An Intelligent Job Scheduling Algorithm for Optimization of Loop Latency and Enhancement in Performance in Fog Computing Environment

Meena Rani[1]
Kalpna Guleria[1,*]
Surya Narayan Panda[1]
[1]Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India

E-mail: meena.rani@chitkara.edu.in (Meena Rani)
Corresponding Author: guleria.kalpna@gmail.com (Kalpna Guleria)
E-mail: snpanda@chitkara.edu.in (Surya Narayan Panda)

ABSTRACT

Every day, an increasing number of Internet of Everything (IoE) devices generate enormous volumes of data. Cloud computing provides processing, analysis, and storage services to handle these kinds of large data quantities. The increasing latency and bandwidth consumption are unacceptable for real-time applications such as smart healthcare devices, online gaming, and video monitoring. In order to tackle the rise in latency as well as bandwidth usage in cloud computing technology, FC (Fog Computing) has been developed. At the periphery of a network, FC offers networking, processing, storage, and analytics functions. Since FC is still in its infancy, scheduling jobs and allocating resources are two of its major issues. With the help of this innovation, there are resource limitations on the fog devices at the network's edge. Consequently, choosing a fog node for a job's assignment and scheduling is crucial. The energy usage and application request response time can be decreased with an intelligent and effective work scheduling algorithm. This research presents a novel Quality of Service Priority Tuple Scheduling (QoSPTS) scheduler that maximizes network capacity and latency while supporting service provisioning for the IoE. Presented here is a case study showing the effective management of IoE device requirements, efficiently allocating resources across fog devices, and optimizing scheduling to enhance quality of life. Taking energy efficiency and latency into account as performance measures, here, using iFogSim to compare the suggested scheduling algorithm to other methods. According to the findings, the suggested scheduler's latency and network bandwidth improved by 34% and 18%, respectively, in comparison with the FCFS (First Come First Serve) approach.

KEYWORDS: Fog Computing, Latency, Energy Efficiency, Resource use efficiency, QoSPTS, Job Scheduling

1.0 INTRODUCTION

The term Internet of Things (IoT) is known as a broad environment made up of different common things connected via heterogeneous networks, including vehicles, farms, factories, computers, appliances, and animals. Through the usage of networks and algorithms, IoTs have enabled these objects to carry out valuable tasks without requiring human intervention [1]. Initially, the IoT was primarily concerned with integrating intelligence into physical objects. However, Cisco later expanded on this idea to create the IoE, which prioritizes connecting people, processes, data, and objects to create intelligent connections. Through the facilitation of People-to-Machine (P2M), Machine-to-Machine (M2M), and People-to-People (P2P) communications, the IoE model aids in the automation of human tasks.

IoE systems that use Cloud-Centric IoT architecture for processing, analytics, as well as storage include smart traffic, smart farming, and smart healthcare monitoring [2]. Long delays are typically experienced by CIoT data centers since they are generally located many steps away from a source node. It is assumed that by 2020, the world's connected device count is anticipated to increase from 5.4 billion in 2019 to 1 trillion in 2025 [4]. Vast amounts of data will be created as a result, more than the cloud can handle, which will cause extended latency times and network congestion. Even now, low latency is a prerequisite for latency-sensitive, certain real-time, and geographically dispersed IoT applications, like virtual reality, smart traffic surveillance, smart healthcare monitoring, and others, which cannot be efficiently handled by cloud computing [5]. Many strategies, like edge computing, mobile computing, FC, and so on., have been put out to get over these CIoT restrictions by offering networking, storage, processing, and decision-making capabilities close to the node of source or end-user application [6].

Of all the suggested strategies, fog computing is one that has received the most interest lately. By enabling computation, networking, decision-making, FC is a distributed computing model that falls in line with cloud computing. The “Sense Process Actuate Model” (SPAM) is mostly utilized in FC. Sensors in SPAM detect and gather information, which is then sent to fog devices for processing. The outcomes are routed to actuators for action after processing. Some of the data that fog nodes forward to the cloud is used for longer-term analytics processing and storage management. Devices that could function as fog nodes include smartphones, smart gateways, internal modems, switches, routers, cellular base stations, and more [7]. Such devices are limited in computing power, storage capacity, bandwidth, and power and have varied architectures. Applications utilizing fog computing in real-time need a quicker response time than those that can tolerate delays. Consequently, many applications should contend for these few resources. One of the main issues in FC is resource management because fog applications are latency-sensitive and resource-constrained [8]. As a result, choosing how to allocate resources and schedule jobs is crucial. In smart systems, it is desirable to have quick and timely replies from work scheduling that is done well. For instance, in a system of smart healthcare, prompt notification of a patient's status is necessary to preserve their life [9]. To optimize the use of these diverse and resource-constrained fog devices, an effective job scheduling algorithm must be developed [10]. The ultimate goal is to reduce reaction times and network utilization while maintaining energy efficiency [11].

While there have been several proposed work scheduling methods, such as FCFS-based Round Robin, FCFS, delay-priority, Concurrent, etc., fog computing job scheduling is still in its early stages [12]. Current algorithms, like Round Robin and FCFS, carry out the tasks on the basis of the order in which they arrive. Because of this, they are unable to shorten latency-sensitive applications' response times. For applications that are sensitive to latency, a scheduling method that can reduce average response time as well as network utilization while maintaining optimal energy consumption must be developed. Given a thorough and methodical analysis of the optimization of loop latency and enhancement in performance in an FC environment. The goal is to offer a thorough and brief summary of the most recent research-related literature [13], [14], [15], [16], [17], [18].

1.1 Objectives and Contributions

The present research offers a novel multi-objective fog scheduler that facilitates IoE service provisioning for applications that are sensitive to latency. The goal is to lower latency as well as energy usage while maximizing the usage of fog devices that have been present at a network's edge.

The following are the suggested work's main contributions.

1.1.1 Optimization of Loop Latency

Reducing service response time is the most crucial factor in latency-sensitive applications. The jobs on the fog nodes are carried out using the suggested algorithm based on their durations. The loop latency is optimized first by carrying out the smallest job.

1.1.2 Network Bandwidth Optimization

Network utilization rises with the number of IoE devices linked to every application. As a result, network congestion occurs. The effort also aims to optimize network bandwidth to enhance the application's performance.

1.1.3 Energy Efficiency Enhancement and Resource Utilization

The suggested algorithm aims to improve energy efficiency and resource usage for fog devices, which are resource-constrained and use energy like all other network devices.

1.2 Organization of the Paper

This work's remaining portions are organized as follows. The associated work has been explained in Section II. The design framework is explained in Section III. Describe the conception and execution of the suggested technique in Section IV. Section V discusses the evaluation and validation of the performance outcome of the experiments. Section 6 concludes the points and future prospects of the upcoming projects and opportunities.

2.0 RELATED WORK

The current section examines a few resource management strategies that are currently in use and have been suggested for improving fog computing performance. Authors refer to jobs and tasks interchangeably. Most of these optimizations for resource management deal with job assignments, while some techniques focus on job scheduling on such resource-constrained fog devices. The goals of scheduling and allocation of jobs are to maximize network availability, improve resource utilization, minimize costs, reduce loop delays, minimize device energy consumption, minimize make-span, and lower network utilization. Thus far, algorithms have only been proposed that maximize one or two of the aforementioned criteria's parameters. An outline of a few of these suggested methods for job allocation is provided below.

