STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **13**, January 2025, pp 309–330.
Published online in International Academic Press (www.IAPress.org)

Enhancing Performance and Latency Optimization in Fog Computing with a Smart Job Scheduling Approach

Meena Rani, Kalpna Guleria*, Surya Narayan Panda

Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India

Abstract Nowadays, Internet of Everything (IoE) devices are growing rapidly, producing vast amounts of data. Cloud computing offers processing, analysis, and storage solutions to manage these large data volumes. However, the rising latency and bandwidth usage could be more suitable for real-time applications, including intelligent healthcare devices, online gaming, and surveillance via video. In order to tackle the rise in latency and bandwidth utilization in cloud computing technology, Fog Computing (FC) has been developed as it offers networking, processing, storage, and analytics functions. Since FC is still in its early stage, scheduling jobs and allocating resources are two significant issues. With the help of this innovation, there are resource limitations on the fog devices at the network's edge. Consequently, scheduling is crucial for choosing a fog node for a job assignment. An intelligent and effective work scheduling algorithm can decrease energy usage and application request response time. This research introduces an innovative Quality of Service Priority Tuple Scheduling (QoSPTS) scheduler that optimizes network capacity and latency while enabling service for the IoE. This case study demonstrates the effective management of IoE device requirements by efficiently allocating resources across fog devices and optimizing scheduling to enhance quality of life. The study uses iFogSim to compare the proposed scheduling algorithm with other methods by considering energy efficiency, network utilization, cost, and latency as performance measures. Results showed that the proposed Scheduler's latency network bandwidth, energy utilization and cost are highly enhanced compared to traditional approaches such as FCFS, Concurrent, Delay-Priority, and QoS-Aware.

Keywords Fog computing, energy efficiency, resource allocation, latency optimization, resource use efficiency, QoSPTS, job scheduling

AMS 2010 subject classifications 68U35

DOI: 10.19139/soic-2310-5070-2141

1. Introduction

The IoT (Internet of Things) includes a vast ecosystem of everyday objects interconnected through heterogeneous networks including vehicles, farms, factories, computers, appliances, and animals. Utilizing networks and algorithms, IoT allows these objects to perform valuable tasks autonomously without 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. By means of enabling communications between machines and people (M2P), machines and machines (M2M), and people and people (P2P), the IoE model aids in the automation of human tasks. IoE systems that use cloud-centric IoT (CIoT) architecture for processing, analytics, and storage include intelligent traffic, smart farming, and smart healthcare monitoring [2, 3]. CIoT data centers typically experience long delays since they are generally located many steps away from a source node. By 2020, the world's connected device count is expected to increase from 5.4 billion in 2019 to 1 trillion by 2025 [4]. The creation of vast amounts of data

*Correspondence to: Kalpna Guleria (Email: guleria.kalpna@gmail.com). Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India

ISSN 2310-5070 (online) ISSN 2311-004X (print)

will exceed the cloud's capacity, leading to extended latency 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 FC, edge computing and mobile computing etc., were implemented to overcome these CIoT restrictions by offering networking, storage, processing, and decision-making capabilities [6]. Fog computing has received much attention lately compared to all the suggested strategies. FC is a collaborative computing model that aligns with cloud computing by enabling computation, networking, and decision-making and generally utilized Sense Process Actuate Model (SPAM). Sensors in SPAM detect and gather information, which is sent to fog devices for analysis before actuators are tasked with taking appropriate action. Some 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 FC 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 prompt patient status notification, an intelligent healthcare system is necessary to preserve their life [9]. An effective job scheduling algorithm must be developed to optimize the use of these diverse and limited resource fog devices [10]. The ultimate goal is to reduce reaction times and network utilization while maintaining energy efficiency [11]. While several proposed work scheduling methods have been proposed, such as FCFS, Round Robin, delay-priority, Concurrent, QoS-Aware etc., FC job scheduling is still in its early stages [12]. Current algorithms, such as Round Robin and FCFS, execute tasks in their arrival order, making them ineffective at reducing response times for latency-sensitive applications. A scheduling method which reduces average response time and network usage while maintaining optimal energy consumption must be developed for applications sensitive to latency. A thorough analysis of the optimization of loop latency and performance enhancement in an FC environment is discussed. The goal is to offer a thorough and brief summary of the most recent related literature [13, 14, 15, 16, 17, 18].

1.1. *Objectives and Contributions*

Fog computing, as an extension of cloud computing, addresses the need for low-latency processing by bringing computational resources closer to the data source, often at the network's edge. This paradigm introduces several challenges for scheduling algorithms, primarily due to the heterogeneous nature of fog nodes, which vary significantly in terms of computational capacity, storage, and energy resources. Moreover, fog environments are resource-constrained, necessitating algorithms that prioritize efficient resource allocation and energy conservation. Additionally, many applications in fog computing are latency-sensitive, requiring scheduling methods that minimize delays to meet real-time processing demands. The dynamic topology of fog networks, characterized by the mobility of nodes and varying connectivity, further complicates scheduling, as algorithms must adapt to these changes in real-time. Scalability is also a critical concern, given the exponential growth in connected IoT devices, which necessitates scheduling solutions that can efficiently manage large-scale, distributed environments. Finally, security and privacy concerns in fog computing add another layer of complexity, requiring that scheduling algorithms not only optimize performance but also ensure the protection of sensitive data. Our proposed QoSPTS algorithm is designed to address these challenges by optimizing network capacity, reducing latency, and efficiently allocating resources across diverse fog nodes. The present research offers a novel multi-objective fog scheduler that facilitates IoE service provisioning for applications sensitive to latency. The goal is to achieve lower latency and energy usage while maximizing the usage of fog devices at network's edge. The following are the main contributions of the suggested work:

1. Optimization of Loop Latency: Reducing service response time is the most crucial factor in latency-sensitive applications. The job on the fog nodes are carried out using the suggested algorithm based on their duration and the loop latency is optimized first by carrying out the minor job.

2. Network Bandwidth Optimization: Network utilization rises with the total IoE devices linked to every application, resulting in network congestion. The effort also aims to optimize network bandwidth to enhance the application's performance.
3. Energy Efficiency Enhancement and Resource Utilization: The suggested algorithm aims to improve energy efficiency and resource usage for resource-constrained fog devices that uses energy like all other network devices.

1.2. Organization of the Paper

The sections of this research are organized as follows: Section II reviews related research work, Section III outlines the design framework, Section IV details the development and implementation of the proposed method, Section V evaluates and validates the experimental performance results, and Section VI summarizes the findings, discusses ongoing projects, and explores future opportunities.

2. Related Work

Pham et al. [19] proposed a dependent job allocation technique for fog nodes. Dependent jobs necessitate data exchange and are represented by workflows or Directed Acyclic Graphs (DAGs), with tasks prioritized by traversing the DAG and assigned to fog nodes. If fog nodes are resource-constrained, the job is sent to the cloud for completion. Another work image placement and scheduling approach in FC was presented by Zeng et al. [20]. The job file is kept on a server, and fog nodes and embedded clients work together to leverage storage servers for computations. Tasks are scheduled to ensure minimal completion times and optimize the experience of user. Ni et al. [21] introduced a vibrant 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, describing a resource allocation algorithm based on heuristics to maximize energy utilization. The resource allocation is viewed as an issue that is cognizant of the bin packing penalty. Hoang et al. [23] created a different heuristic-based job scheduling system to improve performance and minimize latency. It is advised to assign tasks to clouds and fog regions using a fog-based region design. 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. The improved Non-Dominated Sorted Genetic Algorithm II theory is utilized for organizing resources across fog nodes for optimizing multiple objectives in the same cluster and achieved more stable work performance and short delays. Liu et al. [10] extracted association rules using a priori technique and scheduled a task to a fog device by combining the produced rules with the task's least completion time. Shortening average wait and execution times is asserted. FC supports a wide range of applications, both delay-tolerant and latency-critical, with reducing loop delay being crucial for the latter. Another method for scheduling jobs on a latency-critical application uses the FCFS scheduling algorithm, as reported in Gupta et al. [4] work. On fog nodes, FCFS algorithm is employed to compute loop delay, consumption of energy, and network usage and also demonstrates how placing modules on edge is far superior to placing them on the cloud.

