the dispatched workload and decreases the energy consumption of server used for the processing of the request. Therefore, as shown in Figure 6B, the average energy consumed FNs using the proposed algorithm (see the circle-marked curve) is less than that of FCFS algorithm. Interestingly, this figure confirms that FNs decrease the average energy consumption of the server to about 60% below the case where only the Cloud system is used (see the curves in Figure 6A) and SJF algorithm could manage itself for large scale network and its value will be similar to FCFS.

5.5 | Network usage

The third evaluation parameter is *network usage* $\mathbb{N}_{use}$. Increasing the number of devices increases the network usage and it results in network congestion. This congestion ultimately results in poor performance of the application running on Cloud network. Fog computing helps in decreasing the network congestion by distributing the load on intermediate Fog devices. We compute the network usage through Equation (5)

$$\mathbb{N}_{use} = \sum_{i=1}^{N} L_i \times \mathcal{N}_i, \tag{5}$$

where N is the total number of tuples, L_i is the latency, and $\mathcal{N}_i$ is the network size of i^{th} tuple. Figure 7 shows the comparison of network usage of SJF against FCFS algorithm. The number of nodes is shown along x-axis, while the average network usage for 400 simulation time is represented along y-axis. The triangle-marked curve indicates the average network usage using the proposed algorithm, while the circle-marked curve shows the same value using FCFS algorithm. The result shows that the network usage using the proposed algorithm can save about 30% of the network when the number of nodes are below 300 and, when the number of nodes increases, network saving decreases to about 20% when we compare it against the FCFS algorithm.

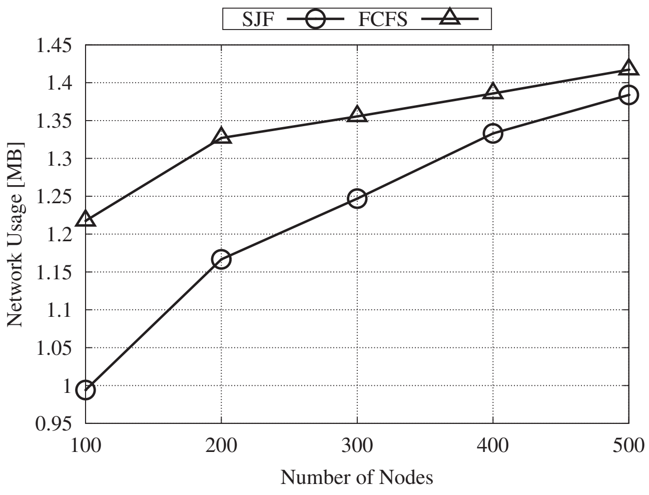

FIGURE 7 Network usage of Fog devices in [MB] versus different number of nodes (IoE devices)

The results show that the proposed scheduling algorithm performs better in terms of latency and network usage as compared to FCFS, while the average energy consumption of Fog nodes is a bit higher for the proposed approach than that of FCFS when Fog nodes are lesser.

6 | CONCLUSION AND FUTURE WORK

Increasing number of Internet of everything (IoE) devices are generating huge amount data due to which Cloud computing is unable to meet requirements of real-time applications like low latency, location awareness, and mobility support. To overcome these limitations, Fog computing has emerged as a new computing paradigm that complements Cloud computing by providing support for real-time processing and analytics and storage facilities near the edge device. As the edge devices are resource-constrained, therefore job scheduling in Fog computing is a great challenge.

In this paper, we have designed and implemented an optimized job scheduling algorithm to minimize the delays for latency-critical applications. We have given an example case study of healthcare to demonstrate the latency-sensitive and delay-tolerant requests from IoE devices. The proposed algorithm schedules jobs on Fog devices on the basis of length and reduces the average loop delay and network usage.

Although the proposed SJF algorithm reduces the average waiting time, it can starve tuples with larger lengths. To solve this issue, in the future, we intend to implement some meta-heuristic, hyper-heuristic, reinforcement learning-based methods, etc, for scheduling. We also intend to use an analytical model for capacity planning of these resource-constrained devices.

ORCID

Mohammad Shojafar https://orcid.org/0000-0003-3284-5086
Humaira Ijaz https://orcid.org/0000-0002-9810-9180