A dependent job allocation technique for fog nodes was proposed by Pham and Huh [19]. Jobs that are dependent on one another require data exchange with one another. Workflows or the Directed Acyclic Graph (DAG) are utilized to depict dependent jobs. The authors prioritize each task by looking through a DAG, after which they assign these tasks to fog nodes. A job is sent to the cloud if the fog nodes are resource-constrained and cannot complete it. Important factors like the fog provider's budget and the deadlines for completing processes are overlooked by the writers. Another work image placement as well as scheduling approach in FC was presented by Zeng et al. [20]. On a server, the job image is kept for storage. The storage servers can be used together for calculations by embedded clients & fog nodes. Tasks are arranged to satisfy minimum completion timeframes and optimize user experience. Ni et al. [21] introduced a dynamic resource allocation technique that uses “Priced Timed Petri Nets” (PTPNs) to optimize resource usage and user QoS needs based on the job completion time as well as the credibility of fog nodes. Pooranian et al. [22] proposed a different task allocation method in which they described a resource allocation algorithm based on heuristics to maximize energy utilization. They view resource allocation as an issue that is cognizant of the bin packing penalty. Ni et al. [21] suggested an adaptive double fitness Genetic Task Scheduling technique to optimize task, make-span, as well as communication cost. The method takes into account the computational power, communication expenses, and latency requirements of fog devices to optimize performance. Hoang and Dang [23] created a different heuristic-based job scheduling system in an effort to improve performance and minimize latency. To allocate jobs to fog regions and clouds, they suggest a fog-based region architecture. A 2-level resource scheduling approach was suggested by Sun et al. [24]. Both fog nodes within the same cluster and different fog clusters receive the same allocation of resources. To schedule resources across fog nodes in the same cluster for multi-objective optimization, they utilize the theory of improved Non-Dominated Sorted Genetic Algorithm II. The authors claim that they achieve more stable work performance and short delays. Liu et al. [10] extracted association rules using an a priori technique. The task is then scheduled to a fog device by combining the produced rules with the task's least completion time. They assert to shorten typical wait times and execution times. FC is intended to support a broad range of applications. These applications may be delay-tolerant or latency-critical. Reducing loop delay is the most crucial factor for latency-critical applications. Another method for scheduling jobs on a latency-critical application uses the FCFS scheduling algorithm, as documented in Gupta et al.’s work [4]. On fog nodes, they utilize the FCFS algorithm to compute loop delay, energy consumption, and network usage. They demonstrate how placing modules on edge is far superior to placing them on cloud.

Bittencourt et al. [12] examined resource allocation in FC by utilizing 3 scheduling policies—(i) concurrent, (ii) FCFS, and (iii) delay-priority—to analyze various application types. The algorithms are used for two distinct kinds of applications: (i) “Electro Encephalo Graphy Tractor Beam Game” (EEGTBG) and (ii) Video Surveillance. While the 2nd application is almost real-time, the first is delay-tolerant. To demonstrate that resources should be distributed based on the mobility requirements of the application, loop latency as well as network utilization was calculated. In order to lower costs and total response times in FC, Choudhari et al. [25] suggested a prioritized job scheduling approach. A job's priority is defined by its deadline when it is received. A job is placed in the fog layer using its computed priority. There is disregard for other crucial goals like network utilization and energy consumption. Additionally, Bitam et al. [26] employed the bio-inspired Bees Life Algorithm (BLA) as an optimization method for job scheduling. The method was applied to fog nodes to distribute jobs optimally. Comparing the suggested approach against the “Particle Swarm Optimization” (PSO) algorithm, it was discovered that the above had worse run-time as well as memory allocation values. In addition, Gai and Qiu [27] optimized resource allocation & Quality of Experience (QoE) using Reinforcement Learning (RL). In order to create cost mapping tables and allocate resources as efficiently as possible, they suggested two RL algorithms. A cluster of fog nodes had been set up to respond to every user request and fulfill detailed goals, like delay.

Zhang et al. [28] suggested a “Double Deep Q-Learning” (DDQ) model to lower energy consumption in edge computing. Through the use of DDQ, the paradigm calculated the Q-value for every “Dynamic Voltage and Frequency Scaling” (DVFS) technique. To keep the gradient from vanishing, “Rectified Linear Units” (ReLu) was utilized as the activation function rather than the Sigmoid function. A genetic method was presented by Nguyen et al. [29] to optimize job scheduling in the cloud-fog environment. Their goal was to shorten work execution times. Every chromosome signifies a task assigned to a node. To generate a new population, crossover and mutation were employed. In comparison to BLA and Modified PSO, the authors asserted that their outcomes were superior. Using the iFogSim simulator, Rahbari and Nickray recently built the “Greedy Knapsack-Based Scheduling” (GKS) algorithm for work scheduling [30]. They used the EEGTBG and Video Surveillance to test their system. They asserted to have achieved better outcomes when comparing their execution cost as well as energy usage with FCFS, concurrent as well as delay priority scheduling. The authors, however, disregarded network utilization, which is a crucial factor.

Mai et al. [31] employed deep reinforcement learning for “Real-Time Task Assignment” (RTTA) by training a neural network using evolutionary techniques for reinforcement learning in order to schedule real-time jobs. Just a small number of methods have taken into account multi-objective optimization for work scheduling because FC is still in its early stages. The solutions that have been provided do not have the same efficiency in work scheduling when it comes to simultaneously preserving energy as well as delay in FC as does the proposed methodology [32]. Here, putting into action a job scheduling strategy for latency-sensitive applications in fog computing to optimize performance and delay.

3.0 DESIGN FRAMEWORK

Because FC acts as a mediator between source nodes and the cloud, it expands on cloud computing. As seen in Fig. 1, FC architecture is dispersed, hierarchical, as well as bi-directional, comprising several layers. The cloud layer is at the top, fog devices are in the intermediate layers, and smart devices with sensors as well as actuators are at the bottom of the IoE layer.

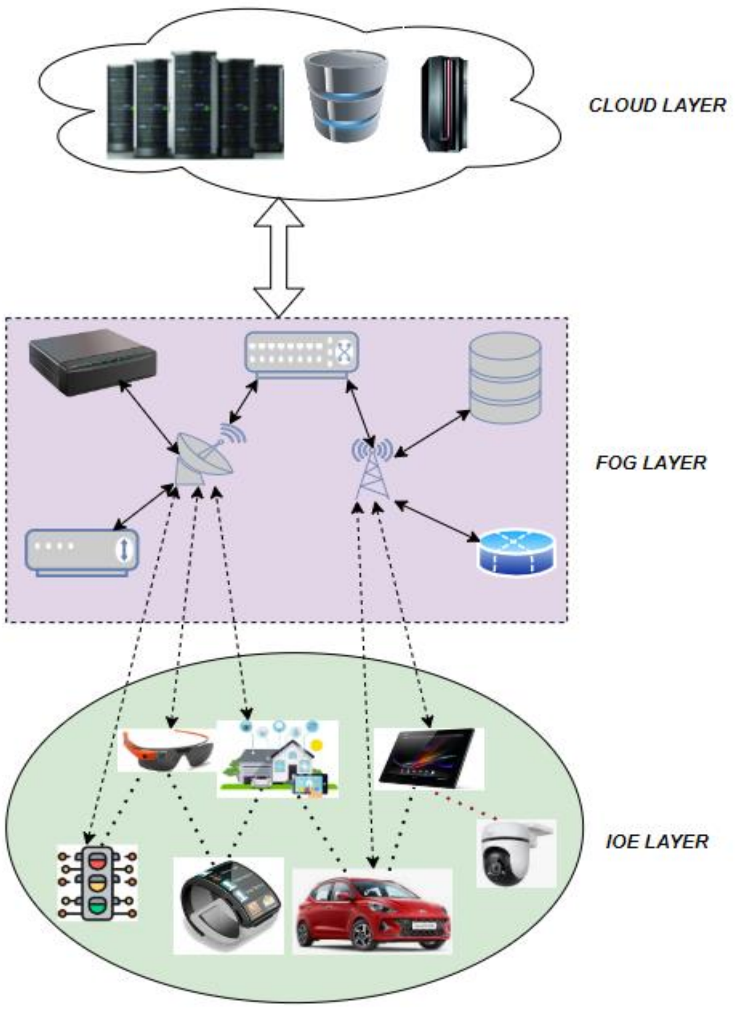


Fig. 1. Fog Architecture

⟷ Fog node to Fog node Link (F2F)

←- - - - - - - - - - - -→ Fog node to Thing Link (F2T)

- - - - - - - - - - - - - - - - - - Thing to Thing Link (T2T)

3.1 IoE Layer

The IoE layer comprises smart devices that are operated by sensors and actuators. Heart rate monitor watches, cameras, smart traffic lights, smart eyewear, GPS sensors in cars, smart home sensors, and other sensors gather unprocessed environmental data, transform it into signals, as well as send it to fog nodes for additional processing. As seen in Figure 1, there are 2 different kinds of communication links allowing devices to communicate with one another in the IoE layer: T2T and F2T linkages. The T2T communication link allows the neighboring IoE devices to communicate with one another via Bluetooth, Zigbee, and WiFi. These Internet of Everything devices can connect to higher-level fog nodes via an F2T communication channel.

3.2 Fog Layer

The fog layer serves as a mediator between the cloud and end devices. It allows for a high degree of interaction and independent evolution between layers by loosely integrating the cloud and IoE layers connected to the fog. The fog layer serves as an intermediary that facilitates communication between devices and the cloud while handling certain processing tasks locally. This is particularly valuable in scenarios where quick decision-making and low-latency responses are critical, like in IoT applications, smart cities, industrial automation, and healthcare. The fog layer efficiently makes usage of the cloud services above even though serving as the backbone of the underlying IoE platform. The fog layer is made up of several fog devices, including routers, switches, cellular base stations, and proxy servers, that have varying degrees of processing, storage, and networking capacity. A cloud server is used to connect these servers. As seen in Figure 1, fog nodes in the fog layer utilize an F2F link to interact with other fog nodes and an F2T link to communicate with IoE devices.