Bittencourt et al. [12] examined the allocation of resources in FC by utilizing three scheduling policies namely (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 second application is almost real-time, the first is delay-tolerant. Loop latency and usage of network were calculated to demonstrate that the distribution of resources should be based on the mobility requirements of the application. In order to lower costs and total response times in FC, the prioritized job scheduling technique is suggested by Choudhari et al. [25]. A job's priority is determined by its deadline upon receipt and placed in the fog layer based on this priority, neglecting other important factors such as network utilization and energy consumption. Additionally, Bitam et al. [26] employed the bio-inspired Bees Life Algorithm (BLA) to optimize 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 inferior memory allocation and run-time values. In addition, Gai et al. [27] optimized allocation of resources and Quality of Experience (QoE) using Reinforcement Learning (RL). In order to create cost mapping tables and allocate resources as efficiently as possible, two RL algorithms were suggested. A fog nodes cluster 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. Using DDQ, the paradigm calculated the Q-value for every Dynamic Voltage and Frequency Scaling (DVFS) technique. Rectified Linear Units (ReLu) was utilized as the activation function to keep the gradient from vanishing. A genetic method was presented by Nguyen et al. [29] to optimize the scheduling of jobs in the cloud-fog environment. The objective is to lower job performance times, with every chromosome representing a task assigned to a node. Crossover and mutation were employed to create a new population, and the obtained results surpassed those of BLA and Modified PSO. Using the iFogSim simulator, Rahbari et al. [30] recently built the Greedy Knapsack-Based Scheduling (GKS) algorithm for the scheduling of jobs [30]. The system was tested using EEGTBG and Video Surveillance, with results showing improved operational cost and energy consumption compared to FCFS and delayed scheduling. However, the authors overlooked network utilization, a critical 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. Since FC is still in its early stages, only a few strategies have been investigated for multi-objective optimization for job scheduling. The solutions that have been provided have different efficiencies in work scheduling when it comes to simultaneously preserving energy and delay in FC, as does the proposed methodology [32]. Here, we are implementing a job scheduling strategy for latency-sensitive applications in fog computing to optimize performance and delay.

Benzerogue et al. [33] introduced the Multi-Path Transmission Protocol for Video Streaming (MPTP-VS), which enhances video delivery by optimizing network resources such as bandwidth, jitter, and latency. While effective for improving video transmission in vehicular environments, its applicability is limited to this context. Similarly, Apat et al. [34] developed a hybrid meta-heuristic algorithm combining genetic algorithms, particle swarm optimization, and simulated annealing to optimize make-span and energy. Although promising in simulations, the model faces significant challenges in real-world implementation due to high cost and complexity. Saurabh et al. [35] focused on improving Quality of Service (QoS) through Sigmoid Neural Network Clustering (SNNC) and Entropy-Based Scheduling (EBS), using fog computing to process sensor data and achieve better performance in QoS parameters than traditional methods. Shirvani et al. [36] proposed a QoS-aware optimization algorithm to minimize bandwidth and power consumption, but its focus on just two parameters limits its broader applicability. Abu-Amssimir et al. [37] introduced a two-stage approach to reduce end-to-end latency in surveillance systems through a combination of greedy algorithms and depth-first search. However, this method does not account for factors like energy efficiency or cost. Finally, Varshney et al. [38] proposed a framework that evaluates trust in smart applications based on QoS parameters, but it overlooks key aspects such as user satisfaction and bandwidth utilization.

3. Design Framework

FC extends cloud computing by mediating between source nodes and the cloud. As illustrated in Figure 1, the FC architecture is distributed, hierarchical, and bidirectional, consisting of multiple layers: the cloud layer at the top, fog devices in the middle, and smart devices with sensors and actuators at the bottom of the IoE layer.

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 which is further transformed into signals, and sent to fog nodes for additional processing. As shown in Figure 1, the IoE layer enables device communication through two types of links: T2T and F2T. The T2T communication link allows the neighboring IoE devices to communicate via Bluetooth, Zigbee, and WiFi. These IoE devices establish a connection with higher-level fog nodes via an F2T communication channel.

3.2. Fog Layer

The fog layer serve as a bridge between the cloud and end devices, facilitating communication and independent evolution by loosely integrating the cloud with the IoE layers connected to fog. This intermediary layer handles specific processing tasks locally, enabling quick decision-making and low-latency responses essential for IoT applications, smart cities, industrial automation, and healthcare. While efficiently utilizing cloud services, the fog layer serves as the support of the IoE platform. It comprises various fog devices, including routers, switches, cellular base stations, and proxy servers, each with different processing, storage, and networking capacities. These devices are interconnected through a cloud server. As illustrated in Figure 1, fog nodes communicate with each other via F2F links and with IoE devices via F2T links.

3.3. Cloud Layer

The cloud represents the architecture's highest tier. It gathers information through 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 key 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 realise and collect data from the IoE devices using the SPAM architecture. After processing data, fog devices communicate the results to actuators for action. These applications utilize P2P, client-server, or cluster approaches to facilitate node cooperation [39, 40]. The DDF model, structured as Directed Acyclic Graphs (DAGs) with various modules, guides the development of these systems. In a DAG, an input module receives data, processes it, and forwards the output to the next module.

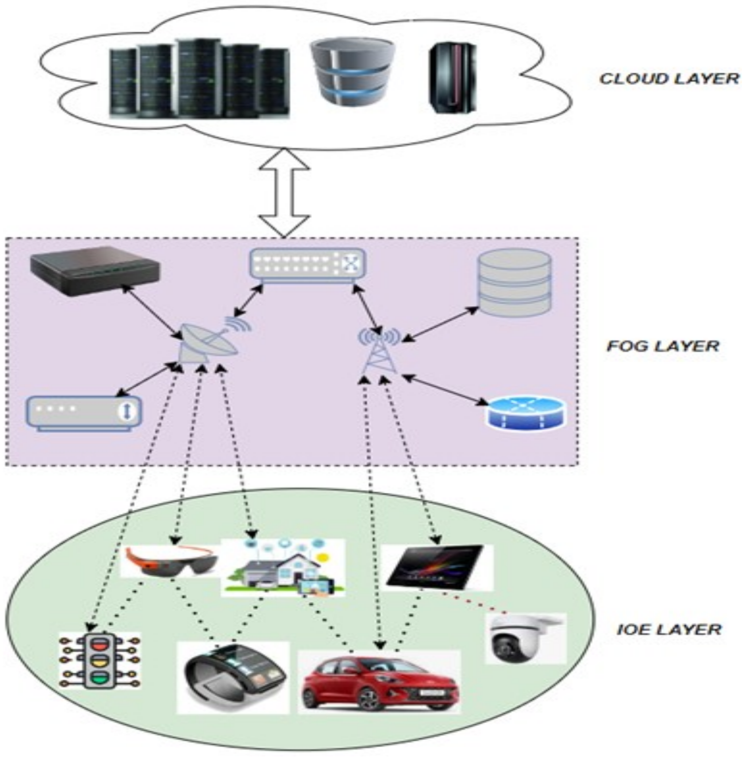
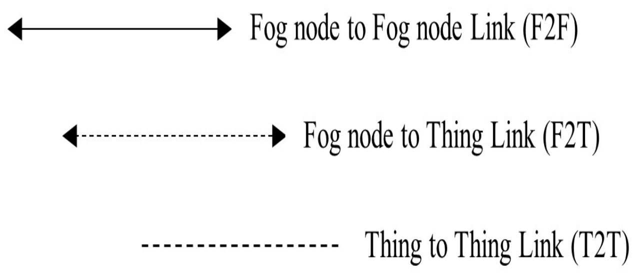


Figure 1. Schematic representing the architecture of Fog Computing.


Fog node to Fog node Link (F2F)
Fog node to Thing Link (F2T)
Thing to Thing Link (T2T)

4. 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 diverse devices with limited resources. 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 goals including optimization of loop latency, energy efficacy enhancement and network bandwidth optimization must be involved in the design along with the implementation of the job scheduling algorithm in fog:

4.1. Case Study 1: Advance Mining Technology (AMT) System

Mining is a vital and prominent industry that require extensive data analysis. However, the industry is also inherently hazardous, with the risks such as chemical reactions, dangerous gas emissions, suffocation, and rock slides posing significant threats to the safety of miming personnel during mineral and coal extraction. To mitigate these risks, it is essential to deploy IOT devices, including various types of sensors, to detect gases, chemicals, and environmental conditions, thereby providing early warnings to personnel about potential hazards. These sensors collect data from the environment before the digging process begins, which can help predict the likelihood of discovering coal and minerals in specific areas, ultimately saving both money and energy. Gas sensors can be strategically placed throughout mines to monitor and control gas emissions, as well as to measure the levels of biological gases in mines and tunnels. Additionally, the occurrences of numerous chemical reactions within mines can be extremely hazardous to workers. Therefore, gathering and analysing data on environmental conditions, biological gases, and chemical reactions is invaluable for predicting both the success of the digging process and the occurrence of dangerous events in the mining industry.

