Received February 6, 2020, accepted February 17, 2020, date of publication February 21, 2020, date of current version March 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2975741

Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization

MOHAMED K. HUSSEIN[1] **AND MOHAMED H. MOUSA**[1,2]
[1]Faculty of Computers and Informatics, Suez Canal University, Ismailia 41522, Egypt
[2]Department of Information Technology, College of Computer Science, University of Jeddah, Jeddah 25651, Saudi Arabia

Corresponding author: Mohamed K. Hussein (m_khamiss@ci.suez.edu.eg)

ABSTRACT The current thinking concerning computations required by Internet of Things (IoT) applications is shifting toward fog computing instead of cloud computing, thereby achieving most of the required computations at the network edge of the IoT devices. Fog computing can thus improve the quality of service of delay-sensitive applications by allowing such applications to take advantage of the low latency provided by fog computing rather than the high latency of the cloud. Therefore, tasks in various IoT applications must be effectively distributed over the fog nodes to improve the quality of service, specifically the task response time. In this paper, two nature-inspired meta-heuristic schedulers, namely ant colony optimization (ACO) and particle swarm optimization (PSO), are used to propose two different scheduling algorithms to effectively load balance IoT tasks over the fog nodes under communication cost and response time considerations. The experimental results of the proposed algorithms are compared with those of the round robin (RR) algorithm. The evaluations show that the proposed ACO-based scheduler achieves an improvement in the response times of IoT applications compared to the proposed PSO-based and RR algorithms and effectively load balances the fog nodes.

INDEX TERMS Fog computing, Internet of Things, quality of service, task offloading and scheduling.

I. INTRODUCTION

The Internet of Things (IoT) will dominate urban cities and transform overcrowded cities into smart cities. The aim of a smart city is to use cutting-edge IT technology to improve the quality of services offered to residents. To achieve the goal of the smart city, the IoT consists of a large number of geographically distributed nodes. Each node contains a number of smart sensors, where each sensor senses a physical parameter of the surrounding environment within its transmission range. An aggregation and analysis are performed on the collected data to generate decisions and coordinated actions to be performed. IoT application areas are diverse and include smart homes, health care, traffic control, smart transportation, video surveillance and emergency response [1].

Cloud computing is used as a backend layer for data storage and data analysis because the cloud can provide enormous storage and processing capabilities that are otherwise not available to the IoT devices. This is called task offloading, where the workload, the large amount of data that the IoT devices generate, is transferred to the cloud for processing [2]–[4]. However, it is inefficient to offload tasks to the cloud layer because this will cause network bandwidth overhead, as much of the data can be filtered due to high redundancy. Furthermore, IoT task offloading to the cloud delays the response time of the data analysis as a result of the relatively high network latency. Therefore, the fog computing concept was developed to bring computing capabilities closer to the IoT devices [5].

IoT task offloading to fog computing instead of the cloud avoids high network congestion and reduces the data analysis response time by taking advantage of low network latency. This concept is also called edge computing because it provides computing capabilities, including better response times and low latencies, at the edge of the network near the IoT devices. Yet, there is a distinction between fog computing and edge computing. Fog computing provides computing, networking, and storage services to IoT devices using network devices, such as routers and gateways, within the LAN of the IoT devices. On the other hand, edge computing provides

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Marozzo.

computing and storage services using small data centers close to the IoT devices via WiFi access points [6].

Fog computing provides the quality of service (QoS) required by certain smart applications such as traffic control, health care applications, and autonomous driving. However, it is important to effectively select the proper fog nodes for task offloading and load balance the tasks over the fog nodes while satisfying the required QoS requirements of the tasks, including the response times of the tasks.

The aim of this research is to satisfy the QoS requirements of IoT delay sensitive applications by offloading tasks on the data generated by IoT sensor nodes onto the fog nodes, therein considering the communication costs and the existing load on the fog nodes. Furthermore, it is essential to effectively load balance the tasks over the fog nodes to improve the QoS of IoT applications, specifically the response time, and to improve the utilization of the fog nodes. However, this problem is NP-hard problem where the difficulty increases exponentially as the number of sensors and fog nodes increases. Therefore, it is challenging to use the traditional greedy search methods [7]. To overcome this problem and to achieve the goals of this research, two evolutionary meta-heuristic task offloading scheduler, namely, ant colony optimization (ACO) and particle swarm optimization (PSO), are proposed. The experimental evaluation results show that the proposed ACO-based scheduler achieves an improvement compared to the proposed PSO-based scheduler and the round robin (RR) algorithm in terms of the IoT application response time and effective use of fog nodes.