3.3 Cloud Layer

The cloud represents the highest tier of the architecture. It gathers information from networking devices for data analysis and long-term behavior, then sends the findings back to fog devices so that additional actions are taken as needed. The cloud layer in computing signifies a pivotal tier within the overarching architecture of cloud computing. This model revolutionizes the provisioning and access to computing services by leveraging the internet for the on-demand availability of shared resources. Sensors are utilized in FC applications to sense and gather data from the Internet of Everything devices using the Sense-Process-Actuate-Model (SPAM) architecture. After the data has been processed by fog devices, the outcomes are communicated to actuators so that actions can be taken. Four “Peer-to-Peer” (P2P), client-server, or cluster approaches can be used by these applications to facilitate nodal cooperation [33], [34]. The “Distributed Data Flow” (DDF) model is followed in the development of these systems, which are created as Directed Acyclic Graphs with various modules. An input module in a Directed Acyclic Graph receives input, processes it, and then forwards the output to another module.

4.0 CONCEPTION AND EXECUTION

While some fog computing applications are delay-tolerant, others are latency-sensitive. These applications provide dynamic, variable-length workloads that occasionally call for priority execution at both the edge and the cloud. In a heterogeneous environment, applications compete for the limited resources of diverse devices. These jobs are distributed to different fog nodes and carried out there. When using the FCFS technique in conjunction with a basic round-robin (RR) algorithm for job scheduling in FC, all jobs are given equal priority, which causes workloads with short burst times to respond more slowly. On the other hand, the fog computing paradigm seeks to reduce network traffic, waiting times, and response times. As a result, the following goals must be included in the design along with the implementation of the job scheduling algorithm in fog:

i. Optimization of loop latency
ii. Energy efficiency enhancement and resource utilization
iii. Network bandwidth optimization

4.1 Case Study

IoE devices' ubiquitous nature makes it possible to monitor various healthcare system operations. An increasingly significant architectural component for ubiquitous computing is fog computing [35], [36]. Cloud-based solutions in e-healthcare cause longer latency, which is intolerable in emergency situations. By utilizing FC, a large portion of healthcare computing tasks can be accomplished by adjacent fog nodes, with smaller delays and increasing availability [37].

There are various use case types for smart healthcare applications; some are critical, while others can wait [38], [39]. For instance, information on a patient is gathered and stored for a future examination by a doctor. Delays may be tolerated and this kind of data storage as well as retrieval is not very crucial. However, in certain situations- like when a patient is in a severe condition data analysis is needed to produce emergency notifications. These kinds of operations are more latency-sensitive since a delayed response to an emergency signal may endanger the life of the patient.

These are the 3 different application use cases that make up a smart healthcare case study.

i. Critical Incident Response System

The Critical Incident Response System is designed to handle vital patient information, like BP, body temperature, and heart rate, along with blood glucose, from a variety of bodily sensors in an emergency. Important details regarding the patient's condition, such as BP above 150/90 mmHg or blood glucose level above 500mg/dl, are contained in this data and should be processed in an emergency. Subsequently, this data undergoes processing and analysis to produce timely notifications that are necessary to preserve the life of a patient.

ii. Medical Appointment System

The Medical Appointment System is an e-healthcare tool that facilitates the timely scheduling and management of patient appointments. It reduces the likelihood of scheduling a similar time slot for several patients, making it less crucial.

iii. Medical Record Management System

A database that holds patient records is called the Medical Record Management System. The patient's personal data, medical history, appointments, therapy, and test results are all included in this file. Every time a patient arrives at the hospital, the receptionist makes a new record for them to keep in the cloud-based Medical Record Management system.

The following 3 application modules are used to make this smart healthcare case study a fact.

i. DCPE

The DCPE (Diagnostic Criteria and Patient Evaluation) module, which obtains data from all 3 application modules, is placed on a lower-level fog device. It gathers vital information from body sensors, processes it, as well as examines it to produce alerts for emergencies. Additionally, it gets data for the medical record management system and medical appointment system, but it sends it to the facilitator module.

ii. Facilitator

A high-level fog device with a facilitator module is installed that receives data from DCPE. In response to appointment requests, it creates time slots that are allocated to patients. Patients are informed of this schedule of appointments. It sends the medical record of the patient to the medical record database along with some essential data.

iii. Medical Record Database

Cloud storage is used for this module. This Module can get data from the facilitator for analysis and archiving. It creates patterns of patients' medical conditions, hospital visits, and hospital stays, which it then transmits to the facilitator.

The modules of the case study are shown in Fig. 2. Table 1 lists the maximum CPU lengths of tuples for intermodular communication. The CRITICAL tuple must be handled first because it includes the most important information. The suggested algorithm chooses the first work to be executed from this list by ranking all of the arrived jobs according to the needed a million instructions per second (MIPS). The CRITICAL tuple experiences the least amount of delay since the suggested algorithm processes the shortest tuples first. Let's assume that, at a given moment, the tuples CRITICAL(1), ORGANIZE(2), and DOCUMENT RECORD(3) arrive at DCPE in the order listed in Table 2. Table 3 shows the tuples' waiting times for utilizing FCFS that are CRITICAL, ORGANIZE, and DOCUMENT RECORD. It demonstrates that the most critical tuple, CRITICAL, has the least waiting time when utilizing the suggested technique. The ORGANIZE tuple is processed next, and the delay-tolerant DOCUMENT RECORD tuple has the longest waiting time. Table 4 displays the symbols for the suggested algorithm. Suggested steps of job scheduling algorithm are listed in the algorithm.

Table 1. MIPS requirement for every module

TYPE OF TUPLE	LENGTH OF CPU
ORGANIZE	1250
CRITICAL	860
DOCRECDATA	1400
CRIDATA	450
MEDCRI	450
ORGHIS	850
TIME	110
MEDREC	650
DOCUMENT RECORD	3570
CRIHIS	650
ORGDATA	450
MEDORG	450
DOCRECHIS	850
ALERT	120
APPORG	570

Table 2. Arrival sequence of tuples

3	**2**	**1**	**2**	**1**	**3**	**1**	**3**	**2**

Table 3. Tuples waiting time

TUPLE	FCFS	PROPOSED ALGORITHM
DOCUMENT RECORD	6444.4	9480
ORGANIZE	8125	3588
CRITICAL	7464.6	2392

Table 4. Symbols for suggested algorithm

SYMBOL	MEANING
F_L	Finish List of Tuples
W_L	Waiting List of Tuples
T_f	Finished Tuples
T_i	Incoming Tuples
T_{min}	Tuples with the shortest MIPS

According to the suggested approach, the organizer determines how long the waiting list of tuples (W_L) is before a fog node, which has received a tuple T_i from a sensor or another fog node, examines its length. The scheduler allots processing resources to the incoming tuple and it is executed until it is finished if W_L is empty. T_i is added to the final list of tuples F_L and is set to T_f upon completion. T_i is added to the waiting list W_L if it is not empty, and the list is arranged according to the length of the tuples in increasing order. After a tuple is finished, the organizer chooses the shortest tuple T_{min} from the top of the W_L and assigns VM to it until it is completed. T_{min} is added to the completed list of tuples F_L and set to T_f after it is done.

4.2 Complexity Analysis

The suggested algorithm's time complexity is based on the waiting list's (W_L) length or N. The time complexity of the suggested technique is $O(nlogn)$, and the space complexity is $O(n)$ since the scheduler sorts this list.

Fig. 3 demonstrates the block diagram of scheduling. In this, a tuple is submitted to the QoSPTS work scheduler when it reaches a fog device from a sensor node or another fog device. On the fog device, the Scheduler determines if it is the first tuple. In such a case, the received tuple is used. If not, it is added to the queue that is kept up to date independently for each fog device. Every time a tuple is added, the scheduler sorts this queue. Tuples are added to the finished tuple queue when they have finished executing. Subsequently, the scheduler proceeds to choose the shortest tuple from the top of the W_L for execution.