REFERENCES

1. Baccarelli E, Naranjo PGV, Scarpiniti M, Shojafar M, Abawajy JH. Fog of everything: energy-efficient networked computing architectures, research challenges, and a case study. *IEEE Access*. 2017;5:9882-9910.
2. Hussain F, Al-Karkhi A. Big data and fog computing. In: *Internet of Things: Building Blocks and Business Models*. Cham, Switzerland: Springer; 2017:27-44.
3. Anawar MR, Wang S, Azam Zia M, Jadoon AK, Akram U, Raza S. Fog computing: an overview of big IoT data analytics. *Wirel Commun Mob Comput*. 2018;2018. Article ID 7157192.
4. 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 Exper*. 2017;47(9):1275-1296.
5. 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*. 2018;5(2):677-686.
6. Aazam M, Zeadally S, Harras KA. Fog computing architecture, evaluation, and future research directions. *IEEE Commun Mag*. 2018;56(5):46-52.
7. Dastjerdi AV, Gupta H, Calheiros RN, Ghosh SK, Buyya R. Fog computing: principles, architectures, and applications. In: *Internet of Things: Principles and Paradigms*. Cambridge, MA: Morgan Kaufmann; 2016:61-75.
8. Shojafar M, Cordeschi N, Baccarelli E. Energy-efficient adaptive resource management for real-time vehicular cloud services. *IEEE Trans Cloud Comput*. 2019;7(1):196-209.
9. Mishra S, Jain S. Ontologies as a semantic model in IoT. *Int J Comput Appl*. 2018. https://doi.org/10.1080/1206212X.2018.1504461
10. Liu L, Qi D, Zhou N, Wu Y. A task scheduling algorithm based on classification mining in fog computing environment. *Wireless Commun Mobile Comput*. 2018;2018. Article ID 2102348.
11. Mishra S, Sagban R, Yakoob A, Gandhi N. Swarm intelligence in anomaly detection systems: an overview. *Int J Comput Appl*. 2018. https://doi.org/10.1080/1206212X.2018.1521895
12. Bittencourt LF, Diaz-Montes J, Buyya R, Rana OF, Parashar M. Mobility-aware application scheduling in fog computing. *IEEE Cloud Comput*. 2017;4(2):26-35.
13. Pham X-Q, Huh E-N. Towards task scheduling in a cloud-fog computing system. Paper presented at: 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS); 2016; Kanazawa, Japan.
14. Zeng D, Gu L, Guo S, Cheng Z, Yu S. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Trans Comput*. 2016;65(12):3702-3712.
15. Ni L, Zhang J, Jiang C, Yan C, Yu K. Resource allocation strategy in fog computing based on priced timed petri nets. *IEEE Internet Things J*. 2017;4(5):1216-1228.
16. Pooranian Z, Shojafar M, Vinueza Naranjo PG, Chiaraviglio L, Conti M. A novel distributed fog-based networked architecture to preserve energy in fog data centers. Paper presented at: 2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS); 2017; Orlando, FL.
17. Hoang D, Dang TD. FBRC: optimization of task scheduling in fog-based region and cloud. Paper presented at: 2017 IEEE Trustcom/BigDataSE/ICESS; 2017; Sydney, Australia.
18. Sun Y, Lin F, Xu H. Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II. *Wirel Pers Commun*. 2018;102(2):1369-1385.
19. Choudhari T, Moh M, Moh T-S. Prioritized task scheduling in fog computing. In: Proceedings of the ACMSE 2018 Conference; 2018; Richmond, KY.
20. Bitam S, Zeadally S, Mellouk A. Fog computing job scheduling optimization based on bees swarm. *Enterprise Inf Syst*. 2018;12(4):373-397.
21. Gai K, Qiu M. Optimal resource allocation using reinforcement learning for IoT content-centric services. *Appl Soft Comput*. 2018;70:12-21.

22. Zhang Q, Lin M, Yang LT, Chen Z, Khan SU, Li P. A double deep q-learning model for energy-efficient edge scheduling. *IEEE Trans Serv Comput*. 2018;12(5):739-749.
23. Nguyen BM, Thi Thanh Binh H, Do Son B. Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment. *Applied Sciences*. 2019;9(9):1730.
24. Rahbari D, Nickray M. Low-latency and energy-efficient scheduling in fog-based IoT applications. *Turk J Electr Eng Comput Sci*. 2019;27(2):1406-1427.
25. Mai L, Dao N-N, Park M. Real-time task assignment approach leveraging reinforcement learning with evolution strategies for long-term latency minimization in fog computing. *Sensors*. 2018;18(9):2830.
26. Mahmud R, Kotagiri R, Buyya R. Fog computing: a taxonomy, survey and future directions. *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*. Singapore: Springer; 2018:103-130.
27. Rahmani AM, Gia TN, Negash B, Anzanpour A, Azimi I, Jiang M, Liljeberg P. Exploiting smart e-health gateways at the edge of healthcare Internet-of-Things: a fog computing approach. *Futur Gener Comput Syst*. 2018;78:641-658.
28. Chao K-M, Chung J-Y. Edge and fog services. *SOCA*. 2019;13(1):1-2. https://doi.org/10.1007/s11761-019-00256-y
29. Kraemer FA, Braten AE, Tamkittikhun N, Palma D. Fog computing in healthcare–a review and discussion. *IEEE Access*. 2017;5:9206-9222.

How to cite this article: Jamil B, Shojafar M, Ahmed I, Ullah A, Munir K, Ijaz H. A job scheduling algorithm for delay and performance optimization in fog computing. *Concurrency Computat Pract Exper*. 2019;e5581. https://doi.org/10.1002/cpe.5581