The remainder of this paper is organized as follows. Section II discusses related work. Section III outlines a formal model of the research problem, while Section IV presents the proposed meta-heuristic algorithms for solving the problem. Section V describes the experimental results and the evaluation of the proposed approach. Finally, Section VI concludes the paper with outlines for future research.

II. BACKGROUND AND RELATED WORK

Task offloading is the process of delegating computations from devices with low computational capacity and power, such as IoT devices, to remote powerful servers to satisfy certain QoS requirements [8], [9]. Sending the data generated by IoT devices to the cloud for aggregation and analysis produces network overhead and delays the response times. Hence, fog computing technology moves computations to the edge of the network near the IoT devices. This enables improvements in the response times by taking advantage of the low latency. This is particularly important for real-time applications such as video streaming, intelligent transportation, and autonomous vehicles [1].

There have been a number of studies on task offloading to the edge of the IoT network. For example, the research in [10] proposes a framework for task offloading to minimize the service delay of IoT-fog-cloud applications. The framework uses a delay-minimizing policy that considers the loads on the fog nodes and the type of workloads generated by the IoT devices. If the calculated service delay based on the current load on the fog node is less than a threshold, the task will be accepted at the fog node; otherwise, the task will be forwarded to the best neighboring fog node. The best neighboring node is calculated using the expected service delay and the propagation delay of all neighbor nodes. Finally, if the task reaches the maximum amount of offloading, the task will be offloaded to the cloud. However, this work assumed that the fog nodes are directly connected to each other; hence, the delay-minimizing policy does not consider the communication cost.

In [11], the researchers proposed a heuristic based on linear programming for task distribution of medical cyberphysical systems. The proposed algorithm considers the deployment of virtual machines for virtual medical devices and provides the required QoS of real-time medical applications by considering the communication cost, computation cost, virtual machine placement, and task distribution cost. However, the proposed algorithm is based on a cellular network architecture where the fog nodes are represented as cellular network base stations. Moreover, no actual task offloading or load balancing capabilities are considered, only virtual machine placement. Hence, the method cannot be generalized to a fog computing architecture.

In [12], an analysis of the workload allocation between the fog nodes and the cloud nodes is investigated to minimize the power consumption under service delay constraints. However, this analysis assumed that the fog nodes are connected to the cloud nodes through a single point of communication. Further, the study focused on the cooperation between the fog nodes and the cloud computing servers. In our study, we focus on the task offloading between the IoT devices and the fog nodes.

In [13], a task offloading and management between the embedded devices and the fog nodes is proposed. The aim is to minimize the computation time of software-defined embedded systems. The proposed strategy uses mixed-integer non-linear programming joint optimization of task scheduling and storage placement.

In [14], a simple task offloading architecture for information-centric IoT applications is developed based on task classification and a simple cost function. The proposed architecture does not consider the communication cost. In [15], a heuristic algorithm is proposed to determine task offloading placement on the fog nodes based on a cost function that considers the communication cost, computation cost, and power consumption.

In [16], an IoT-mobile edge computing task offloading service orchestration scheme is proposed. The service orchestration scheme uses software defined networking (SDN) technology to reduce the network transmission load. Further, a heuristic algorithm for the differentiated cloud-edge offloading decisions is proposed using an optimization function based on communication energy consumption, computation energy consumption, task delay models. Also, in [17] the SDN scheme is proposed to reduce the service response delay and data redundancy in the cloud-edge layer to meet the requirement of the network.