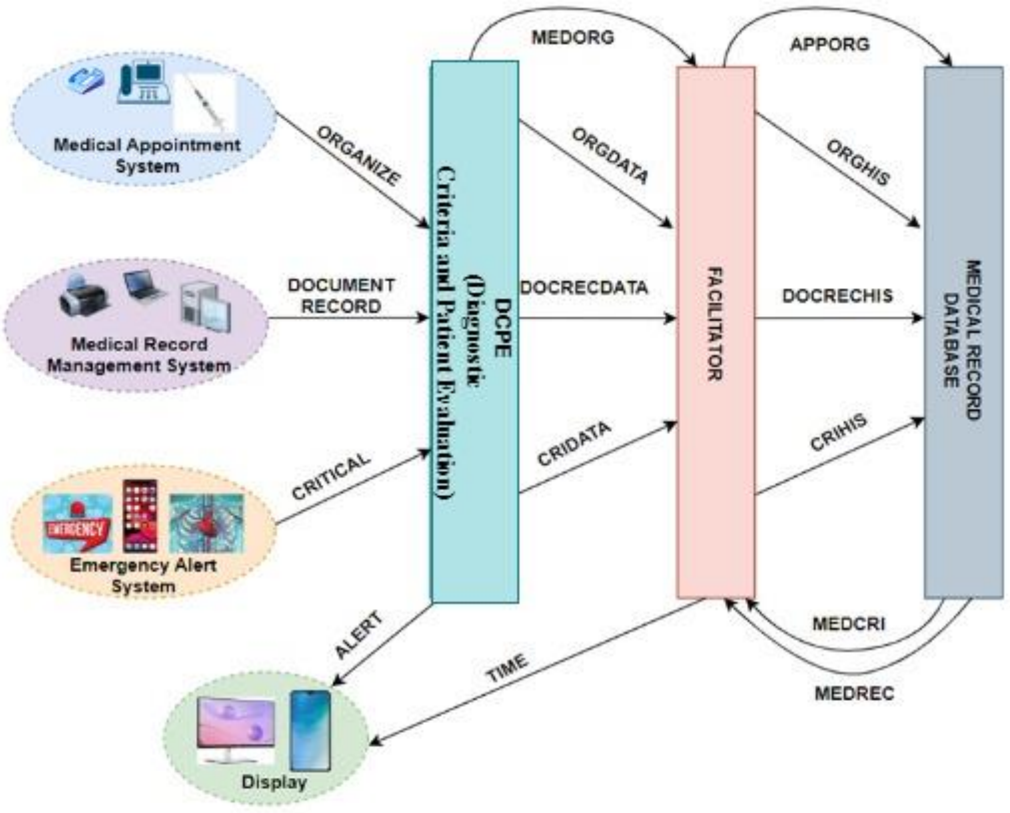


Fig. 2. Case study and its different Modules

Here, utilized the iFogSim toolbox, an improvement over iCloudSim, to develop the suggested approach. Resource management strategies in FC and IoE environments can be effectively simulated with iFogSim [4]. In order to construct the scheduler, new classes to iFogSim are added, and change a few existing ones. A brief overview of the classes in use is provided here.

i. Sensors

IoT sensor simulation can be done with this class. Tuples containing the data from sensor instances can be sent to fog devices. To create tuples with varying sizes, utilize this class.

ii. Fog Device

This class's instances are utilized to model various fog devices. The memory, CPU, storage capacity, and uplink & downlink bandwidths of every fog device are listed. Multiple-level fog nodes are possible. Using the tuples, every fog node can interact with other fog nodes at a higher level as well as devices at the IoE layer. Based on the rising order of MIPS, the scheduler can pick which arriving tuples every fog node can process.

iii. Tuples

All of the entities in fog communicate with one another through tuple class instances. The source, destination, and processing requirements of each tuple are expressed in MIPS.

iv. QoSPTS Scheduler

Quality of Service Priority Tuple Scheduling (QoSPTS) class is an extension of *CloudletScheduler*, which is responsible for keeping two lists: the finished list (F_L) and the waiting list (W_L). All of the tuples on the waiting list are awaiting execution, and all of the tuples on the finish list have finished their execution. The smallest tuple is chosen from the W_L when a tuple is finished.

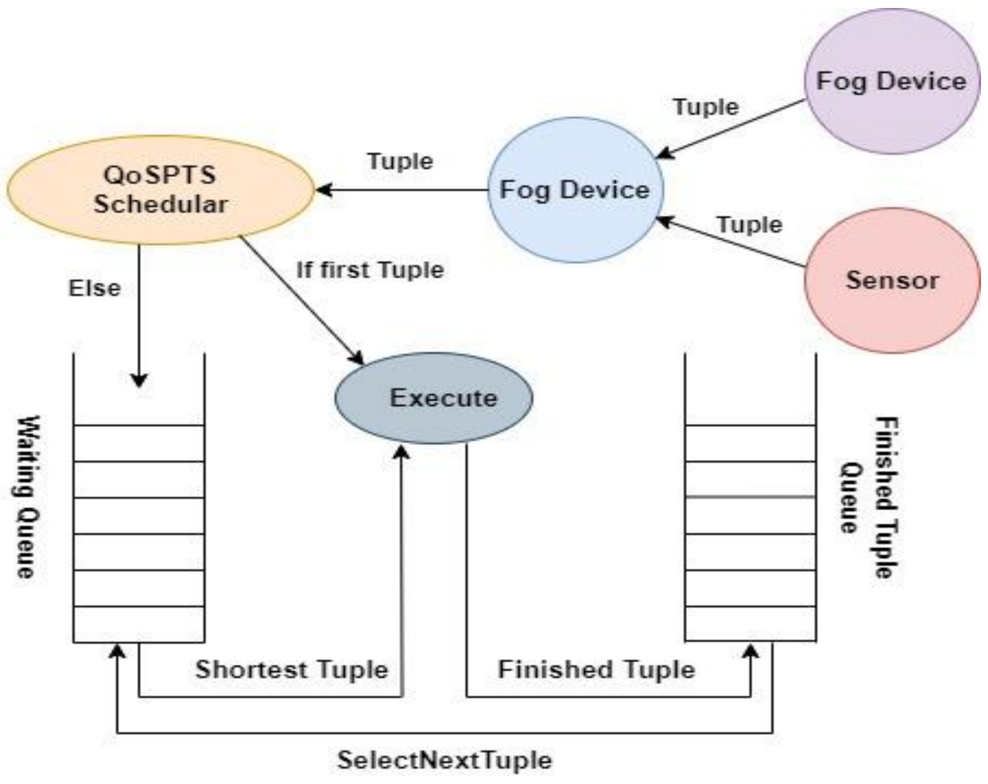


Fig. 3. Scheduling Block Diagram

Figure 4 illustrates the flow diagram for the QoSPTS (Quality of Service Priority Task Scheduling) algorithm. The process begins with the input of multiple tasks. Every task is assigned to a VM (Virtual Machine). Subsequently, the scheduler examines the waiting list. If the current tuple is the first one, it is executed immediately. If not, the tuple is added to the waiting queue. In addition to the waiting queue, the QoSPTS algorithm sorts the queue based on task priority. The task with the minimum value (indicating the highest priority) is selected for execution first. After execution, the task is moved to the finished list. Finally, the finished list is displayed, concluding the scheduling process.

Figure 5 illustrates how a tuple sent from a sensor using the Transmit() method is sent via the Send(tuple) technique to a low-level fog device that is connected. When a tuple arrives, the fog device calls a callback function called processTupleArrival(). This technique determines if the tuple must be processed at the fog device or if it should be forwarded to a higher-level fog device. The SubmitTuple() method is used to send the tuple to the QoSPTS Scheduler if it is to be processed by a fog device. The tuple is added to the waiting list (W_L) and sorted according to its length using this method. The smallest tuple is then chosen by SchedulenextTuple(), and the scheduled tuple is then sent to the fog device for execution. The CloudletFinish() method asks the scheduler to run the next tuple after the current one has finished. Using this method, the completed tuple is added to the finished tuple list F_L. The scheduler then chooses the shortest tuple from the head of the waiting list W_L.

