Proceedings of the Pakistan Academy of Sciences: A Physical and Computational Sciences 61(1): 19-31 (2024)

ISSN (Print): 2518-4245; ISSN (Online): 2518-4253
https://doi.org/10.53560/PPASA(61-1)674

Pakistan Academy of Sciences

Research Article

Efficient Resource Scheduling in Fog: A Multi-Objective Optimization Approach

Tayyiba Hameed, Bushra Jamil, and Humaira Ijaz*

Department of Information Technology, University of Sargodha, Sargodha, Pakistan

Abstract: Fog computing is a novel idea that extends cloud computing by offering services like processing, storage, analysis, and networking on fog devices closer to IoT devices. Numerous fog devices are required to process the ever-growing amount of data generated by IoT applications. The heterogeneous tasks from various IoT applications compete for a limited number of resources of these devices. The process of assigning this set of tasks to different available fog nodes according to QoS requirements for processing is resource scheduling. Resource scheduling aims to optimize resource utilization and performance metrics however, the dynamic nature of the Fog environment, resource-constrained, and heterogeneity in fog devices make resource scheduling a complex issue. This research presents the design and implementation of a multi-objective optimization-based resource scheduling algorithm using Modified Particle Swarm Optimization (MPSO) that addresses the application module placement and task allocation issues. This two-step MPSO-based resource scheduling model finds the optimal fog node to place each application module and assigns appropriate tasks to the most optimal fog nodes for execution. The proposed model unlocks the full potential of fog resources along with maximization of overall system performance in terms of optimization of cost, latency, energy consumption, and network usage. The simulation results indicate that using MPSO energy consumption is reduced by 53.94% and 43.58% as compared to First Come First Serve (FCFS) and Particle Swarm Optimization (PSO), respectively. The loop delay, network usage and cost using MPSO are reduced by 40.3%, 67.69% and 90.01% respectively, as compared to PSO algorithm.

Keywords: Fog Computing, MPSO, Multi-Objective Optimization, Resource Scheduling, Task Allocation, Cloud Computing, Internet of Things (IoT).

1. INTRODUCTION

The Internet of Things (IoT) is a groundbreaking technology that has the potential to transform the manner we live and work [1]. It's a network of physical objects, machines, gadgets, automobiles, and other things, having sensors, software, and connectivity, that enables them to gather and share data to make intelligent decisions. IoT makes devices intelligent and responsive to their surroundings. This interconnected network of devices spans various industries, from smart homes, cars, and cities to healthcare and manufacturing, and has the potential to enhance automation, decision-making, and efficiency. This interconnectivity of devices has led to an accelerated growth in the amount of data produced by these devices. Currently, cloud-based IoT (CIoT) is providing a powerful solution for storing, processing and analyzing this sheer amount of data. As CIoT architecture is centralized and the cloud data centers are multi-hops away that increases latency bandwidth consumption and network bottlenecks while processing this sheer amount of IoT data [2].

Later on, Edge computing was used to handle the high latency problem in delay-sensitive applications by providing storage and processing resources near the end user IoT devices. Edge computing provides many advantages like high-speed processing, low latencies, and real-time availability of network resources. This paradigm can solve many issues like energy usage, security, and privacy by reducing the distance the data must travel [3]. However, although edge devices have short access latencies because of their proximity to end users, they are still resource-constrained and prone to availability issues.

Received: December 2023; Revised: February 2024; Accepted: March 2024
*Corresponding Author: Humaira Ijaz <humaira.bilalrasul@uos.edu.pk>

To address this problem fog computing has been introduced, which just evolved as a logical extension of cloud computing [4]. The term "Fog Computing" refers to moving services like computing, processing, storage, and networking services near the proximity of the end user [5]. This results in reduced latency, faster decision-making conservation of network bandwidth, and improved reliability. Fog computing has different characteristics like dense distribution of heterogeneous, resource-limited fog nodes, context awareness, mobility, real-time interaction, and executing diverse-natured IoT applications. The advancement in communication technologies and the development of pervasive computing caused a rapid surge in the number of IoT applications and IoT devices that generate a massive amount of data to be processed by these fog devices [6]. The heterogeneous, resource and energy-limited, dynamic nature of fog applications makes resource management challenging [7]. Among various resource management techniques, resource scheduling is crucial for taking full benefits of fog computing. Resource scheduling determines when and where different applications or services should use resources, such as CPU, memory, storage, and network bandwidth. Resource scheduling aims to identify the available resources and allocate them to specific applications to ensure their effective utilization and timely completion of latency-sensitive tasks. Efficient resource allocation thus leads to better resource utilization, reduced latency, energy consumption, and improved user experience. Resource scheduling becomes a challenging issue due to the diverse and highly dynamic nature of fog computing. We have reviewed the existing resource scheduling techniques for fog computing environments presented by researchers and highlighted different metrics optimized in these studies, such as latency, network bandwidth, energy consumption, cost, and quality of service, etc.

Benedetti *et al*. [8] developed the distributed job scheduler JarvSis for fog-based IoT applications. To optimize time and energy consumption, Movahdi *et al*. [9] employed Integer Linear Programming to formulate the task scheduling problem in fog computing environment.

Wu *et al*. [10] presented a multi-objective algorithm that learns and optimizes the fuzzy offloading technique from various IoT applications and allocate tasks in fog computing environment. Jamil *et al*. [11] proposed a heuristic-based task-scheduling algorithm that schedules the tasks on the fog nodes according to their processing needs. iFogSim is used for implementation with the objective to optimize delay and energy consumption. In another study, Josilo [12] considers a fog computing system to offload the computational tasks to nearby devices or an edge server.

The selection and distribution of resources are studied by Zhang *et al*. [13], who use the Stackelberg game for solving resource allocation problems. Zhang [14] describes a game-based resource management system consisting of three entities authorized users, fog nodes, and data service operators. This stable paradigm maximizes the usefulness of each entity.

The resource scheduling between fog devices in the same fog groups is presented by Sun *et al*. [15] and Bitam *et al*. [16] using meta-heuristic algorithms. Chen and Wang [17] developed two dynamic scheduling methods based on dynamics in the fog infrastructure response times and latency.