In [18], [19], a cloud-fog-based architecture is proposed for a resource management in a smart grid (SG) scenario. Different algorithms are used for scheduling requests from smart devices, such as smart meters, on the virtual machines of the fog nodes, including round robin, throttled, particle swarm optimization (PSO), ant colony optimization (ACO), and a hybrid artificial bee colony and ACO algorithm that outperformed the other algorithms. However, the focus of the research is mainly on the cloud-fog architecture of a smart grid scenario, and there is neither a clear description of the proposed algorithms nor a load balancing mechanism. Similar research is conducted in [20] where the ACO algorithm is used for scheduling IoT tasks on the fog nodes in a SG. The proposed algorithm focused on minimizing the response time of the IoT tasks. However, the proposed ACO algorithm depends on profiling the IoT tasks. In [21], a graph-based heuristic is proposed to load balance tasks over the fog nodes. However, the proposed heuristic is impractical because of the dynamic nature of the loads on the fog nodes. In [22], a bee life algorithm is proposed to schedule IoT tasks on the fog nodes with the aim of achieving a trade-off between computation time and memory. In [23], a genetic algorithm is proposed to schedule IoT tasks on the fog nodes with the aim of achieving a trade-off between computation time and the operational cost. A similar work is proposed in [24] using a PSO algorithm. However, none of these studies considers the communication cost.

A similar research problem considers the optimal placement of fog devices over a specified large area for the optimization of fog node power consumption while satisfying the QoS requirements of IoT-based applications. For example, in [25], genetic algorithms are used to find a trade-off between fog node placement, power consumption and the QoS for task offloading. The study in [26] investigates the edge server placement problem, where PSO based on a multi-objective function is used to reduce the total energy consumption and maintain an acceptable access delay. In [7], a hybrid genetic-simulated annealing latency-minimum offloading decision algorithm in IoT-fog computing is designed to find the best offloading decision with minimum latency. Also, in [27], a genetic algorithm is proposed for the offloading decisions in IoT-fog computing. However, these algorithms do not consider the balancing the load balancing on the fog nodes.

In this paper, we investigate the problem of IoT task offloading to the fog with the main goal of reducing the IoT-based application response time by load balancing the tasks on the fog nodes while considering the communication cost, computation time and the existing load conditions on the fog nodes.

III. IoT-FOG SYSTEM MODEL

This paper investigates an efficient task offloading algorithm in the fog computing infrastructure using the ACO meta-heuristic. The main goal is to minimize task response times while considering the network bandwidth, network latency, and existing load on the fog devices.

TABLE 1. Main notation and descriptions.

Symbol	Meaning
S_i	Sensor number i
λ_i	Data rate produced by sensor S_i
c	Application class
fg_j	Fog node number j
λ_j	Data arrival rate at fog node fg_j
λ_{jc}	Data arrival rate at fog node fg_j from all sensors belonging to application class c
L_{ij}	Network latency between S_i and fg_j
$Dsize_i$	Data size generated from sensor S_i
BW_l	Local network bandwidth
$M_{sensors}$	Total number of sensors
N_{nodes}	Total number of fog nodes
L_{jc}	Cloud latency between fg_j and the cloud
BW_c	Cloud network bandwidth
μ_{ij}	Service rate of fg_j for data produced by S_i
μ_{jc}	Service rate of fg_j for application class c

A. IoT-FOG NETWORK ARCHITECTURE

The IoT-fog network architecture used in this paper consists of three layers, the IoT layer, the fog layer, and the cloud layer, as shown in Figure 1. The same architecture is used in [7], [27]. In the bottom layer, the IoT layer, a number of geographically distributed sensor nodes are connected using a local network. The IoT layer collects different data from different applications such as weather, air quality, and traffic intensity [28]. The IoT layer is connected to a fog layer that contains a number of fog nodes. Each fog node is used for aggregating, filtering, and light processing of the collected sensor data. Each fog node has a local agent that is responsible for collecting performance data, such as sensor data arrival rate and sensor service rate. The sensors send the task offloading requests to the fog nodes, and the fog nodes forward the requests to a master fog node that is responsible for scheduling the offloaded tasks to the fog nodes using the collected data from the fog agents. Finally, the cloud layer provides powerful computing and storage capabilities. The fog layer is connected to the cloud layer through the Internet rather than the local network. The cloud layer is used for heavy processing and data storage.