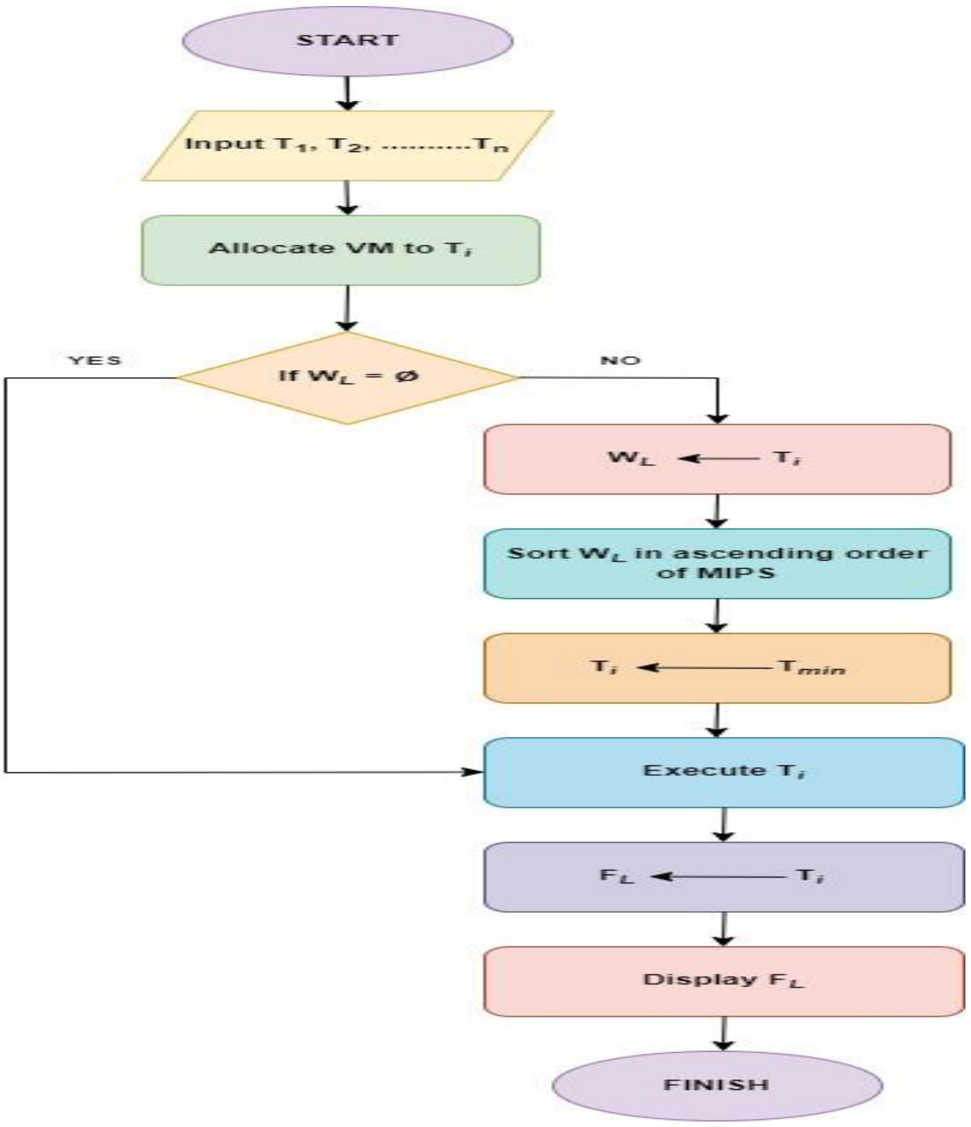


Fig. 4. Flow diagram for QoSPTS

Algorithm: QoSPTS Algorithm

```
INPUT: Tuple List (T1, T2, ……………….., TN)
OUTPUT: Finished Tuple $T_f$
    do
        if new Tuple arrived then
            $T_i \Leftarrow$ received tuple
            if ($W_L$) = Ø then
                Allocate VM to $T_i$
                Execute $T_i$
                $T_f \Leftarrow T_i$
                $F_L \Leftarrow T_f$
            else
                $W_L \Leftarrow T_i$
                Sort ($W_L$) in ascending order of MIPS
            end if
        end if
        if $T_f$ then
            $T_{min} \Leftarrow$ Select $T_i$ with minimum MIPS from $W_L$ return Allocate VM to $T_{min}$
            Execute $T_{min}$
            $T_f \Leftarrow T_{min}$
```

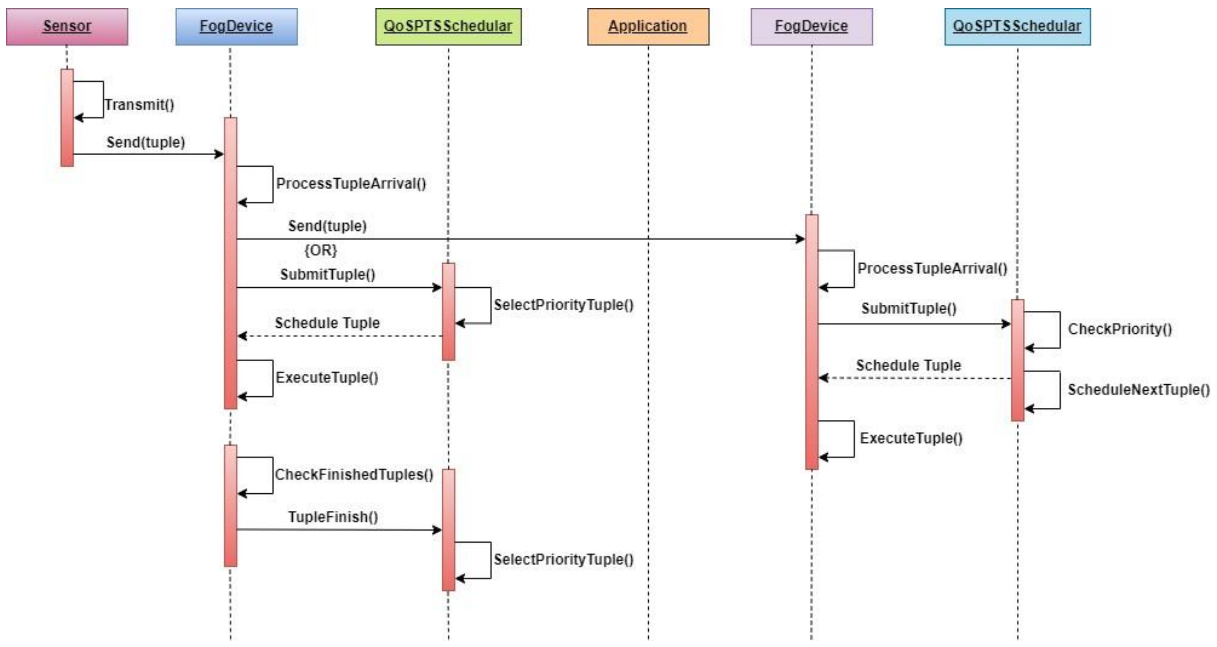


Fig. 5. Sequence diagram of tuple scheduling in Fog computing paradigm

5.0 EVALUATION AND VALIDATION OF PERFORMANCE OUTCOME

The current section displays the findings of the suggested QoSPTS algorithm in several settings compared to the most advanced FCFS algorithm. Here, three metrics are chosen: mean loop latency, energy utilization, and network utilization. Table 5 displays the notations utilized in the different equations.

Table 5. Simulation Setup Notations

SYMBOL	MEANING
ET_i	Execution End Time of Tuple
ST_i	Execution Start Time of Tuple
T_C	Current Time
$\mathcal{E}_C$	Current Energy Consumption
P_H	Host Power in Last Utilization
T_i	Update Time of Last Utilization
Ni	Network Size of ith Tuple
Li	Latency of ith Tuple

5.1 Simulation Environment Setup

Suggested QoSPTS scheduler algorithm is applied and assessed via a simulated fog environment with the iFogSim toolkit [4]. QoSPTS scheduler's performance is validated through a case study named Smart Healthcare, which is described in Section 4.1, to demonstrate the algorithm. There are delay-tolerant and latency-sensitive tuples in this case study. An Intel(R) Core(TM) i5-3210M CPU@2.50 GHz computer with 4 GB RAM and a 64-bit Microsoft Windows 10 operating system was used for the entire simulation process.

Three modules- Critical Incident Response System, Medical Appointment System, and Medical Record Management System- are used to create tuples of varying lengths. The sensors produce CRITICAL tuples, which are made up of patient's condition data that needs to be processed, examined, and handled as soon as possible. It is less important because the Medical Appointment System creates an ORGANIZE tuple with an appointment request as well as the appointment time shown after processing. The DOCUMENT RECORD tuple from the medical record management system is delay-tolerant since it contains patient data that will be saved on the cloud later. Here, require the quickest alert because CRITICAL tuples are the most important. Calculating latency for end-to-end modules since ORGANIZE tuples are less important and DOCUMENT RECORD tuples are latency tolerant.

i. Critical Alert Loop: DCPE->Facilitator->display.
ii. Medical Appointment Loop: DCPE-> Facilitator ->DCPE->display.
iii. Medical Record Management Loop: DCPE-> Facilitator -> Medical Record Database.
iv. History: Facilitator-> Medical Record Database-> Facilitator->DCPE.

Since loop number one is a critical condition, it is the least delay-tolerant. As such, it is the most important. Since loop number four requires the patient data to be kept on the cloud for long-term analysis, it can tolerate delays better than loops number two and three, which deal with appointment scheduling.