The data flow and application model of AMT is shown in Figure 2. Heterogeneous sensors, including Environmental Monitoring Sensors, Gas Monitoring Sensors, and Chemical Monitoring Sensors within the fog-based AMT, are connected to fog nodes through a router device. Four modules named (i) Central Module (ii) Environmental Monitoring Module (iii) Gas Monitoring Module (iv) Chemical Monitoring Module have been built. The Central Module will gather data from all types of sensors, categorize it, and then route it to the corresponding modules. For instance, data from Gas Monitoring Sensors will be directed to the Gas Monitoring Module, where it will be processed and analysed. The Gas Monitoring Module will assess the Gas values and send the results back to the Central Module. The response generated will be based on the Gas Monitoring Sensors values. Table 1 outlines the properties of tuples transmitted along the edges between the modules in the application. The workflow of the case study is represented by the following application loops:

(i) Application Loop 1: Gas Monitoring Sensor $\xrightarrow{GMS}$ Central Module $\xrightarrow{GMSTask}$ Gas Monitoring Module $\xrightarrow{GMSResponse}$ Central Module $\xrightarrow{Act_{C}ontrolGMS}$ Gas Activator

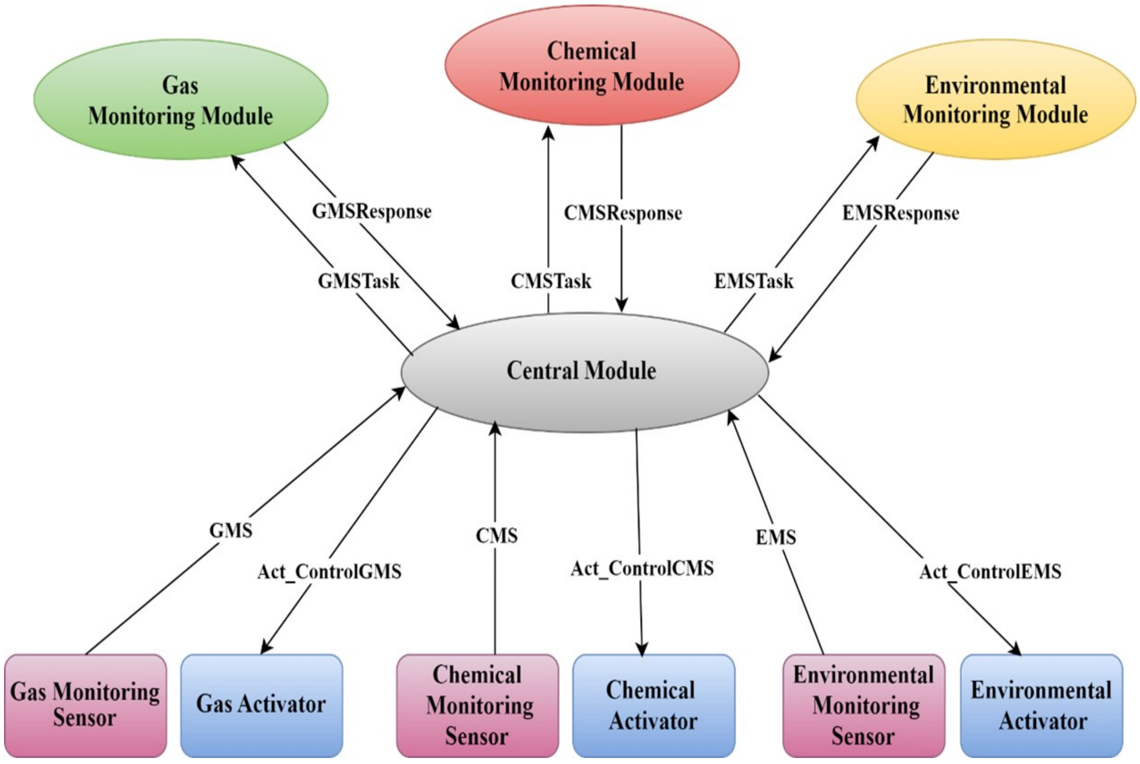


Figure 2. Application framework and Data Flow for Advance Mining Technology (AMT).

(ii) Application Loop 2: Chemical Monitoring Sensor $\xrightarrow{CMS}$ Central Module $\xrightarrow{CMSTask}$ Chemical Monitoring Module $\xrightarrow{CMSResponse}$ Central Module $\xrightarrow{Act_ControlCMS}$ Chemical Activator

(iii) Application Loop 3: Environmental Monitoring Sensor $\xrightarrow{EMS}$ Central Module $\xrightarrow{EMSTask}$ Environmental Monitoring Module $\xrightarrow{EMSResponse}$ Central Module $\xrightarrow{Act_ControlEMS}$ Environmental Activator

4.2. Case Study 2: Smart Healthcare System

The universality of IoE devices enables the examination of various healthcare system operations. FC has become an increasingly significant component of ubiquitous computing architectures [41, 42]. Cloud-based solutions in e-healthcare frequently lead to longer latency, which are unacceptable during crises. Utilizing FC, healthcare tasks computation can be executed by adjoining fog nodes, reducing delays and increasing accessibility [43]. Smart healthcare products can be categorized into various use cases, some critical and others non-critical [44, 45]. For example, patient information can be collected and stored for future examination by a doctor, where delays are tolerable. However, in critical situations, such as when a patient is in severe condition, data analysis is needed to produce emergency alerts. These operations are highly latency-sensitive, as delayed responses to emergency signals can endanger the life of patient. Thus, rapid healthcare case studies include three different application use cases based on their urgency and latency requirements as shown in Figure 3.

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

Table 1. A Summary of Tuples in the Advance Mining Technology (AMT) Application.

Origin	**Target**	**Tuple Format**	**Tuple CPU Span (MIPS)**	**Tuple Network Span (B)**
Gas Monitoring Sensor	Central Module	GMS	1500	2500
Chemical Monitoring Sensor	Central Module	CMS	1500	2800
Environmental Monitoring Sensor	Central Module	EMS	1700	3200
Central Module	Gas Monitoring Module	GMSTask	2200	4200
Central Module	Chemical Monitoring Module	CMSTask	1900	3700
Central Module	Environmental Monitoring Module	EMSTask	2800	5300
Gas Monitoring Module	Central Module	GMSResponse	250	590
Chemical Monitoring Module	Central Module	CMSResponse	180	5700
Environmental Monitoring Module	Central Module	EMSResponse	130	430
Central Module	Gas Activator	Act_ControlGMS	80	350
Central Module	Chemical Activator	Act_ControlCMS	55	290
Central Module	Environmental Activator	Act_ControlEMS	50	250

2. 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.
3. Medical Record Management System This is a database that holds patient records such as personal data, medical history, appointments, therapy, and test results of the patient. Every time a patient arrives at the hospital, the receptionist creates a new record in the cloud-based medical record management system. The following three application modules make this smart healthcare case study a fact.

1. Diagnostic Criteria and Patient Evaluation (DCPE): The DCPE module is installed on a lower-level fog device and gathers data from all three application modules. It gathers vital information from body sensors, analyse it, and examines it to produce emergency alerts. Additionally, it gets data for the medical record management and appointment systems but sends it to the facilitator module.
2. 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 allocated to patients. Patients are informed of this appointment schedule. It sends the patient's medical record and essential data to the medical record database.
3. Medical Record Database: This module uses cloud storage to obtain 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.

Table 2 lists the maximum CPU lengths of different tuples for intermodular communication. The CRITICAL tuple must be handled first because it includes the most essential information. The suggested algorithm chooses the first work to be executed from this list by ranking all arrived jobs according to the required million instructions per second (MIPS). The CRITICAL tuple experiences the least delay since the suggested algorithm processes the shortest tuples first. Let us assume that, at a given moment, the tuples CRITICAL(1), ORGANIZE(2), and