Bittencourt *et al*. [18] proposd a paradigm for scheduling that categorizes applications and client mobility as two key elements to provide effective resource management related to scheduling. Wadhwa and Aron [19] introduced OSCAR to optimize task scheduling in fog-based IoT environments. They use QoS-based scheduling and task clustering to enhance system throughput and increased bandwidth usage. Du *et al*. [20] proposed a computational offloading method to reduce device delay and energy usage. The method solves the nested resource allocation problem by taking offloading decisions.

Shahidani *et al*. [21] suggested a multi-objective task scheduling approach for fog-edge-cloud environment using reinforcement learning. They optimized network congestion, service delays, energy consumption and network usage. Subbaraj *et al*. [22] proposed a hybrid metaheuristic optimization technique called the Cow Search Algorithm (CSA) for solving resource allocation and scheduling issues in the fog environment. The proposed algorithm combines the pivoting

rule (local search) with the crow search algorithm (global search) to harness the advantages of both exploration and exploitation.

Liu *et al*. [23] introduced a resource-scheduling strategy to address the challenges of load balancing and task scheduling in a fog computing environment to decrease latency and energy consumption. The proposed technique uses a particle swarm optimization algorithm to search the optimal load balance on fog devices in a single cluster. They use a particle swarm algorithm for genetic joint optimization and an Artificial Bee Colony algorithm (PGABC) for optimizing the task scheduling among fog clusters.

After reviewing the existing literature, we found that most resource scheduling approaches are based on mono or bi-objective optimization that cannot balance these objectives effectively generating sub-optimal solutions. Therefore, we need a multi-objective optimization-based resource scheduling that generates a set of Pareto-optimal solutions to simultaneously trade off multiple and conflicting objectives of various heterogeneous tasks of different applications in a dynamic fog computing environment. There is a need to design an efficient multi-objective optimization-based resource scheduling approach that will perform optimal application module placement and allocate resources of fog nodes to tasks along with efficient utilization of resources and performance optimization. Therefore, we have designed and implemented a multi-objective optimization-based resource-scheduling algorithm based on MPSO to optimize different metrics. We have compared the results of MPSO with PSO and FCFS using the iFogsim simulator.

This research paper is concerned with the design, implementation, and evaluation of a multi-objective optimization-based resource scheduling algorithm that efficiently allocates resources to available suitable fog nodes according to the given requirements. The following are the main contributions of the suggested work.

1. To review current resource scheduling techniques for fog computing.
2. To investigate the limitations and shortcomings of the recently used task allocation (resource scheduling) methods.
3. An MPSO-based two-level model for resource scheduling is proposed that combines application module placement and task allocation with objectives to minimize latency, network usage, cost, and energy consumption to optimize resource utilization.
4. Moreover an effective task-scheduling algorithm based on the traditional Shortest Job First (SJF) is applied that prioritizes and executes the shortest tasks first on fog nodes, which minimizes the latency for critical tasks.
5. To compare the performance of the suggested resource scheduling algorithm to other algorithms and evaluate the execution results.

The remainder of the article is structured as follows. Fog computing architecture and proposed modified optimization-based (MPSO) resource scheduling are presented. Then the proposed simulation model and evaluation results of the MPSO-based resource scheduling algorithm are described. Finally, we explain the present research outcomes and provide recommendations for further study.

2. MATERIALS AND METHODS

The Fog-IoT model is comprised of three layers, each performing crucial functions: IoT, fog, and cloud layer. Figure 1 presents the architecture of Fog computing.

IoT Layer: The IoT layer is the bottommost layer in FIoT architecture. It includes devices such as sensors, actuators, and internet-connected IoT devices closer to the end-users [4]. This layer gathers and transfers data generated by IoT devices to upper layers for further processing.

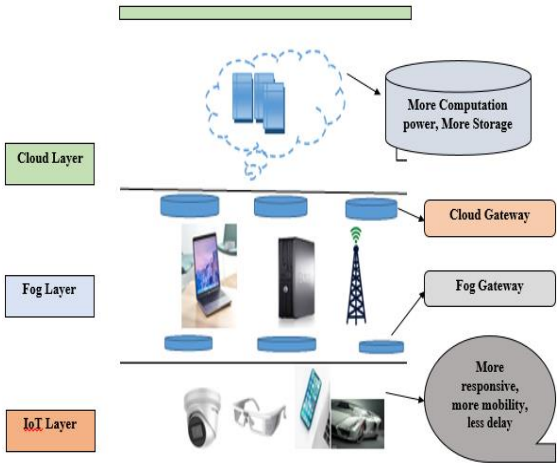

Fig. 1. Fog Computing Architecture.

Fog Layer: This intermediate layer is made up of heterogeneous fog nodes having different processing, storage, and networking capabilities; for example, routers, roadside units, proxy servers, cellular base stations, and mobiles [6]. These fog nodes collect data from the IoT layer, process it, perform necessary computations for intelligent decisions, and transfer it to the Cloud Layer for long-term decision-making.

Cloud Layer: The topmost layer servers as the central repository comprised of many powerful computers and data centers. Data centers give storage and processing services to IoT devices for storage and long-term analysis.

Fog devices are heterogeneous, dynamic, and resource-constrained, and the applications send requests/tasks for processing to these nodes [24]. Firstly, the resources of fog nodes are compared with the computational needs of these tasks, and if the resources are sufficient to process these tasks they should be allocated to that node and executed. The task will be shifted to the cloud for execution if no resources are available at the fog nodes. Figure 2 presents the block diagram of existing edge-ward resource scheduling in which modules are placed using FCFS on the nearest edge, fog nodes and cloud. Using FCFS, fog node manager places incoming module on fog nodes in order or arrival without prioritizing computing layers.

Effective task allocation or resource scheduling strategies are necessary to optimize resources, accelerate response times, and minimize cost and energy utilization. Therefore, we have developed

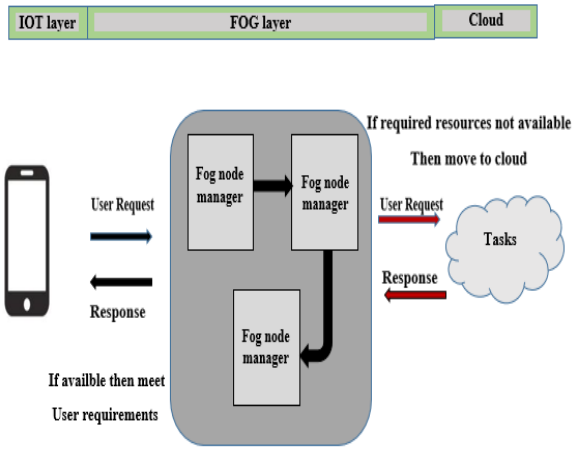

Fig. 2. Block Diagram of existing edge-ward resource scheduling.

a multi-objective optimization-based resource scheduling approach to allocate resources of fog nodes to user's jobs according to their needs while maximizing resource utilization, minimizing latency, cost, energy consumption, and network usage.