5.2 Parameters Specification

A detailed description of various devices used in the smart healthcare case study is presented in Table 6. During the simulation process, five parameter specifications are taken into consideration, using 5 high-level fog devices at level 2 (as facilitators). By changing the number of fog nodes (acting as DCPE) at level 1, a thorough evaluation is carried out. Three different sorts of tuples- Medical Appointment, Critical Alert, and Medical Record Management- are arriving at level 1 fog device. Each of the fog devices is connected with 20, 40, 60, 80, & 100 fog nodes to it at this level, correspondingly. Thus, every specification Spec 1, Spec 2, Spec 3, Spec 4, & Spec 5 have 100, 200, 300, 400, & 500 number of fog nodes. Additionally, the application modules in the smart healthcare case study are set up with RAM, MIPS, bandwidth, module size, and 10 B, 100 B, 10000 B, & 100 B/S, correspondingly.

Table 6. Device Configuration in Smart Healthcare case study and its Parameters

NAME	CPU (MIPS)	RAM (MB)	Uplink Bw (MB)	Downlink Bw (MB)	Level	Rate per MIPS	Busy Power (W)	Idle Power (W)
Cloud	44 800	40 000	100	10 000	0	0.01	1738	1242
Proxy Server	7500	4000	10 000	10 000	1	0.00	109.342	81.4303
2nd level Fog Node	5200	4000	10 000	10 000	2	0.00	109.342	81.4303
1st level Fog Node	1100	1200	10 000	1000	3	0.00	89.63	80.34

5.3 Mean Loop Latency

To determine the latency of each module in the loop, employing a control loop. Here, calculates the average CPU time $\overline{T_{\text{CPU}}}$ used by all tuples of a specific type of tuple in order to calculate the loop latency. Here, equation (1) to calculate the mean loop latency.

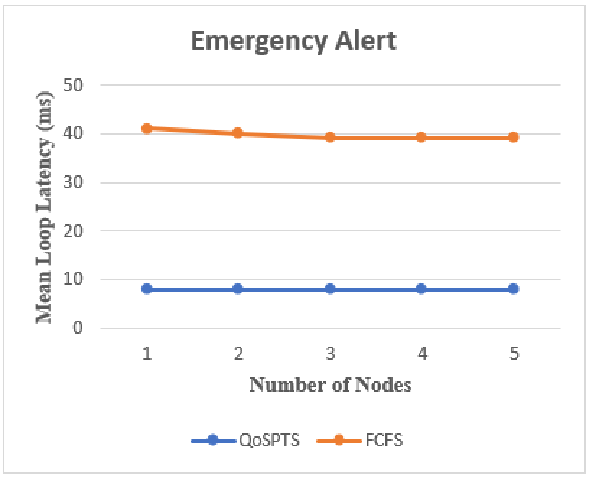
$$\overline{T_{\text{CPU}}} = \begin{cases} \dfrac{ST_i \times N + ET_i - ST_i}{N+1} & \text{If the mean CPU time for a specific kind of tuple has previously been calculated} \\ ET_i - ST_i & \text{Otherwise} \end{cases} \quad (1)$$

where N is denoted as the total number of performed tuples of a specific type, ET_i is the i^{th} tuple's ending execution time, and ST_i is indicated as the starting execution time. The equation is used to calculate each tuple's latency in equation (2).

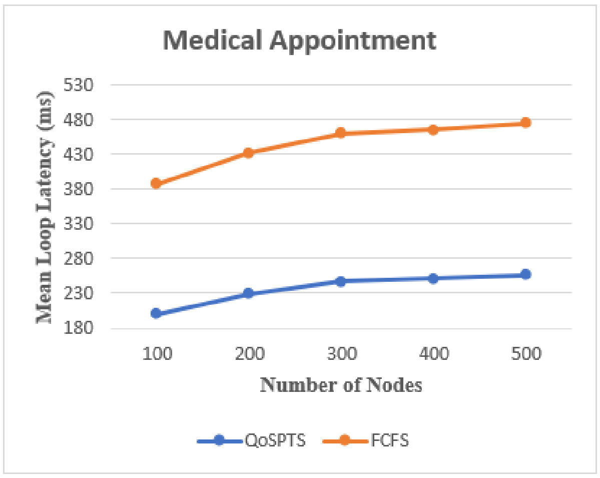
$$Latency = ST_i - ET_i, \pounds \in Ʈ \quad (2)$$

where tuple set Ʈ is denoted as the current set. Since FCFS is the sole scheduling method that has been implemented in iFogSim, comparing the mean loop latency that the suggested QoSPTS scheduler produces with the outcomes of that algorithm. The mean loop latency, calculated in milliseconds with FCFS and the suggested multiple scheduling strategies, is displayed in Figure 6. The number of nodes is displayed along the x-axis, and the mean application loop delay for 450 simulation times is displayed along the y-axis.

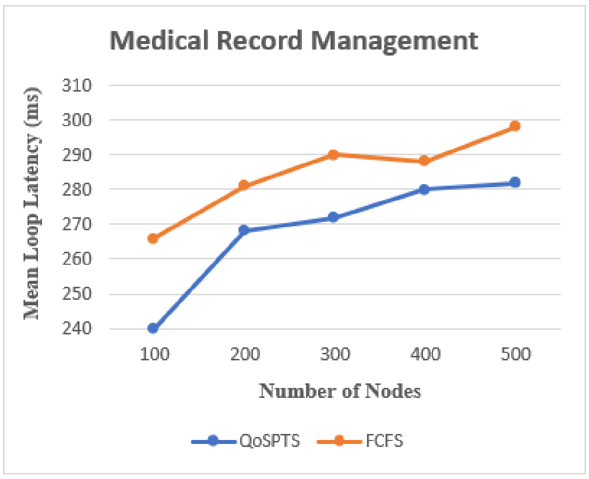
The mean loop latency for the most important loop-the emergency alert-is displayed in Figure 6A. Here, the QoSPTS waits are noticeably shorter than the FCFS delays. The delay doesn't increase as the number of nodes increases when using QoSPTS. The mean loop delay for a medical appointment, which is not too severe, is shown in Figure 6B. In this case, FCFS outperforms QoSPTS since QoSPTS prioritizes the most important loop more. The mean loop latency for the Medical Record Management is displayed in Figure 6C, where the suggested approach once again produces latencies that are lower than those of FCFS. When there are more than 450 nodes in an FCFS network, the latency increases significantly. The latency for the history of various number of IoE nodes that generates the medical history of the patient is displayed in Figure 6D, where the mean loop latency utilizing QoSPTS is less than that of FCFS. Utilizing QoSPTS reduces the delay after 450 nodes.



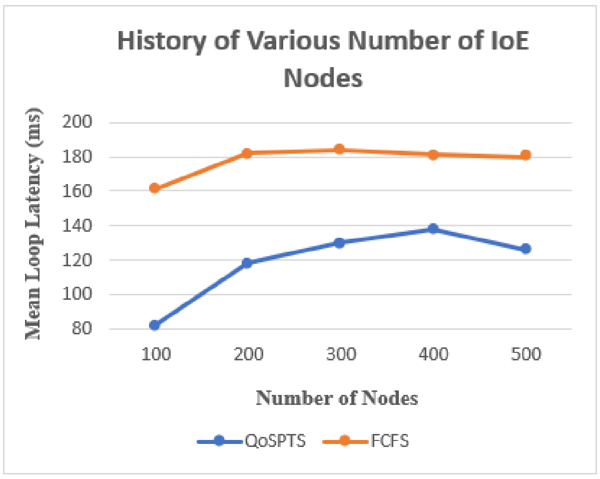
(A)



(B)



(C)



(D)

Fig. 6. Mean loop latency in milliseconds (ms): (A) Emergency alert, (B) Medical appointment, (C) Medical record management, (D) History of various number of IoE nodes

5.4 Energy Utilization

Here, calculate the energy utilization by a Fog device, $\mathcal{E}_{FN}$, using equation (3)

$$\mathcal{E}_{FN} = \mathcal{E}_C + (T_C - T_i) \times \mathrm{P}_H \qquad (3)$$

Any fog device's energy can be calculated by utilizing the power of every host for a predetermined execution period, where P_H is denoted as the variables are the host power during the last utilization, T_C is denoted as the current time, T_i is denoted as the update time of the previous use, and $\mathcal{E}_C$ is denoted as the current energy consumption.