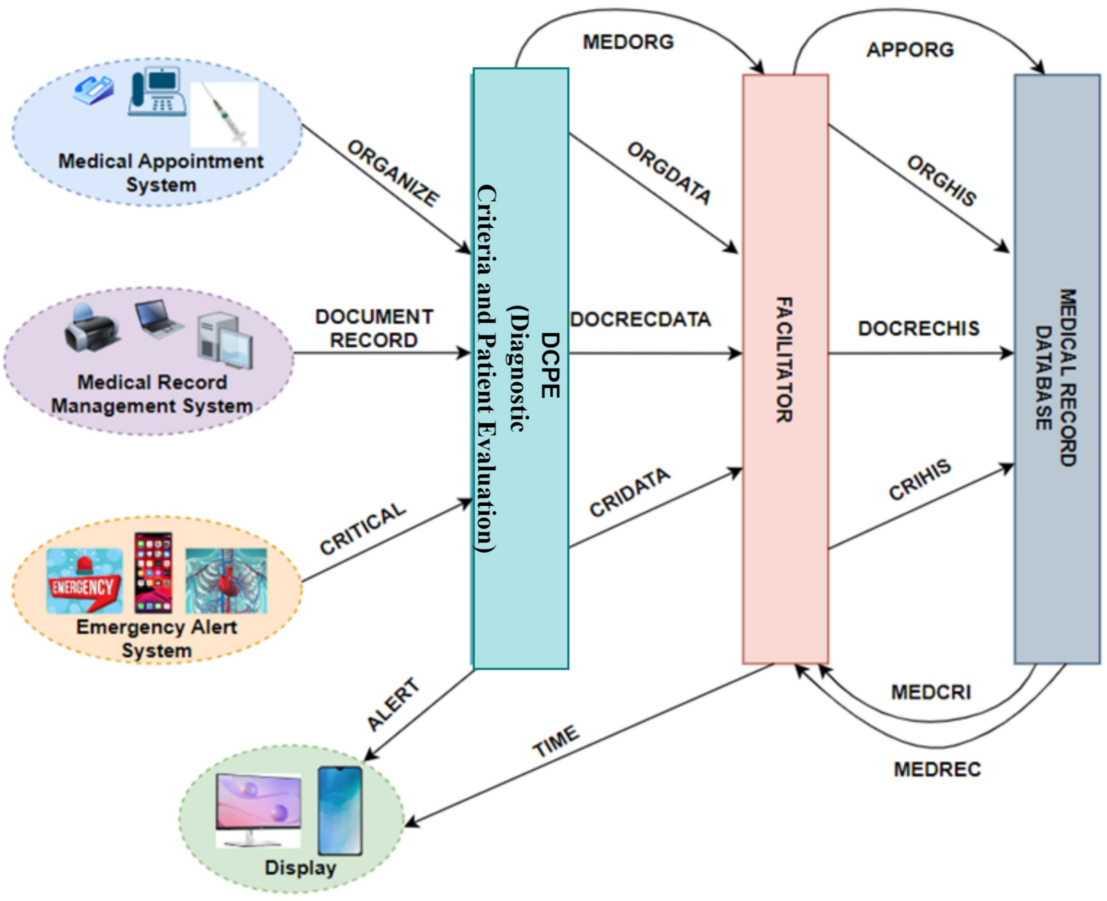


Figure 3. Diagramatic representation of case studies and its Modules.

DOCUMENT RECORD(3) arrive at DCPE in the order listed in Table 3. Table 4 presents the symbols for the proposed algorithm, with the job scheduling algorithm's steps detailed within the algorithm itself. The DCPE module is installed on a lower-level fog device and gathers data from all three application modules.

According to the suggested approach, the organizer determines how long the 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 allows processing resources to the incoming tuple, executed until it is finished if W_L is empty. T_i is added to the final tuples F_L list and set to T_f upon completion. T_i has been 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 W_L and assigns virtual machine (VM) until it is completed. T_{min} has been added to the tuples completed list F_L and set to T_f after it is done.

4.3. Complexity Analysis

Complexity of the suggested process is dependent on the waiting list (W_L) or N length. The suggested QoSPTS algorithm is designed with time complexity technique is O(nlogn), and the O(n) space complexity since the Scheduler sorts this list type, where n is the number of tasks waiting to be scheduled. The sorting mechanism used to prioritize tasks based on Quality of Service (QoS) attributes, such as latency sensitivity and resource requirements, forms the basis of this complexity. For each fog node, the scheduler sorts the incoming tasks and selects the highest-priority job for execution. While this complexity is efficient for small- to medium-sized fog environments,

Table 2. 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 3. Arrival sequence of tuples.

3	2	1	2	1	3	1	3	2

Table 4. Symbols for proposed algorithms.

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

it may require optimization for extremely large-scale systems where task counts could grow exponentially. Figure 4 demonstrates the block diagram of scheduling. A tuple is submitted to the QoSPTS work scheduler when it reaches a fog device by a sensor node or alternative 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 has been added to the queue and updated 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 chooses the shortest tuple from the top of the W_L for execution.

Here, the suggested approach was developed using the iFogSim toolbox, an improvement over iCloudSim. Resource management strategies in FC and IoE environments can be effectively simulated with iFogSim [4]. To construct the Scheduler, new classes are added to iFogSim, and a few existing ones are changed. The following is a summary of the classes:

1. Sensors: It is used for IoT sensor simulation by creating tuples containing data from sensor instances that could be sent to fog devices. This class can also be utilized to create tuples with varying sizes.
2. Fog Device: This class's instances are utilized to model various fog devices. The memory, CPU, storage capacity, and down-link and up-link bandwidths of every fog device are listed. Multiple-level fog nodes are possible. Using the tuples, every fog node can interact with others at a higher level and devices at the IoE layer. Based on the rising order of MIPS, the Scheduler can pick which arriving tuples every fog node can process.

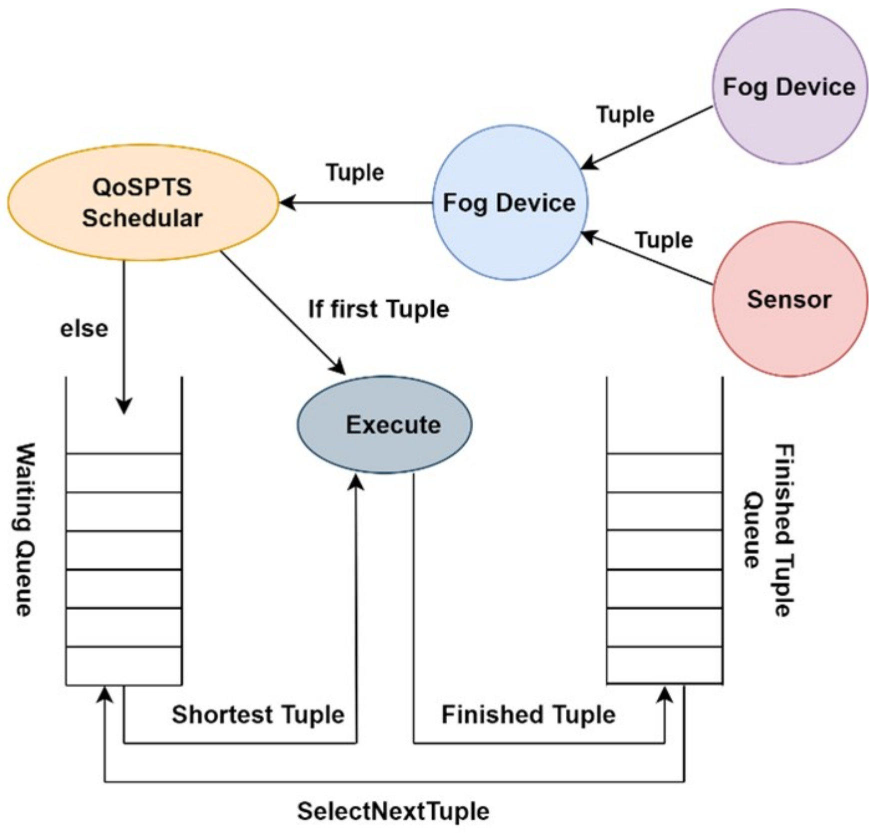


Figure 4. Scheduling Block Diagram.

3. Tuples: All of the entities in fog communicate with one another through tuple class instances. Each tuple's source, destination, and processing requirements are expressed in MIPS.
4. QoSPTS Scheduler: 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 waiting line when a tuple is finished.

Figure 5 illustrates the flow diagram for the QoSPTS algorithm. The process begins with the input of multiple tasks (T_1, T_2,. . . . . . ,T_n), where each task is assigned to a VM. The Scheduler then examines the waiting list W_L. If the waiting list is empty, indicating that the current task is the first one, it is executed immediately. If the waiting list is not empty, the current task T_i is added to the waiting queue W_L. The QoSPTS algorithm then sorts the waiting queue in ascending order based on the MIPS values, prioritizing tasks with higher efficiency. The task with the minimum MIPS value T_{min}, indicating the highest priority, is selected for execution first. After the selected task is executed, it is moved to the finished list F_L. The process continues in this manner until all tasks have been executed. Finally, the finished list F_L is displayed, showcasing the completed tasks and concluding the scheduling process.

Figure 6 illustrates the sequence of operations for processing a tuple sent from a sensor. The process begins with the sensor using the Transmit() method to send a tuple via the Send(tuple) method to a connected lower-level fog device. Upon arrival, the fog device invokes the process TupleArrival() method to estimate whether the tuple be processed locally or forwarded to a higher-level fog device. If the fog device is to process the tuple, the SubmitTuple() method sends it to the QoSPTS Scheduler. The QoSPTS Scheduler adds the tuple to the W_L and sorts it based on priority. The scheduleNextTuple() method then selects the highest-priority tuple, which is