2.1. Case Study

We have selected a healthcare scenario as a fog computing use case that includes an Appointment Coordination System, Health Record Management System, and Urgent Notification System as healthcare applications [12].

Urgent Notification System: The Urgent Notification System monitors patient's critical data such as heart rate, sugar, blood pressure, oxygen saturation and generates instant response to this critical data.

Appointment Coordination System: This system manages appointments of different patients in different time slots with less critical data as compared to data of urgent notification systems.

Health Record Management System: The Healthcare Record Management System contains information such as names, addresses, and historical records of health that are recorded and stored on the cloud for future analysis and decision-making.

Challenges: Accurate results and quick responses are essential for real-time health data processing. However, various tasks compete in a heterogeneous environment for a limited number of resources of fog devices. Therefore, one of the main difficulties in fog and edge computing is resource management. For instance, a patient's status in a smart healthcare system requires quick notice to rescue the patient. It is difficult to allocate resources to requested tasks while maintaining low latency and efficient energy use. Therefore, we have applied a multi-objective optimization-based resource scheduling algorithm called MPSO to solve the resource management issues of latency, energy consumption, and resource utilization in smart healthcare.

2.1.1 MPSO algorithm for resource scheduling

Modified Particle Swarm Optimization (MPSO) [25] is an ideal solution for resource scheduling because it can handle multiple objectives and constraints

due to its meta-heuristic nature. It is adaptive to dynamic environments and easy to implement in a distributed and scalable environment. In short, MPSO-based resource scheduling can improve the efficiency, adaptability, and user satisfaction of the fog computing system, making it more suitable for a wide range of applications. MPSO algorithm follows the steps given below:

1. *Issue Propagation*
   i. Reduce Latency: The main objective is to reduce the time needed to process and analyze health data.
   ii. Reduce Energy Consumption: The second objective is to reduce energy use, while carrying out tasks through wearable technology and fog nodes.
2. *Population Initialization*
   Each possible solution in the population represents a method for task allocation (resource scheduling). Each solution specifies, taking into consideration processing power and energy resources, which tasks are assigned to which fog nodes. The population (swarm) is initialized randomly in the MPSO algorithm.
3. *MPSO Execution*
   To identify a collection of best-optimal solutions that balance minimizing delay, network usage, cost and energy usage, MPSO develops the population. To select the best work task allocation arrangements, MPSO refines solutions by adding a new variable named inertia weight $\boldsymbol{w}$ to the initial part of velocity updation.
4. *Fitness Evaluation*
   Based on task execution time, communication latency, and energy consumption, solutions are assessed. The properties of wearable technology, fog nodes, and the communication network are used to determine these measures.
5. *Inertia Weight*
   The best optimal solutions found by MPSO using additional concept inertia weight in PSO reflect various trade-offs between reducing latency and maximizing energy efficiency.
6. *MPSO Swarm Update*
   MPSO particles modify their locations to move in the direction by updating their local best and global best by changing velocity and position. The task allocation (resource scheduling) configurations are improved using the swarm exploration method.
7. *Resource Allocation Decision*
   The trade-offs between minimizing latency and energy usage are considered while making final resource allocation selections. To save energy, fewer time-sensitive processes might be assigned to the cloud and, critical jobs can be allocated to fog nodes for speedy analysis. To utilize resources, we arranged tasks into two categories: critical and non-critical.

Fog computing's resource allocation system enhances real-time health data analysis in healthcare settings by combining MPSO algorithms. In a hospital setting, this method assures prompt replies, resource efficiency, and energy effectiveness, all of which improve patient care and medical decision-making.

Therefore, we have proposed an MPSO-based two-level model for resource scheduling that combines application module placement and task allocation with the objectives of minimizing latency, network usage, cost, and energy consumption to optimize resource utilization and improve system performance. In the first step, the application module placement is used to find the optimal fog node to place each application module, considering different criteria such as the computing capacity, available bandwidth, and energy consumption of each node.

Afterward, task allocation deals with assigning the appropriate tasks to the most optimal fog nodes for execution by taking into account the available resources of fog nodes and the requirements of various heterogeneous tasks of different applications. This two-step MPSO-based resource scheduling model has paved the way to unlock the full potential of fog resources and optimize overall system performance.

The algorithm for MPSO-based resource scheduling is given as (Algorithm 1). We adopted MPSO for resource scheduling in our proposed algorithm. To choose the optimal tool for processing incoming jobs, MPSO is employed. In this algorithm, we've created a resource pool in which all resources are kept. In the first step, the decision maker will decide where to place the application modules. In the second step, when a task arrives, it will compare with the resources of each fog device using MPSO before sending it to fog devices; if the

device is found then it will be added to the hash map (queue). If the current device is not suitable, then the same steps will be performed for searching the next devices. If not, all available devices fulfill the task's needs, it will move to the cloud.

Algo. 1. MPSO-based resource scheduling.

```
Input: Tuples
Output: Task execution
    Fog devices created
    Application created
    Tuple generated by the sensor
    Reached at gateway
    Passed to the decision-maker
    The decision maker placed the modules on
    respective fog nodes
    The decision maker checks the available
    best fog node against the generated task by
    Algorithm 2
    Fog node process task
    Sent back to the respective gateway
    The result sent to the Actuator
For all unassigned tasks t1, t2, t3…tn do
    send to Cloud
End
```

Algorithm 2 will assign resources to tasks to the appropriate fog nodes.