B. IoT-FOG MODEL FORMULATION

The IoT layer consists of a number of distributed sensors $S_1, S_2, \ldots, S_m$, where each sensor S_i produces data at rate λ_i. The fog layer consists of a number of fog nodes $fg_1, fg_2, \ldots, fg_n$. Table 1 presents the notation used in this section.

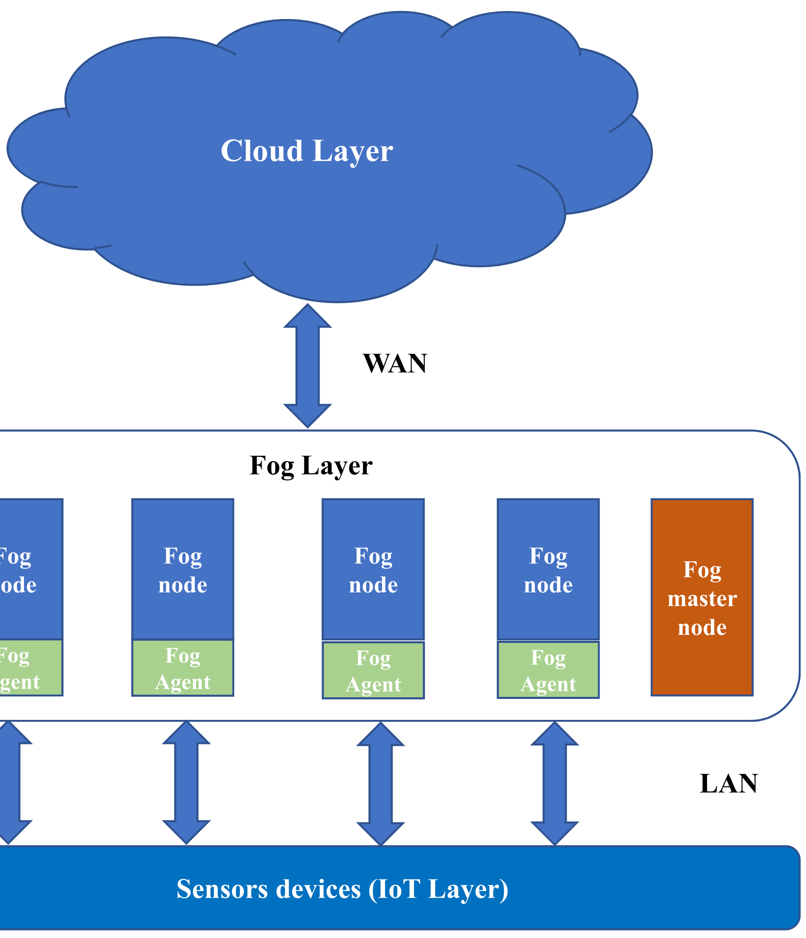

FIGURE 1. Layered fog computing infrastructure.

The data communication cost between the sensor S_i and the fog node fg_j, as shown in Equation (1), is the sum of the network latency L_{ij} between sensor S_i and fog node fg_j and the data transfer time between S_i and fg_j, which is calculated as $Dsize_i/BW_l$, where $Dsize_i$ is the size of the data generated by sensor S_i and BW_l is the local network bandwidth.

$$FogCCost_{ij} = L_{ij} + \frac{Dsize_i}{BW_l} \tag{1}$$

The communication cost between fog node fg_j and the cloud, as shown in Equation (2), is the sum of the cloud latency L_{jc} and the data transfer time between fog node fg_j and the cloud is calculated as $Dsize_i/BW_c$, where BW_c is the cloud network bandwidth.

$$CloudCCost_{ij} = L_{jc} + \frac{Dsize_i}{BW_c} \tag{2}$$

The total communication cost for sensor S_i offloading is calculated using Equation (3).

$$CommCost_{ij} = CloudCCost_{ij} + FogCCost_{ij} \tag{3}$$

Each sensor, S_i, generates data following a Poisson distribution with a rate λ_i. The generated data are sent to the fog nodes to perform filtering, aggregation, and simple analysis. Finally, the fog nodes send the data to the cloud where heavier analysis, expensive processing, and storage are performed.