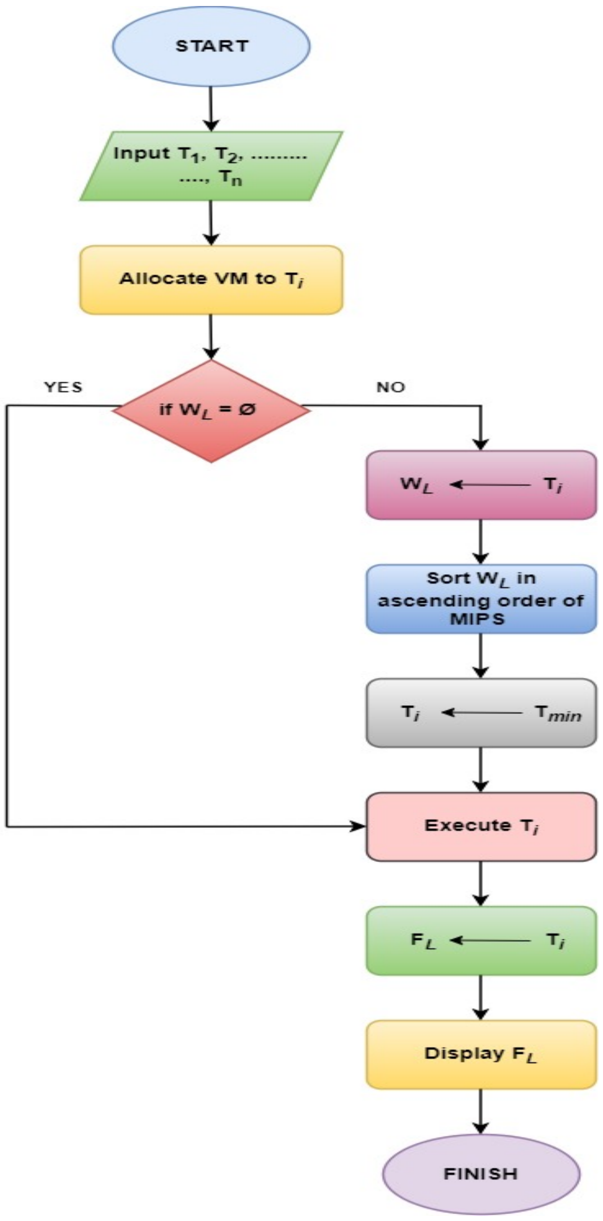


Figure 5. Flow Diagram of QoSPTS Scheduler.

subsequently sent back to the fog device for execution. After the tuple execution, the CloudletFinish() method is called, prompting the Scheduler to process the next tuple. The completed tuple is added to the F_L. The Scheduler continuously selects the highest-priority tuple from the head of the W_L for processing.

4.4. Scalability and Practical Considerations

The scalability of the QoSPTS scheduler is one of its core strengths, as it is designed to manage diverse fog nodes distributed across a network. However, to implement this in real-world environments, especially in large IoT ecosystems, certain optimizations are recommended. One such recommendation is to implement hierarchical scheduling layers, where local fog clusters manage scheduling within their regions, and a higher-level scheduler

```
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 $T_{min}$
                $T_f \Leftarrow T_{min}$
                $F_L \Leftarrow T_f$
            end if
        while Not end of tuples OR $W_L \neq$ Ø
        return tuple
```

oversees coordination between clusters. This approach would allow the algorithm to scale effectively across broader geographic areas and more extensive networks, distributing the computational load and reducing the overall complexity at any single node. In real-world deployments, fog nodes vary in computational power, memory, and energy capacity, requiring a tailored resource allocation strategy. Profiling these nodes enables the scheduler to assign tasks based on their specific capabilities, ensuring optimal resource use. For latency-sensitive applications like healthcare or autonomous driving, the QoSPTS scheduler must prioritize low-latency tasks dynamically. Clustering fog nodes near end devices can further enhance real-time performance. Given the energy constraints of many fog nodes, energy-aware scheduling techniques, such as Dynamic Voltage and Frequency Scaling (DVFS) and workload balancing, are essential to extend device lifespans while maintaining efficiency. Additionally, since fog environments are dynamic, the scheduler should integrate predictive mechanisms to adapt to fluctuating workloads in real-time, ensuring consistent performance under changing network conditions.

5. Evaluation and Validation of Performance Outcome

The current section presents the results of the suggested QoSPTS scheduler under various conditions, comparing its performance to the existing FCFS, Concurrent, Delay-Priority, and QoS-Aware scheduling algorithm [46, 12]. Mean loop latency, energy utilization, and network utilization are the three key metrics that has to be evaluated. Table 5 outlines the notations used in the related equations that is used for simulation setup.

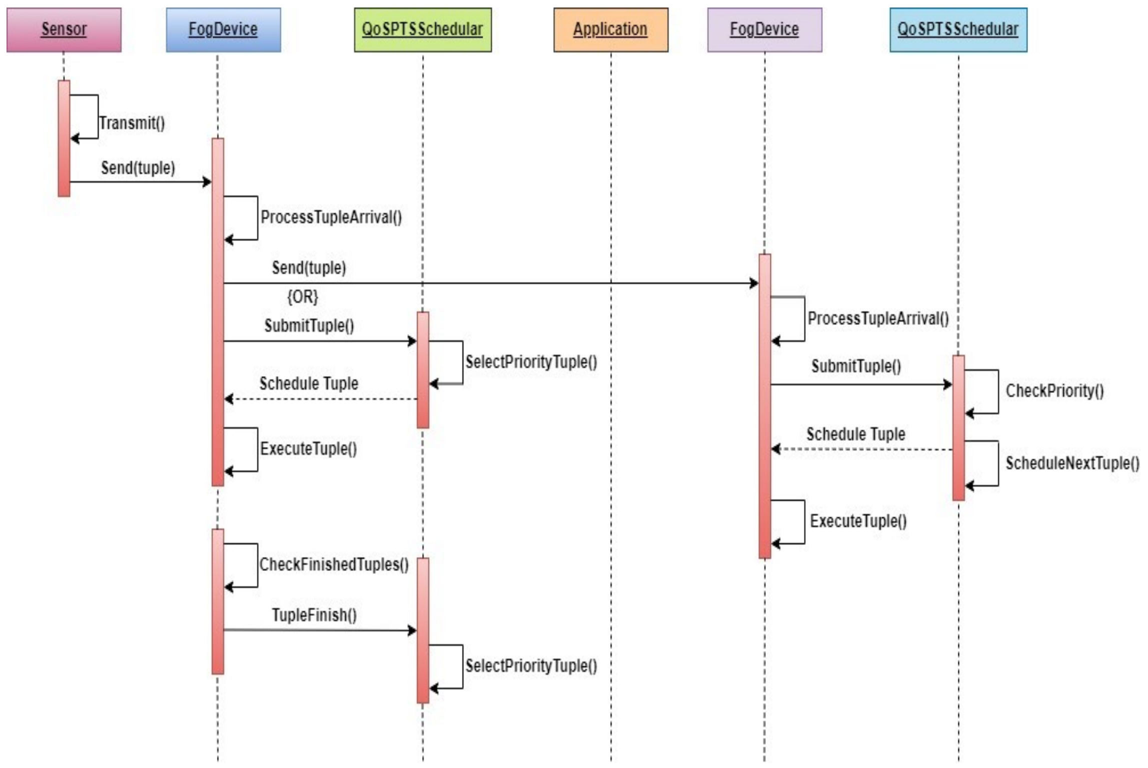


Figure 6. Sequence diagram illustrating the process of tuple transmission and scheduling in the QoSPTS algorithm.

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
ε_C	Current Energy Consumption
P_H	Host Power in Last Utilization
T_i	Update Time of Last Utilization
N_i	Network Size of i^{th} Tuple
L_i	Latency of i^{th} Tuple

5.1. Simulation Environment Setup

The suggested QoSPTS scheduler algorithm is applied and assessed via a simulated fog environment with the iFogSim toolkit [4]. To demonstrate the algorithm, the QoSPTS scheduler's performance was validated by a smart Healthcare case study, which is detailed in Section 4.1. There are delay-tolerant and latency-sensitive tuples in this case study. An Intel(R) Core(TM) i5-3210M CPU@2.50GHz computer having 4GB RAM and a 64-bit MS Windows 10 operating system has been utilized 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 comprise patient condition data that must be processed, examined, and handled as soon as possible. It is less critical because the Medical Appointment System creates an ORGANIZE tuple with an appointment request, and the appointment time is 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 later on the cloud. Here, we require a quick alert because CRITICAL tuples have been the most important. Calculating latency for end-to-end modules since ORGANIZE tuples have been less important, and DOCUMENT RECORD tuples are latency tolerant.

1. Critical Alert Loop: DCPE$\rightarrow$Facilitator$\rightarrow$display.
2. Medical Appointment Loop: DCPE$\rightarrow$Facilitator $\rightarrow$ DCPE$\rightarrow$display.
3. Medical Record Management Loop: DCPE$\rightarrow$Facilitator $\rightarrow$ Medical Record Database.
4. History: Facilitator$\rightarrow$ Medical Record Database$\rightarrow$ Facilitator$\rightarrow$DCPE.

Loop number one is the least delay-tolerant due to its critical condition, making it the most important. In contrast, loop number four, which involves storing patient data on the cloud for long-term analysis, can tolerate delays better than loops two and three, which handle appointment scheduling.

5.2. Parameters Specification

A detailed description of various devices used in the healthcare case study is presented in Table 6. Five parameter specifications are considered during the simulation, using 5 higher-level fog devices at level 2 (as facilitators). A thorough assessment is done by changing the fog nodes (acting as DCPE) at level 1. Three different sorts of tuples-Medical Appointment, Critical Alert, and Medical Record Management- are arriving at level 1 fog device. Each fog device is associated with 20, 40, 60, 80, and 100 fog nodes at this level. The specification have 100, 200, 300, 400, and 500 fog nodes. The smart healthcare case study application modules have RAM, MIPS, bandwidth, size of the module, and 10B, 100B, 10000B, and 100B/S.