Algo. 2. Steps of MPSO

```
Input: Tuple's Mips, Resources
Output: Best fog node
For received request t1, t2, t3…tn do
    For Bz = b1, b2, b3….bn do
        Bz <=  G (leastloaded)
            Allocatelbz  <=  Bz
    End
    Allocate gb = leastlbz
    Assign next task t <= G (gb)
    If nextassigned == last used GB then
        Goto step 3 (leastloadedF)
    Else
        Goto step 7
    End
End
Personalbest (lbz) <=  0
Globalbest (gb) <=  0
G <= g1.g2.g3……gj
Multiplenodes Bz  <= b1.b2.b3….bz
Multiplenodesize <= j/Bz
```

The steps of Algorithm 2 are given as follows:

1. Begin each particle with its position and speed.
2. Each particle's fitness value is assessed in comparison to its top performance, the personal best (pbest). If the current fitness value is higher, the pbest is upgraded to the new best.
3. Calculate the fitness value of each particle using the fitness function.
4. Assign the particle with the highest fitness value, determined in the previous phase to a new position; it was then randomly positioned with a radius of *r* around it, and the other particles' positions and velocities were adjusted following the fitness function. To determine whether a particle's new position is appropriate, compare it to the particle with the lowest (optimal) fitness value.
5. To find the optimal answer, verify the stated criteria (fitness function); if it is discovered, the algorithm has done iterating; otherwise, return to step 4.

Figure 3 presents the flowchart of the MPSO algorithm.

2.2. Simulation Setup

We have selected a healthcare scenario as a fog computing use case discussed in subsection 2.1 and simulated it using iFogSim. Our work can be considered as a combination of both theoretical and experimental work. In a theoretical sense, we

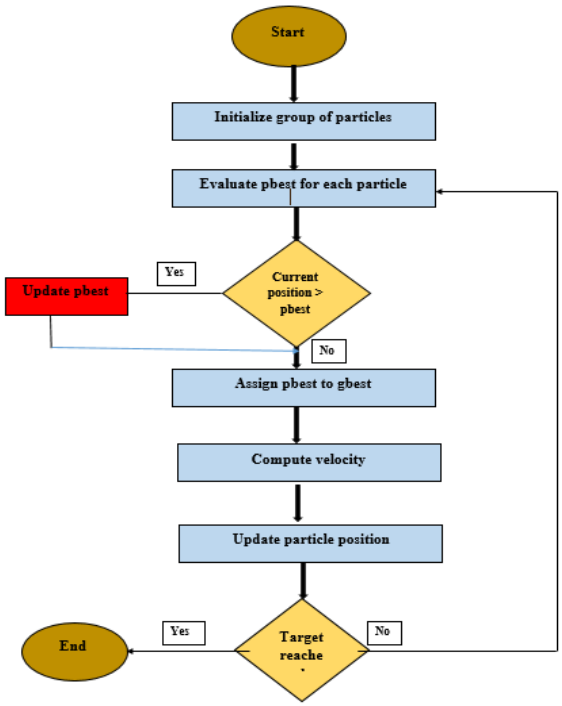

Fig. 3. Flowchart of MPSO.

have simulated a real-world healthcare scenario consisting of three healthcare applications: Appointment Coordination System, Health Record Management System, and Urgent Notification System. In fog computing, it is common practice to compare the resource scheduling based on modified-PSO to base-PSO and First-Come-First-Serve (FCFS) for many reasons. PSO is a nature-inspired optimisation algorithm that finds the optimal solution by moving a population of candidate solutions, known as particles, around the search space. Each particle adjusts its position based on its own and its neighbours' experiences, to arrive at the best solution. Whereas, the FCFS scheduling algorithm is a basic scheduling technique that arranges tasks according to arrival order and always executes the first task. These algorithms serve as baseline scheduling algorithms that help researchers to establish a baseline for performance evaluation that demonstrates the impact and effectiveness of introduced changes, ensuring a thorough evaluation and understanding of their contributions in the context of resource optimization. These comparisons also help to understand the trade-offs between conflicting objectives introduced by the modifications in PSO and whether they strike a better balance across multiple performance criteria such as latency, network usage, cost and energy in a dynamic fog computing environment.

In terms of experimentation, we used iFogsim to compare our proposed resource scheduling algorithm MPSO with traditional (FCFS) and recent meta-heuristic resource scheduling algorithm (PSO) under various conditions and policies. We used real-world data and different scenarios to test our model against real-world performance metrics, showcasing the practical benefits and limitations of modified PSO.

We have used iFogSim [26] to simulate our MPSO-based resource scheduling (task allocation) strategies to allocate the available resources to the task. IFogsim is a powerful toolset for simulating resource management strategies in IoT and fog computing scenarios. For this, we have extended iFogSim with the proposed case study of the healthcare scenario and added some more classes. These classes are used to implement our proposed MPSO-based scheduler. Below is a quick overview of the classes commonly used:

Sensors: The sensor class helps to simulate IoT sensors that generate data tuples with variable lengths and then send them to fog devices.

Fog Device: This class instance imitates different fog nodes with specified memory, computing power, storage space, and uplink and downstream bandwidth.

Tuples: All Fog's entities communicate with each other utilizing tuple class instances. Each tuple has source, destination, and processing demands expressed in MIPS.

Actuator: This class simulates an actuator by defining the outcome of actuation and its network connectivity properties. When a tuple from an application module is received, this classes executes a method that calculates useful metrics.
MPSO System: This class contains the details of fog devices involved in the modified proposed particle swarm algorithm that is in charge of assigning tasks to appropriate fog nodes.

PSO System: This class contains the implementation of the PSO algorithm that employs a group of potential solutions, i.e., particles (known as a swarm). We can use certain equations to shift these particles in the search space.

Scheduler: This class controls the sequence of execution of a list of upcoming tuples by using different algorithms (FCFS, PSO, MPSO).

Particles: The particle class is used for the swarm of particles and looks for the optimal position in the search space. Each element has a specific location $pi = (pi^1, pi^2 \ldots. pi^N)$ and the speed $si = (vi^1, vi^2, \ldots, vi^N)$ in the N-dimensional issue space, the i^{th} particle is represented by I, and N specifies the problem's dimension or the number of unknown variables.

Vector: Vector class is used to determine the specific position and coordinates. To find the location of the particle or node requires the axis and coordinates information. With the help of that, the system can find out the specific location. The vector class, location inside the search space, and best-known position of the entire swarm all serve as cues for the particle motions. These will eventually start

to direct the swarm's motions once better sites are found.

Swarm: The MPSO algorithm's fundamental version uses a swarm class, which employs a population of potential solutions (called particles). These particles are moved around in the search space using a few simple formulae.