This section details the cost of using the QoSPTS method on a cloud system in comparison to the FCFS algorithm. The equation (4) can be used to compute the total execution cost.

$$Cost = \sum_{i=1}^{FN} \left(C_C + (T_C - T_i) \times R_{mips} \times v_i \times M_{host} \right) \tag{4}$$

Equation (4) takes into account the following variables: C_C is denoted as the current cost, current time is shown by T_C , last utilization update time is shown by T_i , rate per millisecond is shown by R_{mips}, v_i is denoted as the last utilization, and M_{host} is denoted as the total milliseconds of all hosts. The last use can be calculated as, $v_i = min(1, AM_{host} / M_{host})$ where AM_{host} is the host's total allotted mips.

The average utilized energy used by the cloud and fog nodes is displayed in Figure 7. The number of nodes is indicated along the x-axis, while the amount of energy used is represented along the y-axis.

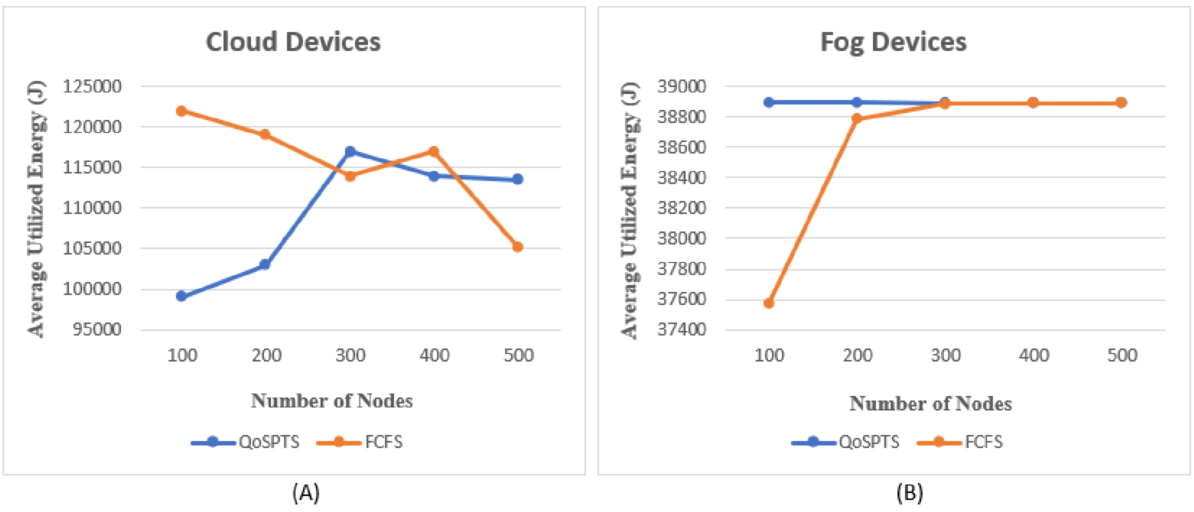


Fig. 7. Average utilized energy in joule (J): (A) Cloud devices, (B) Fog devices

In particular, figure 5 shows the average utilized energy of the cloud component in the architecture. There are two observations in this figure. First, for a limited number of participating IoE nodes (<300 nodes), the average energy utilized by the QoSPTS algorithm is lower than that of FCFS shown in Figure 7A. Second, when the problem gets bigger, this value gets bigger as well. As a result, employing FN (Energy utilized by the device) to lower the energy usage of the server utilized to process the request and give adaptive load balancing for the sent workload. Consequently, figure 7B illustrates, the average energy utilized by FNs while the suggested technique is lower than when employing the FCFS algorithm. It is interesting to note that this figure demonstrates how FNs reduce the average server energy usage to roughly 60% less than when the cloud system is used alone. Additionally, the QoSPTS algorithm can act independently on large networks, with a value that is comparable to FCFS.

5.5 Network Utilization

Network utilization use is the third evaluation parameter. As the number of devices rises, so does the network consumption, which causes congestion. The application operating on the cloud network performs poorly as a result of this congestion. By dividing the load among intermediary fog devices, FC contributes to a decrease in network congestion. The equation (5) is utilized to compute the network utilization.

$$Nuse = \sum_{i=1}^{N} Li \times Ni \tag{5}$$

Where Ni is denoted as size of the network of the i^{th} tuple, L_i is indicated as the latency, and N is denoted as the total number of tuples.

Figure 8 compares QoSPTS and FCFS algorithms' network utilization. Number of nodes shown by x-axis and average network usage is shown by y-axis over 450 simulations. In comparison with the FCFS algorithm, the results show that the technique can save roughly 30% of the network when the node count is less than 300. However, as the node count rises, the network saving drops to roughly 20%.

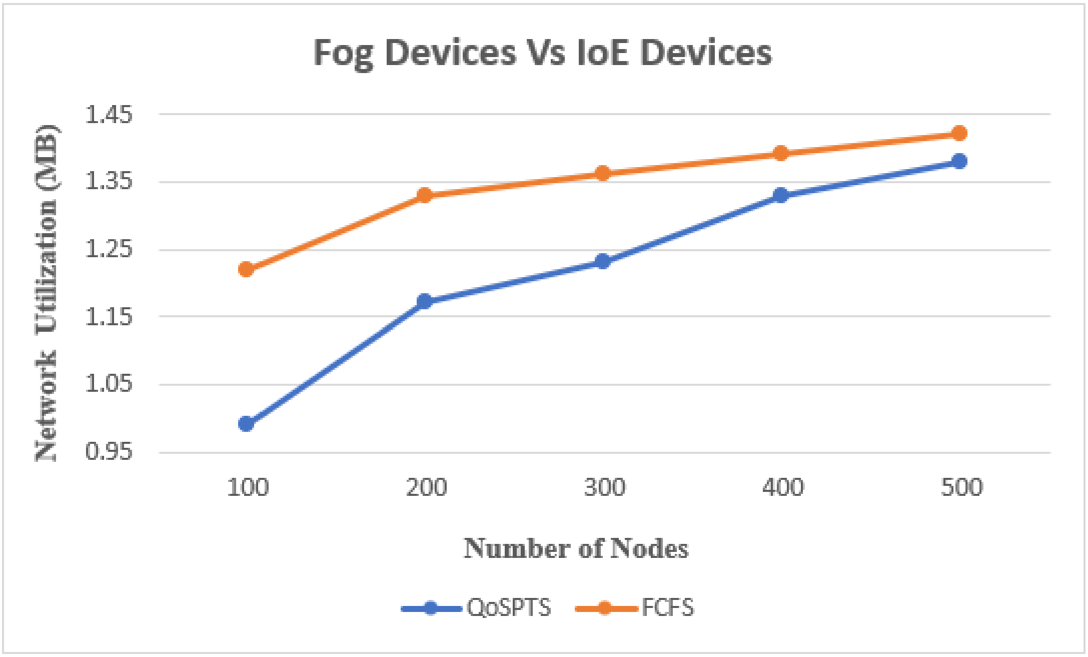


Fig. 8. Fog devices Network utilization in MB (Megabytes) vs. different number of nodes (IoE devices)

The outcomes show that, in comparison with the FCFS, the suggested scheduling method works better in terms of latency as well as network utilization. However, when the number of fog nodes is smaller, the average energy utilization of the proposed technique is slightly greater than that of FCFS.

6.0 CONCLUDING POINTS AND FUTURE PROSPECTS

Due to more and more IoE devices producing massive amounts of data, cloud computing cannot keep up with the demands of real-time applications, which include mobility support, low latency, and location awareness. To get around these restrictions, a new computing model known as FC has surfaced. It works in conjunction with cloud computing to enable analytics, real-time processing, as well as storage capabilities close to the edge device. Fog computing presents a significant difficulty in terms of job scheduling because the edge devices have limited resources.

In this research, an effective job scheduling strategy to minimize latency for delay-sensitive applications is designed and implemented. Presented an example case study in the healthcare sector to highlight the requirements that IoE devices must meet in terms of both delay-sensitivity and delay-tolerates. The suggested approach optimizes energy and network consumption, lowers loop latency, and arranges jobs on fog devices according to their duration.

The suggested QoSPTS technique can starve tuples with longer lengths even though it lowers the average waiting time. Here, plan to use scheduling techniques such as meta-heuristics, hyper-heuristics, reinforcement learning-based approaches, etc. in the future to address this problem. For the purpose of capacity planning these resource-constrained devices, also aim to employ an analytical model.

REFERENCES

[1] E. Baccarelli, P. Vinueza-Naranjo, M. Shojafar, M. Scarpiniti, and J. Abawajy, “Fog of Everything: Energy-Efficient Networked Computing Architectures, Research Challenges, and a Case Study,” *IEEE Access*, Vol. 5, No. 2, 2017, pp. 9882-9910.