Table 6. Device configuration in smart healthcare case study and its parameters.

Name	**CPU (MIPS)**	**RAM (MB)**	**Up-link Bw (MB)**	**Down-link Bw (MB)**	**Level Level**	**Rate Per MIPS**	**Busy Power(W)**	**Idle Power(W)**
Cloud	44800	40000	100	10000	0	0.01	1738	1242
Proxy server	7500	4000	10000	10000	1	0.00	109.342	81.4303
2^{nd} **level Fog Node**	5200	4000	10000	10000	2	0.00	109.342	81.4303
1^{st} **level Fog Node**	1100	1200	10000	1000	3	0.00	89.63	80.34

5.3. Mean Loop Latency

It employs a control loop to determine the latency of each module in the loop. Here, the average CPU time $(\overline{T_{CPU}})$utilized by all tuples of a particular kind is calculated to estimate the loop latency. Equation (1) is used to calculate the mean loop latency.

$$\overline{T_{CPU}} = \frac{ST_i \times N + ET_i - ST_i}{N+1} \text{If the mean CPU time for a specific kind of tuple has previously been calculated}$$

$$ET_i - ST_i \text{otherwise} \tag{1}$$

Where N denotes the total executed tuples of a specific type, ET_i is i_{th} tuple's ending execution time, and ST_i is indicated as the starting execution time. The equation calculates latency of each tuple in equation (2).

$$Latency = ET_i - ST_i, \pounds \in \tau \tag{2}$$

where tuple set τ is denoted as the current set. The comparison of mean loop latency produced by QoSPTS scheduler is compared with the results of baseline algorithms which are FCFS, Concurrent, Delay-Priority, and QoS-Aware. Figure 7 displays comparison of mean loop latency of the baseline algorithms and the proposed QoSPTS scheduler. The x-axis represents the number of nodes, while the y-axis shows the mean loop latency. The results shows that the proposed QoSPTS scheduler remarkably reduces the mean loop latency. Table 7 displays the comparative results for mean loop latency parameters. The result confirms that the proposed QoSPTS scheduler minimizes the latency by an overall average of 49.24%, 41.18%, 39.39%, and 13.89% in comparison of FCFS, Concurrent, Delay-Priority, and QoS-Aware respectively.

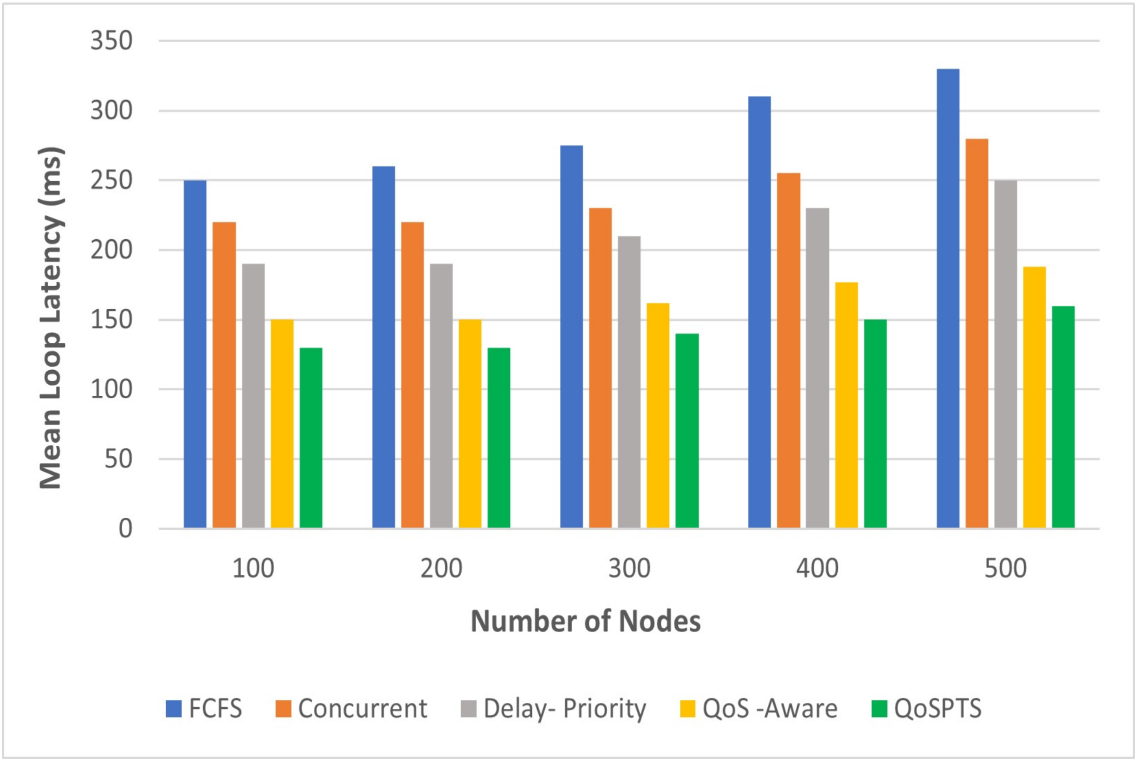


Figure 7. Mean Loop Latency in milliseconds (ms) for different scheduling techniques.

Table 7. The comparison table of Mean Loop Latency in percentage for different number of Fog nodes.

QoSPTS Vs Baseline scheduling techniques	Number of Fog Nodes				
	100	200	300	400	500
FCFS	48.0%	48.23%	47.94%	50.81%	51.21%
Concurrent	40.9%	40.8%	40.34%	41.17%	42.7%
Delay-Priority	31.57%	31.95%	35.04%	34.49%	36.11%
QoS-Aware	13.33%	13.72%	14.72%	13.79%	13.9%

5.4. *Energy Utilization*

Energy utilization (ε_{FN}) by a Fog device can be evaluated using equation (3).

$$\varepsilon_{FN} = \varepsilon_C + (T_C - T_i) \times P_H \tag{3}$$

The fog device energy could be calculated by utilizing every host power 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 has been denoted as the previous user's update time, and ε_C is denoted as the current energy consumption. This section details the cost of using the QoSPTS method on a cloud system compared to the FCFS algorithm.

Figure 8 displays the average energy used by fog devices in different scheduling techniques. The number of nodes is indicated along the x-axis, and the amount of energy utilized is expressed along the y-axis. It is observed from the figure that average energy consumed by the fog nodes using the proposed algorithm QoSPTS scheduler is less than the baseline scheduling techniques. Table 8 shows the comparison of energy utilization in terms of percentage value of proposed QoSPTS scheduler with baseline scheduling techniques. The results demonstrates that the energy is conserved through QoSPTS scheduler by an overall average of 6.85%, 5.23%, 4.32%, and 1.80% in comparison to FCFS, Concurrent, Delay-Priority, and QoS-Aware respectively.

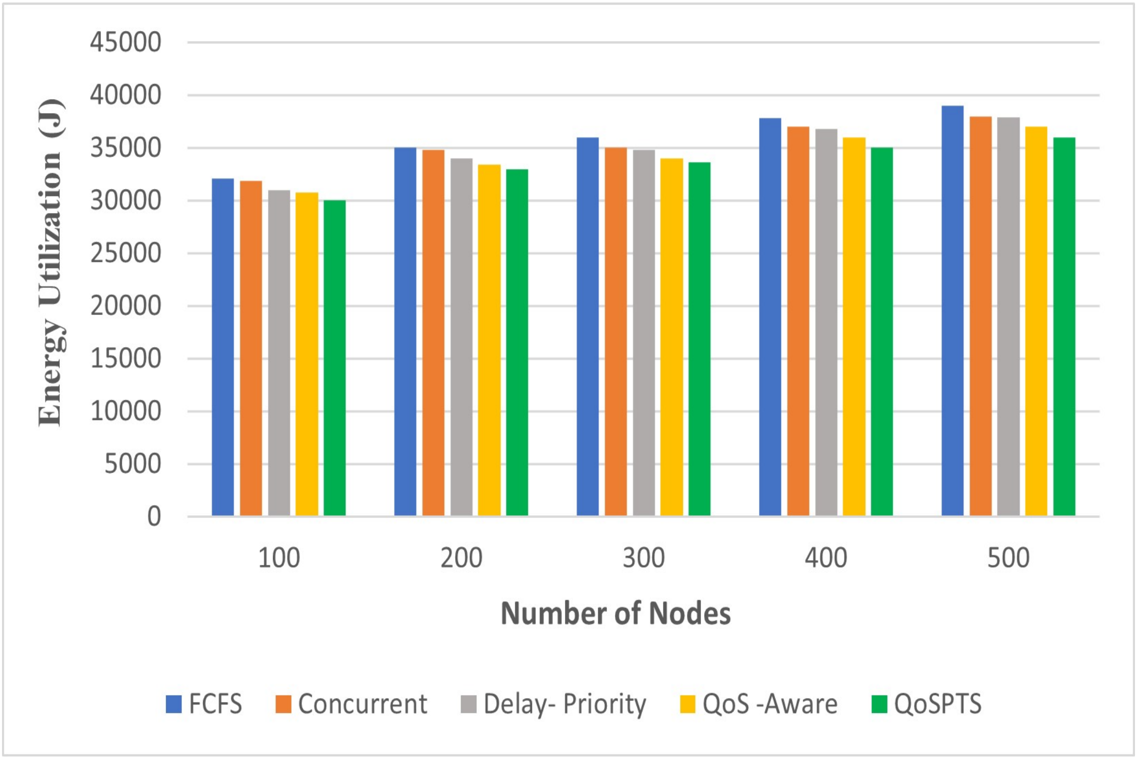


Figure 8. Energy Utilization in joules for different scheduling techniques.