Figure 4 displays the tuple emission, MPSO-based task allocation, and execution of tuples. In the first step, the sensor transmits a tuple using the transmit() function. The MPSO-based scheduler class calls the findbestFog() function to choose the best fog node for placement of appropriate application modules on optimal fog nodes. Afterward, the tasks are allocated to these modules for processing. In findbestFog() function the MPSO scheduler uses both the local best module placement and task allocation solution that it has found by using any task allocation strategy and a set of non-dominated module placement and task allocation strategies for multiple objectives to find the overall best fog node to process the task. The objective function was to minimize delay, energy consumption, and cost. When a tuple is received at any fog node, the fog node calls processTupleArrival() function to execute the tuple or forward it to a higher-level fog node. When a tuple is fully executed, the CloudletFinish() method notifies the scheduler to request the execution of the next tuple.

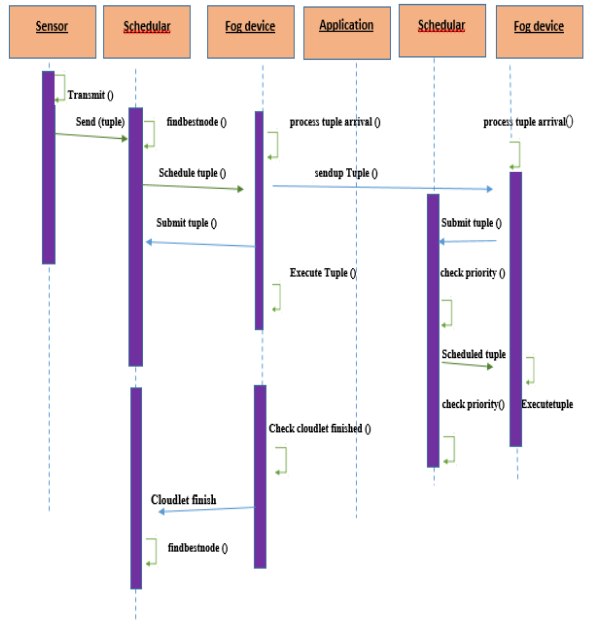

Fig. 4. Sequence diagram of resource scheduling on best node by using MPSO.

2.2.1 Configurations

We have conducted a thorough simulation-based investigation to examine the impact of the modified particle swarm optimization (MPSO) technique. In our simulation, we have used six topologies of 100, 150, 200, 250, 300, and 350 nodes that are arranged in four tiers: sensors and actuators at the bottom-most layer are connected with a set of low-level fog devices that serve as Data Collector and Processor (DCP). These lower-level fog nodes are connected with upper-level fog nodes (that serve as the Coordinator) at the third tier and these high-level fog devices are connected with the Cloud (serving as the patient record database) at the top. We fixed the number of cloud and high-level fog devices for each set of experiments to 1 and 4, respectively. The number of low-level fog generators we used ranged from 25, 50, 75, and 100, resulting in six topologies with 100, 150, 200, 250, 300, and 350 nodes, respectively. Each fog device receives information from sensors that are attached to it and takes appropriate action. For each arrangement, the simulation takes 400 units of time. The configuration of fog devices containing MIPS, Ram, upper-level bandwidth, and down-level bandwidth are given in Table 1.

The details of tuples that are generated by modules and their respective CPU length requirements are mentioned in Table 2.

3. RESULTS AND DISCUSSION

We have compared the performance of MPSO with both traditional and recent meta-heuristic resource scheduling algorithms. We have chosen FCFS from traditional algorithms and PSO from recent meta-

Table 1. Fog nodes configuration.

Device Name	MIPS	Memory	UBW	DBW
Cloud	44800	40000	100	10000
Base station	5600	4000	10000	10000
Upper-level Fog Nodes	5600	4000	10000	10000
Lower-level Fog Node	800	1000	10000	270

Table 2. Tuples with their CPU length (in MIPS).

Tuple type	CPU length
ECG	3000
SENSOR	3500
DATA_REC	1000
DATA_Type	14
RECV_REQ	28
CRITICAL	1000
NON_CRITICAL	1000
DISPLAY	500

heuristic algorithms as used in many recent studies [26-34]. The performance of MPSO algorithms is evaluated by minimizing latency, network usage, cost, and energy consumption.

3.1 Performance Metrics

We have chosen four metrics, latency is calculated using the overall system whole loop as well as for each loop, energy consumption, network utilization, and cost of execution.

i) Average Loop Delay

We have used the term "loop delay" to calculate the end-to-end latency. Loop delay refers to the time required to shift data in multiple modules that are placed on different nodes. The end-to-end latency of each module in the loop is calculated by using a control loop. We have calculated the control loops for different types of tuples transferring data among different modules of the healthcare case study that are explained in subsection 2.1, namely, the Appointment Coordination System, Health Record Management System, and Urgent Notification System.

i. Urgent Notification Loop: DCP->Coordinator->display
ii. Appointment Coordination Loop: DCP-> Coordinator ->DCP->display
iii. Healthcare Record Management Loop: DCP-> Coordinator ->Medical Record Database
iv. History: Coordinator -> Medical Record Database-> Coordinator ->DCP
v. ECG -> MedicalRecordDB-> Coordinator-> MedicalRecordDB -> display

To determine the loop latency, we computed the average processor time T_{cpu}, consumed by tuples of the same type. This average is then calculated using Equation (1).

$$T_{cpu} = \begin{cases} \frac{IT_c \times N + LT_i - IT_i}{N+1}, & \text{if the average CPU time is already computed} \\ LT_i - IT_i, & \text{otherwise} \end{cases} \quad (1)$$

Where IT_i is the initial execution time by all tuples of a specific type of tuple, LT_i is the end execution time of i^{th} tuple, and N is the total number of executed tuples of a certain type. By using Equation (2) we can calculate the execution delay of each tuple.

$$Delay_i = LT_i - IT_i \qquad \forall i \in T \qquad (2)$$

Where T represents the current tuple set.

We have shown along the x-axis number of nodes and y-axis loop delay in Figures 5, 6, 7, 8, 9, and 10 with blue, red, and green bars for FCFS, PSO, and MPSO, respectively. Figure 5 shows the average loop delay for an emergency alert system computed using FCFS, PSO, and MPSO. The loop delay of MPSO and PSO remains almost the same even with an increasing number of nodes but for FCFS it fluctuates.