The service time of task offloading of sensor S_i to fog nodes fg_j is calculated according to the $M/M/1$ queuing model using Equation (4).

$$ST_{ij} = \frac{1}{\mu_{ij} - \lambda_i} \tag{4}$$

where μ_{ij} is the service rate and λ_i is the data arrival rate from sensor S_i.

For a large number of sensors, where each sensor belongs to a specific application class c, the service time is calculated using Equation (5).

$$ST_{jc} = \frac{1}{\mu_{jc} - \lambda_{jc}} \tag{5}$$

where μ_{jc} is the service rate at fog node fg_j for a specific class of application c and λ_{jc} is the total arrival rate of data from sensors belonging to a certain application c, which is calculated using Equation (6).

$$\lambda_{jc} = \sum_{i \in c} \lambda_{ji} \tag{6}$$

The utilization of fog node fg_j, U_j, is the sum of the utilizations of offloaded tasks from all sensors S_i, as calculated using Equation (7).

$$U_j = \sum_c U_{jc} = \sum_c \sum_{i \in c} x_{ij} \times \frac{\lambda_{ji}}{\mu_{ij}} \tag{7}$$

where $x_{ij} = 1$ if sensor S_i tasks are offloaded to fog node fg_j.

The overall response time of the sensor S_i workload as a result of task offloading is the sum of the IoT-fog communication cost, fog-cloud communication cost and average service time on the fog nodes, which is calculated using Equation (8).

$$\begin{aligned} R_{ij} &= CommCost_{ij} + ST_{jc} \\ &= L_{jc} + \frac{Dsize_i}{BW_c} + L_{ij} + \frac{Dsize_i}{BW_l} + \frac{1}{\mu_{jc} - \sum_{i \in c} \lambda_{ji}} \end{aligned} \tag{8}$$

The average response time of fog node fg_j as a result of all tasks offloaded to that node is calculated using Equation (9).

$$R_j = \sum_c \frac{R_{jc}}{1 - U_{jc}} \tag{9}$$

The load $load_j$ on fog node fg_j relative to all fog nodes is calculated using Equation (10).

$$load_j = 1 - \frac{R_j - R_{average}}{\sum_j R_j} \tag{10}$$

IV. EVOLUTIONARY ALGORITHMS FOR IoT-FOG TASK OFFLOADING

The IoT-fog task offloading problem is critical for real-time sensitive IoT applications. The data generated by the IoT devices must be balanced across the fog nodes while considering network latency, network bandwidth, processing times of the fog nodes, and the existing load on the fog nodes. Therefore, the task offloading solution must choose the fog device target that is guaranteed to satisfy the defined QoS constraints, specifically the response time, considering the network characteristics and the current load on the fog nodes. This problem is NP-hard problem where the difficulty increases exponentially as the number of sensors and fog nodes increases. Therefore, it is challenging to use the traditional greedy search methods. We propose two evolutionary meta-heuristics algorithms, using the ant colony optimization and particle swarm optimization, to over come this challenge [7]. This section presents the proposed evolutionary algorithms, namely ACO and PSO, for handling the optimization problem of this research.

A. THE PROPOSED ACO TASK OFFLOADING ALGORITHM

ACO is a probabilistic meta-heuristic method inspired by ants' capabilities to find the shortest path between their colony and a food source. Such collective foraging behavior for food sources by actual ants is used to solve several optimization problems such as finding the shortest path in a graph [29]. Initially, an ant leaves its colony and moves along a randomly selected path to search the surrounding neighborhoods for food sources. Ants indirectly communicate with each other through the release of a chemical signal, called pheromones. On the way back to the colony, the ant releases an amount of pheromone along the path proportional to the quantity and quality of the found food. Subsequently, other ants will have high probability of following paths that have high pheromone concentrations. Eventually, all ants will follow the shortest best path for the food-home journey [30], [31]. For m ants and n possible paths, each ant selects a possible path according to the highest concentration of pheromones on all possible paths.