Table 8. The comparison table of Energy Utilization in percentage for different number of Fog nodes.

QoSPTS Vs Baseline scheduling techniques	**Number of Fog Nodes**				
	100	**200**	**300**	**400**	**500**
FCFS	7.40%	6.55%	6.44%	6.40%	7.45%
Concurrent	5.36%	4.92%	4.97%	5.13%	5.75%
Delay-Priority	3.22%	3.24%	3.46%	3.83%	4.25%
QoS-Aware	1.63%	0.90%	2.05%	2.22%	2.17%

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 operates on a cloud network and functions poorly due to this congestion. By dividing the load among intermediary fog devices, Fog Computing contributes to a decrease in network congestion. The equation (4) is utilized to compute the utilization of the network.

$$N_{use} = \sum_{i=1}^{N} [L_i \times N_i] \tag{4}$$

Where N_i is denoted as the size of the network of the ith tuple, Li is indicated as the latency, and N is denoted as the total tuples. Figure 9 compares the network utilization of the QoSPTS Scheduler to the baseline techniques for various fog nodes. The x-axis depicts the number of nodes, and the y-axis shows the network utilization. The results show that there is a significant effect on the network utilization and promising results towards the proposed QoSPTS scheduler. Table 9 represents the comparative results of network utilization in percentage value when comparing the proposed QoSPTS scheduler with the baseline scheduling techniques. These results displays that the QoSPTS scheduler utilizes the network by an overall average of 57.12%, 47.93%, 48.66%, and 23.08% in comparison to FCFS, Concurrent, Delay-Priority, and QoS-Aware respectively.

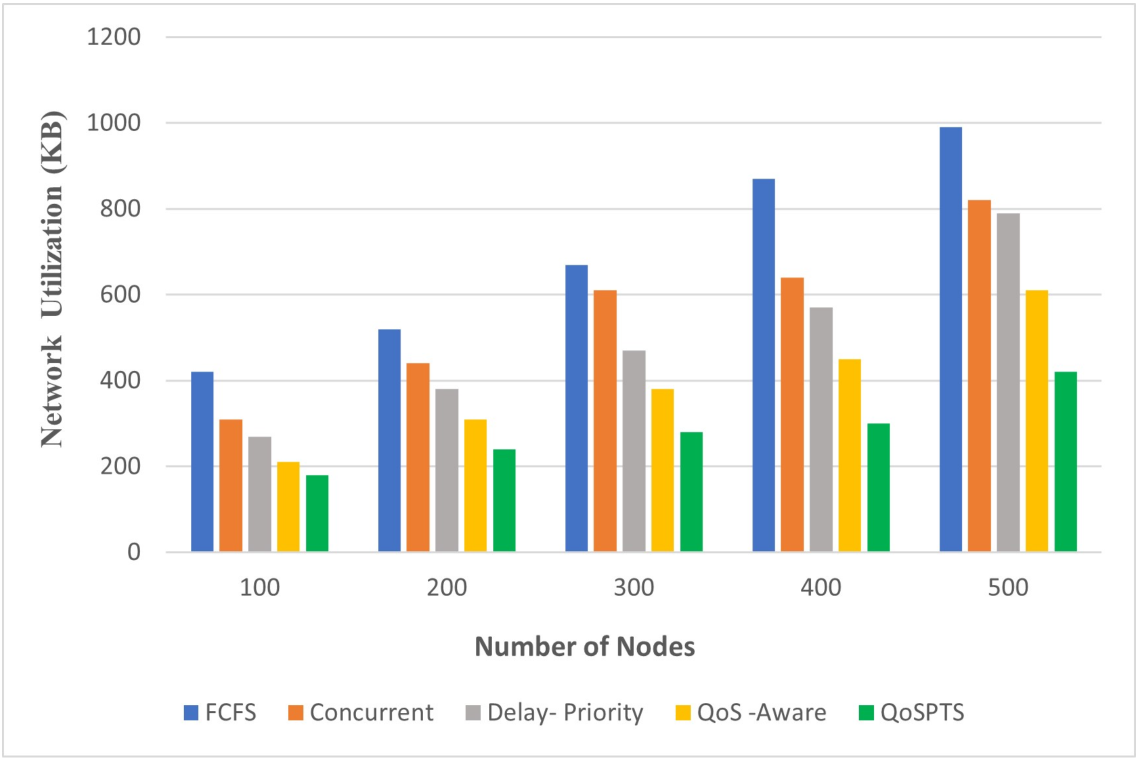


Figure 9. Network Utilization in KB for different scheduling techniques.

Table 9. The comparison table of Energy Utilization in percentage for different number of Fog nodes.

QoSPTS Vs Baseline scheduling techniques	**Number of Fog Nodes**				
	100	**200**	**300**	**400**	**500**
FCFS	57.14%	53.84%	56.25%	60.40%	57.57%
Concurrent	47.05%	44.57%	52.69%	46.55%	48.78%
Delay-Priority	33.33%	37.66%	38.29%	41.00%	46.83%
QoS-Aware	13.87%	22.07%	23.88%	24.55%	31.03%

5.6. *Cost*

The total execution cost is computed by using equation (5).

$$Cost = \sum_{i=1}^{FN} [C_C + (T_C - T_i) \times R_{mips} \times v_i \times M_{host}] \tag{5}$$

Where, C_C is denoted as the current cost, T_C shows the current time, last utilization update time is shown by T_i, rate/ms is shown by R_{mips}, v_i is the last utilization, and M_{host} represents the total milliseconds of all hosts. The last utilization v_i can be calculated by using equation (12):

$$v_i = min(1, AM_{host}/M_{host}) \tag{6}$$

Where AM_{host} is the host allotted MIPS.

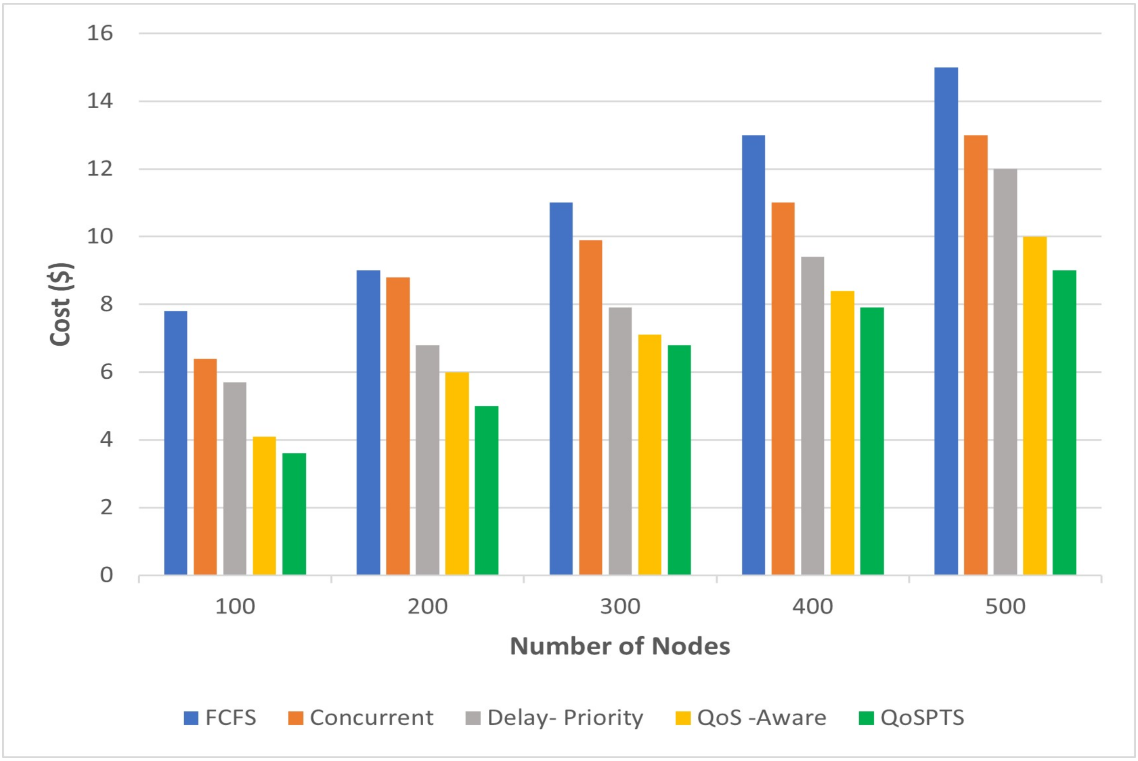


Figure 10. Cost for different scheduling techniques.