The urgent notification system needs minimal latency to be efficient, therefore, both PSO and MPSO have placed the urgent notification system at edge devices ingrained in low latency, high priority and local data processing that leads to similar performance.

In Figure 6 the loop delay of the Appointment Coordination System is shown. The results of

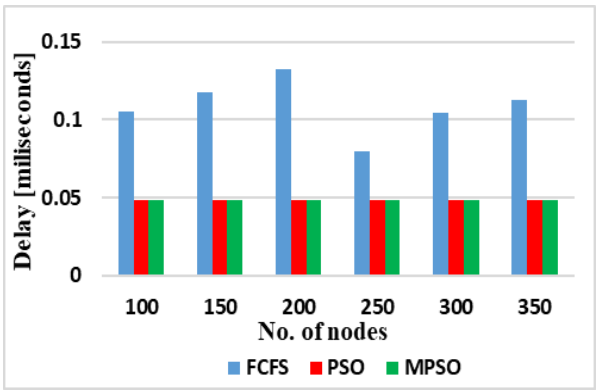

Fig. 5. Loop delay for Urgent Alert.

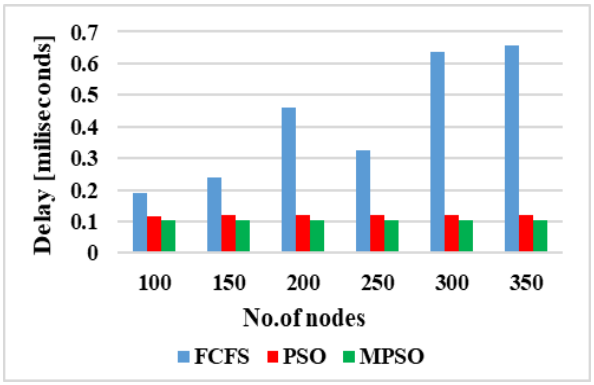

Fig. 6. Loop delay for appointment coordination.

MPSO are better than PSO and FCSF. FCFS has a greater amount of delay starting from a small size and grows as the no. of nodes rises, whereas the PSO and MPSO are the same denoted by bars.

Figure 7 shows the loop delay for medical record management where FCFS performs better than PSO and MPSO but with an increase in the number of nodes from 250 to 350 MPSO shows better results than FCFS and PSO.

In Figure 8 the loop delay of the medical history of patients is plotted, which shows that MPSO causes less delay in processing this record

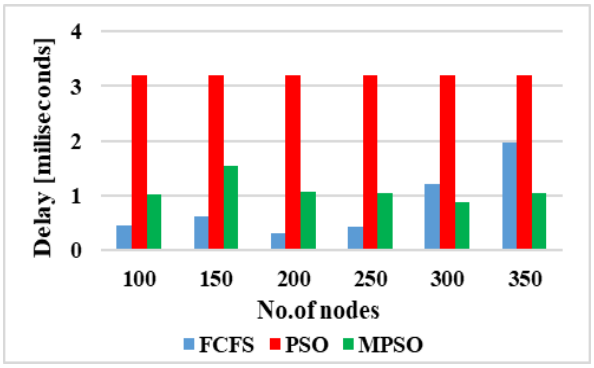

Fig. 7. Loop delay for healthcare record management.

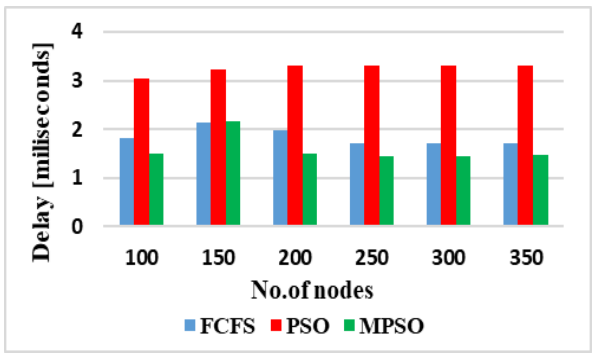

Fig. 8. Loop delay for history.

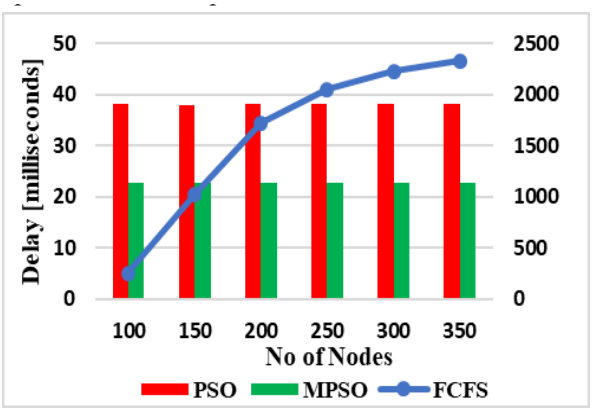

Fig. 9. System whole loop delay.

as compared to PSO and MPSO even with an increasing number of nodes.

ii) System whole loop delay

System whole loop latency is the time delay between the start of the application in a system to the end of the results of the application in a system. Equation (3) shows the formalization of System Latency L_{sys}.

$$L_{sys} = E_T \times N_{tp} (T_f - T_i)/N_{ttp} \quad (3)$$

Where E_T is the total execution time of N^{th} tuples N_{tp}, T_f is the final time of tuple N, N_{ttp} is the total number of tuples, and T_i is a tuple's initial time.

Figure 9 presents the combo plot of the system whole loop delay of FCFS, PSO, and MPSO. MPSO shows less delay as compared to PSO. The blue line represents the loop delay of FCFS which is quite large as compared to the MPSO and PSO.

iii) Energy Consumption

Energy loss is defined as energy supplied to a system that is not immediately consumed by computing processes such as power delivery and conversion, cooling, and lighting. Equation (4) is used to calculate the devices in the system's energy consumption; we can calculate how much energy a Fog device E_{FN} uses.

$$E_{FN} = E_i + (T_{pf} - T_{pi}) \times P_H \quad (4)$$

The energy consumption of any fog node is calculated by the power of all the host nodes in a specific period required for execution. E_i denotes the current consumed energy, T_{pf} is the recent time,

T_{pi} is the update time of last utilization, and P_H is the host power in last utilization.