[2] F. Hussain and A. Alkarkhi, “Big Data and Fog Computing,”*Internet of Things* , pp. 27–44.

[3] M. Anwar, S. Wang, M. Zia, A. Jadoon, U. Akram, and S. S. Raza Naqvi, "Fog Computing: An Overview of Big IoT Data Analytics," *Wirel. Commun. Mob. Comput.*, Vol. 7, No. 1, pp. 1–22, 2018.

[4] H. Gupta, A. Dastjerdi, S. Ghosh, and R. Buyya, "iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments," *Softw. Pract. Exp.*, Vol. 47, No. 3, pp. 285-310, 2016.

[5] J. He, S. Member, J. Wei, K. Chen, and Z. Tang, "Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities," *IEEE Internet Things J.*, Vol. 5, No. 2, pp. 677–686, 2018.

[6] M. Aazam, S. Zeadally, and K. Harras, "Fog Computing Architecture, Evaluation, and Future Research Directions," *IEEE Commun. Mag.*, Vol. 56, No. 2, pp. 46–52, 2018.

[7] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog Computing: Principles, Architectures, and Applications." *Internet of Things*, pp. 61-75, 2016.

[8] M. Shojafar, N. Cordeschi, and E. Baccarelli, "Energy-Efficient Adaptive Resource Management for Real-Time Vehicular Cloud Services," *IEEE Trans. Cloud Comput.*, Vol. 7, No. 2, pp. 196–209, 2019.

[9] S. Mishra Tiwari and S. Jain, "Ontologies as a semantic model in IoT," *Int. J. Comput. Appl.*, Vol. 42, No. 4, pp. 301–311, 2018.

[10] L. Liu, D. Qi, N. Zhou, and Y. Wu, "A Task Scheduling Algorithm Based on Classification Mining in Fog Computing Environment," *Wirel. Commun. Mob. Comput.*, Vol. 44, No. 1, pp. 421–435, 2018.

[11] S. Mishra Tiwari, R. Sagban, A. Yousif, and N. Gandhi, "Swarm intelligence in anomaly detection systems: an overview," *Int. J. Comput. Appl.*, Vol. 43, No. 3, pp. 1–10, 2018.

[12] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. Rana, and M. Parashar, "Mobility-Aware Application Scheduling in Fog Computing," *IEEE Cloud Comput.*, Vol. 4, No. 2, pp. 26–35, 2017.

[13] M. S. Ul Islam, A. Kumar, and Y.-C. Hu, "Context-aware scheduling in Fog computing: A survey, taxonomy, challenges and future directions," *J. Netw. Comput. Appl.*, Vol. 12, No. 2, pp. 103008-103021, 2021.

[14] N. Khattar, J. Sidhu, and J. Singh, "Toward energy-efficient cloud computing: a survey of dynamic power management and heuristics-based optimization techniques," *J. Supercomput.*, Vol. 75, No. 2, pp. 2013-2029, 2019.

[15] I. Seth, S. N. Panda, and K. Guleria, "The Essence of Smart Computing: Internet of Things, Architecture, Protocols, and Challenges," *9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 1–6, 2021.

[16] I. Seth, S. N. Panda, and K. Guleria, "IoT based smart applications and recent research trends," *6th International Conference on Signal Processing, Computing and Control (ISPCC)*, pp. 407–412, 2021.

[17] I. Seth, K. Guleria, and S. N. Panda, "Introducing intelligence in vehicular ad hoc networks using machine learning algorithms," *ECS Trans.*, Vol. 107, No. 1, pp. 8395-8410, 2022.

[18] M. Rani, K. Guleria, and S. N. Panda, "Blockchain technology novel prospective for cloud security," *10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pp. 1-6, 2022.

[19] X.-Q. Pham and E.-N. Huh, "Towards task scheduling in a cloud-fog computing system," *18th Asia-Pacific network operations and management symposium (APNOMS)*, pp. 1–4, 2016.

[20] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System," *IEEE Trans. Comput.*, Vol. 65, No. 4, pp. 10231-10252, 2016.

[21] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu, "Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets," *IEEE Internet Things J.*, Vol. 12, No. 2, pp. 132-145, 2017.

[22] Z. Pooranian, M. Shojafar, P. G. V. Naranjo, L. Chiaraviglio, and M. Conti, "A Novel Distributed Fog-Based Networked Architecture to Preserve Energy in Fog Data Centers," *IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 604–609, 2017.

[23] D. Hoang and D. Dang, "FBRC: Optimization of task Scheduling in Fog-Based Region and Cloud," *2017 IEEE Trustcom/BigDataSE/ICESS*, Sydney, NSW, Australia, pp. 1109-1114, 2017.

[24] Y. Sun, F. Lin, and H. Xu, "Multi-objective Optimization of Resource Scheduling in Fog Computing Using an Improved NSGA-II," *Wirel. Pers. Commun.*, Vol. 102, No. 2, pp. 3210-3223, 2018.

[25] T. Choudhari, M. Moh, and T.-S. Moh, "Prioritized task scheduling in fog computing," *In Proceedings of the ACMSE 2018 Conference*, pp. 1-8. 2018.

[26] S. Bitam, S. Zeadally, and A. Mellouk, "Fog computing job scheduling optimization based on bees swarm," *Enterp. Inf. Syst.*, Vol. 12, No. 4, pp. 373–397, 2018.

[27] K. Gai and M. Qiu, "Optimal resource allocation using reinforcement learning for IoT content-centric services," *Appl. Soft Comput.*, Vol. 70, No. 2, pp. 12–21, 2018.

[28] Q. Zhang, M. Lin, L. Yang, Z. Chen, S. Khan, and P. Li, "A Double Deep Q-Learning Model for Energy-Efficient Edge Scheduling," *IEEE Trans. Serv. Comput.*, Vol. 12, No.2, pp. 212-227, 2018.

[29] B. M. Nguyen, H. Binh, T. Anh, and S. Do, “Evolutionary Algorithms to Optimize Task Scheduling Problem for the IoT Based Bag-of-Tasks Application in Cloud–Fog Computing Environment,” *Appl. Sci.*, Vol. 9, No.3, pp. 1730-1747, 2019.

[30] D. Rahbari and M. NICKRAY, “Low-latency and energy-efficient scheduling in fog-based IoT applications,” *TURKISH J. Electr. Eng. Comput. Sci.*, Vol. 27, No. 3, pp. 1406–1427, 2019.

[31] L. Mai, N.-N. Dao, and M. Park, “Real-Time Task Assignment Approach Leveraging Reinforcement Learning with Evolution Strategies for Long-Term Latency Minimization in Fog Computing,” *Sensors*, Vol. 18, No. 3, pp. 2830-2847, 2018.

[32] M. Rani, K. Guleria, and S. N. Panda, “Unleashing the Power of QoS: A Comprehensive Study and Evaluation of Services-based Scheduling Techniques for Fog Computing,” *Int. J. Intell. Syst. Appl. Eng.*, Vol. 12, No. 4s, pp. 388–405, 2023.

[33] M. Mahmud and R. Buyya, “Fog Computing: A Taxonomy, Survey and Future Directions,” *Internet of Everything - Algorithms, Methodologies, Technologies and Perspectives*, pp. 21-37, 2016.

[34] M. Rani, K. Guleria, and S. N. Panda, “Cloud Computing An Empowering Technology: Architecture, Applications and Challenges,” *9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pp. 1–6, 2021.

[35] A. M. Rahmani *et al.*, “Exploiting Smart E-Health Gateways at the Edge of Healthcare Internet-of-Things: A Fog Computing Approach,” *Futur. Gener. Comput. Syst.*, Vol. 78, No. 4, pp. 5612-5629, 2017.

[36] M. Rani, K. Guleria, and S. N. Panda, “Enhancing Performance of Cloud: Fog Computing Architecture, Challenges and Open Issues,” *9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pp. 1-7, 2021.

[37] K.-M. Chao and J.-Y. Chung, “Edge and fog services,” *Serv. Oriented Comput. Appl.*, Vol. 13, No. 2, 2019.

[38] F. Kraemer, A. Bråten, N. Tamkittikhun, and D. Palma, “Fog Computing in Healthcare – A Review and Discussion,” *IEEE Access*, Vol. 21, No. 2, pp. 9206–9222, 2017.

[39] M. Rani, K. Guleria, and S. N. Panda, “State-of-the-Art Dynamic Load Balancing Algorithms for Cloud Computing,” *ECS Trans.*, Vol. 107, No. 1, pp. 8339-8347, 2022.