Figure 10 shows the average cost of the QoSPTS Scheduler to the baseline techniques for various fog nodes. The x-axis depicts the number of nodes, and the y-axis shows the cost. The results show that there is a significant effect cost reduction and promising results towards the proposed QoSPTS scheduler. Table 10 represents the comparative results of reduction of cost in percentage value when comparing the proposed QoSPTS scheduler with the baseline scheduling techniques. These results displays that the QoSPTS scheduler utilizes the network by an overall average of 44.18%, 36.26%, 30.30%, and 12.93% in comparison to FCFS, Concurrent, Delay-Priority, and QoS-Aware respectively.

Table 10. The comparison table of Cost in percentage for different number of Fog nodes.

QoSPTS Vs Baseline scheduling techniques	**Number of Fog Nodes**				
	100	**200**	**300**	**400**	**500**
FCFS	55.12%	45.65%	40.90%	39.23%	40.00%
Concurrent	46.15%	41.86%	34.34%	28.18%	30.76%
Delay-Priority	37.50%	27.53%	18.75%	16.84%	25.00%
QoS-Aware	16.66%	16.66%	12.16%	9.19%	10.00%

6. Conclusion and Future Prospects

Due to the increasing number of IoE devices producing vast amounts of data, cloud computing needs to catch up with the demands of real-time applications, which include mobility support, low latency, and location awareness. A new computing model known as Fog Computing has surfaced to circumvent these restrictions. Fog computing presents a significant difficulty in job scheduling because the edge devices have limited resources. The present work designs and implements an effective job scheduling strategy to minimize latency for delay-sensitive applications. It presents two case study as examples to highlight the requirements that IoE devices must meet in terms of both delay sensitivity and delay tolerance. The proposed QoSPTS algorithm significantly enhances the performance of Fog Computing environments, particularly for latency-sensitive applications. Compared to the traditional algorithms, QoSPTS demonstrates substantial improvements in several key areas. Specifically, the QoSPTS scheduler reduces mean loop latency, utilized the network, and utilization of energy at less cost by prioritizing critical tasks, such as emergency alerts, ensuring timely responses which are crucial in healthcare application. The effectiveness of the QoSPTS algorithm is evident in its ability to handle varying workloads efficiently. By sorting tasks based on priority and optimizing resource allocation, QoSPTS minimizes delays and enhances the overall user experience. The simulation results show that QoSPTS achieves a 34.66% reduction in latency and an 18.22% improvement in network bandwidth utilization compared to base line scheduling techniques. Additionally, the algorithm maintains lower energy consumption upto 4.37% by distributing tasks efficiently among fog nodes and reduce the cost upto 29.67%, thus extending the operational lifespan of these devices. This study also highlights that the QoSPTS algorithm can adapt to the dynamic nature of IoT environments, outperforming other existing scheduling techniques. While these traditional methods often fail to manage the increasing complexity and demand of real-time applications, QoSPTS effectively prioritizes and processes tasks, leading to more reliable and responsive systems. Therefore, it can be concluded that QoSPTS algorithm provides a robust solution for optimizing performance in fog computing, addressing the critical issues of latency, energy efficiency, network utilization, and cost.

The parameters used for the proposed model development have been configured as mean loop latency, network utilization, energy utilization, and cost. However, the model has not been implemented on other parameter like deadline miss ratio, user satisfaction, failure rate etc. Furthermore, the fine tuning of the parameters can be done and performance can be analysed for achieving the most optimal and efficient results. In future work, the integration of machine learning and artificial intelligence techniques can improve the performance of the fog based scheduling. Additionally, incorporating AI-driven methods to address the growing complexity of emerging paradigms can support autonomic resource management, enhancing the performance by considering the various QoS parameters like availability, throughput, deadline miss ratio, service delivery time etc. Future research should aim to integrate advanced scheduling techniques such as meta-heuristics and reinforcement learning to further enhance the capabilities of fog computing networks, ensuring even greater efficiency and scalability.

7. Declaration

Competing Interests

The authors have no relevant conflicts of interest to disclose.

Funding

The author received no funding support from any agencies or firms.

Acknowledgement

The authors are thankful to Editor and Reviewers for their valuable, insightful, and constructive feedback, which has significantly improved the revised version of the manuscript.

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*, Wireless Communications and Mobile Computing, 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*, Software Practice and Experience, 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 Communications Magazines, 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 Transactions Cloud Computing, vol. 7, no. 2, pp. 196–209, 2019.
9. S. Mishra Tiwari and S. Jain, *Ontologies as a semantic model in IoT*, International Journal of Computer Applications, 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*, Wireless Communications and Mobile Computing, 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*, International Journal of Computer Applications, 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 Computing, 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*, Journal of Network and Computer Applications, 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*, Journal of Super computing, 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*, 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*, International Conference on Signal Processing, Computing and Control (ISPCC), pp. 407–412, 2021.
17. I. Seth, S. N. Panda, and K. Guleria, *Introducing intelligence in vehicular ad hoc networks using machine learning algorithms*, ECS Transactions, vol. 107, no. 1, pp. 8395-8410, 2022.
18. M. Rani, K. Guleria, and S. N. Panda, *Blockchain technology novel perspective for cloud security*, 18th 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 Transaction and Computing, 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 Journal, 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*, 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*, Wireless Personal Communications, vol. 102, no. 2, pp. 3210-3223, 2018.
25. T. Choudhari, M. Moh, and T.-S. Moh, *Prioritized task scheduling in fog computing*, 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*, Enterprise Information System, 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*, Applied Soft Computing, 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 Transactions on Services Computing, 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*, Applied Sciences, 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 Journal of Electrical Engineering and Computer Sciences, 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*, International Journal of Intelligent System and Applications Engineering, vol. 12, no. 4s, pp. 388–405, 2023.
33. S. Benzerogue, S. Abdelatif, S. Merniz, S. Harous, and L. Khamer, *Multi-Path Transmission Protocol for Video Streaming over Vehicular Fog Computing environments*, IEEE Access, 2024.
34. H. K. Apat, B. Sahoo, V. Goswami, and R. K. Barik, *A hybrid meta-heuristic algorithm for multi-objective IoT service placement in fog computing environments*, Decision Analytics Journal, vol. 10, p. 100379, 2024.
35. Saurabh, and R. K. Dhanaraj, *Enhance QoS with fog computing based on sigmoid NN clustering and entropy-based scheduling*, Multimedia Tools and Applications, vol. 83, no. 1, pp. 305-326, 2024.
36. M. H. Shirvani, and Y. Ramzanpoor, *Multi-objective QoS-aware optimization for deployment of IoT applications on cloud and fog computing infrastructure*, Neural Computing and Applications, vol. 35, no. 6, pp. 19581-19626, 2023.
37. N. Abu-Amssimir, and A. Al-Haj, *A QoS-aware resource management scheme over fog computing infrastructures in IoT systems*, Multimedia Tools and Applications, vol. 82, no. 18, pp. 28281-28300, 2023.
38. S. Varshney, R. Sandhu, and P. K. Gupta, *Developing MCDM-based technique to calculate trustworthiness of advertised QoE parameters in fog computing environment*, In Machine Learning, Image Processing, Network Security and Data Sciences: Select Proceedings of 3rd International Conference on MIND, pp. 705-714, 2021.
39. 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.
40. M. Rani, K. Guleria, and S. N. Panda, *Cloud Computing An Empowering Technology: Architecture, Applications and Challenge*, 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), pp. 1–6, 2021.
41. A. M. Rahmani, *Exploiting Smart E-Health Gateways at the Edge of Healthcare Internet-of-Things: A Fog Computing Approach*, Future Generation Computer Systems, Vol. 78, No. 4, pp. 5612-5629, 2017.
42. 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.
43. K.-M. Chao and J.-Y. Chung, *Edge and fog services*, Service Oriented Computing and Applications, vol. 13, no. 2, 2019.
44. 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.
45. M. Rani, K. Guleria, and S. N. Panda, *State-of-the-Art Dynamic Load Balancing Algorithms for Cloud Computing*, ECS Transaction, vol. 107, no. 1, pp. 8339-8347, 2022.
46. Cardellini V, Grassi V, Presti FL, Nardelli M., *n qos-aware scheduling of data stream applications over fog computing infrastructures*, Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC), pp. 271-276, 2015.