Figure 10 shows the energy utilized by various fog nodes of the system. Along the x-axis we have plotted the number of fog nodes for all six sets of experiments starting from 100 and ending at 350. Along the y-axis we have plotted the energy consumed by these fog devices.

iv) Network Usage

Network utilization, N_{use}, is the third assessment factor. As the quantity of units expands, so does network consumption, which causes congestion. We compute network usage using Equation (5).

$$N_{use} = \sum_{i=1}^{N} D_i * N_i \quad (5)$$

N signifies the total number of tuples, D_i denotes the delay, and N_i is the size of the i^{th} tuple.

To illustrate how fog devices use the network, this section compares the FCFS algorithm to the MPSO and PSO. The network use of PSO, MPSO, and FCFS is compared using a combo plot in Figure 11. The x-axis represents the number of nodes, and the y-axis represents the average network utilization. The outcome demonstrates that network usage of MPSO is less than PSO and FCFS for all sets of nodes in all experiments.

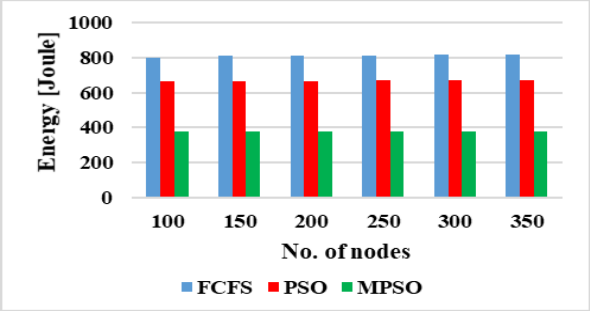

Fig. 10. Energy Consumption.

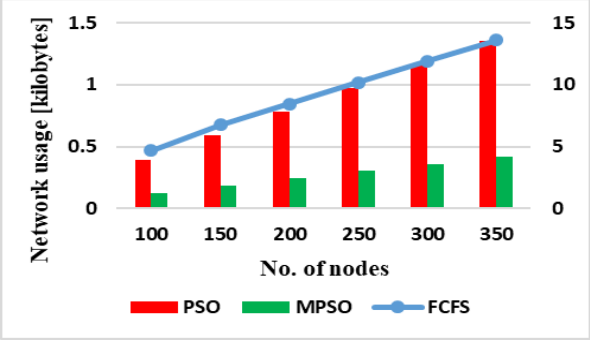

Fig. 11. Network Usage.

v) Cost of Execution

To check the availability and reliability of the proposed module, one of the parameters is the cost of execution. Execution cost can be computed by using Equation (6).

$$C_E = F_C + V_C/NUP \quad (6)$$

C_E is the total cost of execution F_C is for the fixed cost V_C is used instead of variable cost and NUP is for the number of units produced.

Figure 12 shows the execution cost of all six sets of nodes for all experiments. Along x-axis we have taken several nodes and along the y-axis, the cost consumed by various fog nodes in all experiments. The combo graph shows that MPSO has less cost as compared to PSO. The execution cost using FCFS is represented by a blue line that indicates its worst performance.

The outcomes demonstrate that the suggested MPSO outperforms FCFS and PSO in all performance metrics including latency, energy consumption, network usage, and cost of execution. These results reveal the effectiveness and superiority of the proposed MPSO-based resource scheduling as compared to the traditional FCFS and recent meta-heuristic PSO.

4. CONCLUSIONS

Resource scheduling in fog computing aims to maximize resource utilization along different performance metrics, but the heterogeneity of resource-limited fog devices, and the dynamic nature of the fog environment, make resource scheduling a challenging problem. To address this issue, we have proposed a Modified Particle Swarm Optimization (MPSO) based resource scheduling algorithm that

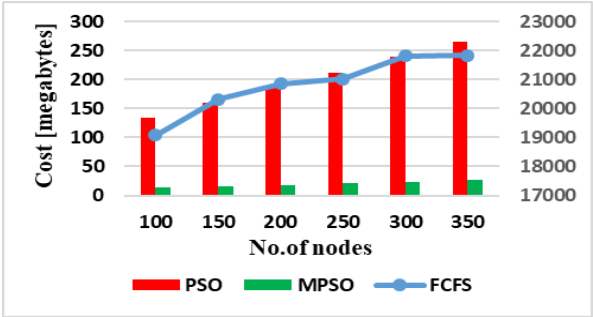

Fig. 12. Cost of Execution.

combines application module placement and task allocation by finding the optimal fog node to place each application module and assign appropriate tasks to the most suitable fog nodes for execution. The MPSO-based resource scheduling aims to maximize the utilization of resources by doing a trade-off of multiple objectives simultaneously, such as minimization of latency, network usage, energy consumption, and execution cost. We have applied our MPSO-based resource scheduling on a healthcare scenario with three healthcare applications namely the Urgent Notification System, Appointment Coordination System, and Health Record Management System as part of our fog computing use case. We then used iFogSim to design, model, and evaluate the performance of these systems under various conditions and scenarios. MPSO outperforms FCFS and PSO in minimizing latency for all healthcare applications including, the Urgent Notification System, Appointment Coordination System, and Health Record Management System. The comparison of results show that MPSO is better suited for all three applications of healthcare scenarios. At the start, the FCFS shows less loop delay than PSO and MPSO, but as the number of nodes increases MPSO shows better results as compared to FCFS and PSO. Furthermore, the analysis of results reveals that MPSO optimizes resource utilization by consuming less energy, low network usage, and reduced cost when compared with FCFS and PSO.

Although the modified PSO improved the performance in dynamic and distributed fog computing environment, but it also results in additional computational complexity and slow convergence. In future, we will address these challenges by applying machine learning algorithm with MPSO to improve its performance.

5. CONFLICT OF INTEREST

The authors declare no conflict of interest.