The ACO algorithm has been proven effective as an optimization meta-heuristic for several NP-hard research problems, including the traveling salesman and job shop scheduling problems [32]. Furthermore, the ACO algorithm has been used in similar scheduling problems in the cloud, including scheduling virtual machines on cloud resources [33], [34] and scheduling tasks on virtual machines with the goal of load balancing the tasks on the virtual machines and reducing the response time of the tasks [35], [36]. Moreover, the ACO algorithm is used for scheduling IoT tasks on the cloud [37]. Additionally, the ACO algorithm has been used for deadline-aware task scheduling for fog computing in a tiered IoT infrastructure [38]. The proposed algorithm focuses on maximizing the profits of a fog service provider while considering the completion deadline constraints of the IoT tasks.

In a search for task offloading of the workload of sensor S_i onto fog nodes fg_j with the objective of minimizing the response time, the k^{th} ant chooses fog node fg_j for task offloading of the workload generated by sensor S_i with the probability given by Equation (11).

$$P_{ij}^k(t) = \frac{(\tau_{ij}(t))^\alpha (\eta_{ij}(t))^\beta}{\sum_s (\tau_{is}(t))^\alpha (\eta_{is}(t))^\beta} \tag{11}$$

where α and β are heuristic constants; $\alpha \geq 0$ is a heuristic parameter that controls the effect of the pheromone quantity and $\beta \geq 1$ is a heuristic parameter that determines the importance of the task offloading quality. $\eta_{ij}(t)$ is a heuristic function that represents the quality of task offloading and is calculated using Equation (12).

$$\eta_{ij}(t) = \frac{load_j}{R_{ij}} \tag{12}$$

where R_{ij} is calculated using Equation (8) and $load_j$ represents the load on the fog node j and is calculated using Equation (10). Clearly, as R_j increases, the $load_j$ decreases and $\eta_{ij}(t)$ decreases. As a result, the probability of offloading the computations of sensor i to fog node fg_j will be small.

$\tau_{ij}^k(t)$ represents the pheromone trail quantity for task offloading for an ant k in iteration (t), and $\tau_{ij}^k(t+1)$ is the pheromone trail of that task offloading for an ant k in iteration

$t + 1$, determined using Equation (13).

$$\tau_{ij}^{k}(t+1) = (1-\rho)\tau_{ij}^{k}(t) + \rho\Delta\tau_{ij}^{k}(t) \tag{13}$$

where $\Delta\tau_{ij}^{k}(t) = 1/R_{ij}$ and ρ is a constant that represents the rate of pheromone evaporation, which simulates the effect of the evaporation of the pheromones at each step.

Additionally, the pheromone trail is globally updated after all ants have completed their possible task offloading, i.e., a complete iteration. The global pheromone update is performed using Equation (14).

$$\tau_{ij}^{k}(t+1) = (1-\rho_g)\tau_{ij}^{k}(t) + \rho_g\Delta\tau_{ij}^{k}(t) \tag{14}$$

where $\Delta\tau_{ij}^{k}(t) = 1/L_{best}$, L_{best} is the best floading task discovered, and ρ_g is a global evaporation rate.

The initial value of the pheromone trail is calculated using Equation (15).

$$\tau_{ij}^{k}(0) = \frac{R_{ij}}{R_{average}} \tag{15}$$

Algorithm 1 The Proposed ACO Task Offloading Algorithm

```
1: Initialize the parameters
   $\alpha, \beta, \rho, \rho_g, M_{ants}, N_{iter}, N_{nodes}, N_k\lambda_i, \mu_{ij}$
2: Calculate $R_{ij}$ using Eq. (8)
3: Initialize the pheromone matrix $\tau_{ij}^{k}(0)$
4: Calculate $\eta_{ij}^{k}(t)$ using Eq. (15)
5: while $iter \leq N_{iter}$ do
6:     Place all ants at the starting nodes randomly
7:     for $ant_k = 1$ to $N_k$ do
8:         for $S_i = 1$ to $N_nodes$ do
9:             Calculate $P_{ij}^{k}(t)$ using Eq. (11)
10:            $ant_k$ chooses $fg_j$ for $S_i$ using the roulette method
11:            Add the selected node to the taboo table of $ant_k$
12:        end for
13:        Update $\tau_{ij}^{k}(t+1)$ using Eq. (13)
14:    end for
15:    Compare with the previous best solution and update
       the best solution
16:    if current solution is the best then
17:        Update $\tau_{ij}^{k}(t+1)$ using Eq. (14)
18:    end if
       $iter = iter + 1$
19: end while
20: Clear taboo table
```