6. REFERENCES

1. A. Alabdulatif, N.N. Thilakarathne, Z.K. Lawal, K.E. Fahim, and R.Y. Zakari. Internet of nano-things (iont): A comprehensive review from architecture to security and privacy challenges. *Sensors* 23(5): 1–26 (2023).
2. F. Alhaidari, A. Rahman, and R. Zagrouba. Cloud of Things: architecture, applications and challenges. *Journal of Ambient Intelligence and Humanized Computing* 14(5): 5957-5975 (2023).
3. K. Cao, Y. Liu, G. Meng, and Q. Sun. An overview on edge computing research. *IEEE access* 8: 85714–85728 (2020).
4. S.N. Srirama. A decade of research in fog computing: relevance, challenges, and future directions. *Software: Practice and Experience* 54(1): 3-23 (2024).
5. M. Aazam, S. Zeadally, and K.A. Harras. Fog computing architecture, evaluation, and future research directions. *IEEE Communications Magazine* 56(5): 46–52 (2018).
6. B. Jamil, H. Ijaz, M. Shojafar, K. Munir, and R. Buyya. Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions. *ACM Computing Surveys* 54(11s): 233 (2022).
7. M. Ghobaei-Arani, A. Souri, and A.A. Rahmanian. Resource management approaches in fog computing: a comprehensive review. *Journal of Grid Computing* 18(1): 1–42 (2020).
8. M.D. Benedetti, F. Messina, G. Pappalardo, and C. Santoro. JarvSis: a distributed scheduler for IoT applications. *Cluster Computing* 20(2): 1775–1790 (2017).
9. Z. Movahedi, B. Defude, and A.M. Hosseininia. An efficient population-based multi-objective task scheduling approach in fog computing systems. *Journal of Cloud Computing: Advances, Systems and Applications* 10(1): 53 (2021).
10. C.G. Wu, W. Li, L. Wang, and A.Y. Zomaya. An evolutionary fuzzy scheduler for multi-objective resource allocation in fog computing. *Future Generation Computer Systems* 117: 498-509 (2021).
11. B. Jamil, M. Shojafar, I. Ahmed, A. Ullah, K. Munir, and H. Ijaz. A job scheduling algorithm for delay and performance optimization in fog computing. *Concurrency and Computation: Practice and Experience* 32(7): e5581 (2019).
12. S. Jošilo, and G. Dán. Decentralized algorithm for randomized task allocation in fog computing systems. I*EEE/ACM Transactions on Networking* 27(1): 85–97 (2019).
13. H. Zhang, Y. Xiao, S. Bu, D. Niyato, R. Yu, and Z. Han. Computing resource allocation in three-tier IoT fog networks: A joint optimization approach combining Stackelberg game and matching. *IEEE Internet of Things Journal* 4(5): 1204–1215 (2017).
14. H. Zhang, Y. Zhang, Y. Gu, D. Niyato, and Z. Han. A hierarchical game framework for resource management in fog computing. *IEEE Communications Magazine* 55(8): 52–57 (2017).
15. Y. Sun, F. Lin, and H. Xu. Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II. *Wireless*

Personal Communications 102(2): 1369–1385 (2018).
16. S. Bitam, S. Zeadally, and A. Mellouk. Fog computing job scheduling optimization based on bees swarm. *Enterprise Information Systems* 12(4): 373–397 (2018).
17. X. Chen, and L. Wang. Exploring fog computing-based adaptive vehicular data scheduling policies through a compositional formal method - PEPA. *IEEE Communications Letters* 21(4): 745–748 (2017).
18. L.F. Bittencourt, J. Diaz-Montes, R. Buyya, O.F. Rana, and M. Parashar. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing* 4(2): 26–35 (2017).
19. H. Wadhwa, and R. Aron. Optimized task scheduling and preemption for distributed resource management in fog-assisted IoT environment. *The Journal of Supercomputing* 79(2): 2212-2250 (2023).
20. J. Du, L. Zhao, J. Feng, and X. Chu. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. *IEEE Transactions on Communications* 66(4): 1594–1608 (2018).
21. F.R. Shahidani, A. Ghasemi, A.T. Haghighat, and A. Keshavarzi. Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm. *Computing* 105(6): 1337-1359 (2023).
22. S. Subbaraj, R. Thiyagarajan, and M. Rengaraj. A smart fog computing based real-time secure resource allocation and scheduling strategy using multi-objective crow search algorithm. *Journal of Ambient Intelligence and Humanized Computing* 14: 1003–1015 (2023).
23. W. Liu, C. Li, A. Zheng, Z. Zheng, Z. Zhang, and Y. Xiao. Fog Computing Resource-Scheduling Strategy in IoT Based on Artificial Bee Colony Algorithm. *Electronics* 12(7): 1511 (2023).
24. M. Aldossary. Multi-layer fog-cloud architecture for optimizing the placement of IoT applications in smart cities. *Computers, Materials & Continua* 75(1): 633-649 (2023).
25. D. Tian, and Z. Shi. MPSO: Modified particle swarm optimization and its applications. *Swarm and Evolutionary Computation* 41: 49-68 (2018).
26. H. Gupta, V.D. Amir, K.G. Soumya, and R. Buyya. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience* 47(9): 1275-1296 (2017).
27. R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization: An overview. *Swarm Intelligence* 1: 33-57 (2007).
28. S.S. Hajam, and S.A. Sofi. Resource management in fog computing using greedy and semi-greedy spider monkey optimization. *Soft Computing* 27(24): 18697-18707 (2023).
29. N. Potu, C. Jatoth, and P. Parvataneni. Optimizing resource scheduling based on extended particle swarm optimization in fog computing environments. *Concurrency and Computation: Practice and Experience* 33(23): e6163 (2021).
30. C. Huang, H. Wang, L. Zeng, and T. Liu. Resource scheduling and energy consumption optimization based on Lyapunov optimization in fog computing. *Sensors* 22(9): 3572 (2022).
31. M. Fahad, M. Shojafar, M. Abbas, I. Ahmed, and H. Ijaz. A multi-queue priority-based task scheduling algorithm in fog computing environment. *Concurrency and Computation: Practice and Experience* 34(28): e7376 (2022).
32. S. Javanmardi, M. Shojafar, V. Persico, and A. Pescapè. FPFTS: A joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for Internet of Things devices. *Software: Practice and Experience* 51(12): 2519-2539 (2021).
33. U.K. Saba, S.ul. Islam, H. Ijaz, J.J. Rodrigues, A. Gani, and K. Munir. Planning Fog networks for time-critical IoT requests. *Computer Communications* 172(C): 75-83 (2021).
34. H. Rafique, M.A. Shah, S.U. Islam, T. Maqsood, S. Khan, and C. Maple. A novel bio-inspired hybrid algorithm (NBIHA) for efficient resource management in fog computing. *IEEE Access* 7: 115760-115773 (2019).