Algorithm 1 shows the proposed meta-heuristic algorithm using ACO to find an efficient task offloading for the IoT sensors on the available fog nodes that guarantees that the defined QoS constraints, specifically the response time, are satisfied while considering the network characteristics, the service time and the current load on the fog nodes. The proposed algorithm starts by setting the values of the heuristic parameters α, β, ρ, ρ_g, the number of ants M_{ants} and setting the maximum number of iterations N_{iter}. Additionally, the algorithm initializes the number of sensor nodes N_{nodes}, the workload production rate of each sensor λ_i, and the fog node service rate μ_{ij}.

Step 2 calculates R_{ij} the response time of fog node fg_j using Equation (8). Steps 3 and 4 initialize the initial pheromone trail $\tau_{ij}^{k}(0)$ and the task offloading heuristic $\eta_{ij}^{k}(t)$ for each ant ant_k using Equations (12) and (15). During each iteration, each ant ant_k offloads the workload computation of sensor S_i to the fog node fg_j with probability $P_{ij}^{k}(t)$ calculated using Equation (11). The selection of fog node fg_j is performed using the roulette wheel method [39]. The roulette wheel method calculates the probability for each solution, and the cumulative probabilities are arranged in ascending order. A random number $\in [0, 1]$ is generated and compared against the calculated cumulative probabilities. Finally, the appropriate solution that corresponds to the generated random number is selected. This method has been proven effective in selecting potentially useful solutions and is frequently used in genetic algorithms [40].

When an ant finishes all the task offloading of all the sensors, the local pheromone trail matrix is updated using Equation 13. After each iteration, the ACO algorithm updates the global pheromone trail matrix $\tau_{ij}^{k}(t)$ using Equation (14). The algorithm iterates until the maximum number of iterations is reached.

B. THE PROPOSED PSO TASK OFFLOADING ALGORITHM

PSO is a meta-heuristic optimization algorithm that is based on a cooperating population of individuals, called a swarm, of size N_{swarm}. Each individual in the swarm, called a particle, represents a solution that is a position in the search space. A particle P_i in a D-dimensional search space is expressed as a D-dimensional vector $P_i = \{p_{i1}, p_{i2}, \ldots, p_{id}\}$. The particles search randomly for a feasible solution to a given problem. The search movement of each particle depends on the local optimal of its historical search and the found global optimal of all particles [41]. During the search process, each particle, P_i, updates its new position, $X_i(t+1)$, based on calculating the velocity, $V_i(t+1)$, as shown in Equations (16) and (17).

$$V_i(t+1) = \omega \times V_i(t) + C_1 \times \alpha \times (Pl_i - P_i) + C_2 \times \beta \times (Pg - P_i(t)) \tag{16}$$

$$P_i(t+1) = P_i(t) + V_i(t+1) \tag{17}$$

where $i = 1, 2, \ldots, N_{swarm}$ and $t = 1, 2, \ldots, iter_{max}$ is the iteration number. ω is the initial velocity inertia weight, which balances exploration and exploitation. If a large inertia weight value is used at the beginning of the search process, exploration is favored, whereas a smaller inertia weight value facilitates more exploitation. C_1 and C_1 are learning coefficients. α and β are random numbers uniformly distributed $\in [0, 1]$. PSO was initially proposed for solving problems in continuous domains. Since the scheduling of mobile task offloading occurs in a discrete search space, PSO must be modified to suit domain [42].

1) DISCERETE PSO ALGORITHM

The position of particle p_{ij} represents the offloading of sensor node S_i to fog node fg_j. The representations of the offloaded sensor tasks and fog nodes are discrete. Figure 2 shows an example of a particle encoding. In the example, the task of sensor node S_1 is offloaded to fog node fg_3, that of S_2 is offloaded to fg_1, that of S_3 is offloaded to fg_2, that of S_4 is offloaded to fg_2, and that of S_5 is offloaded to fg_1.

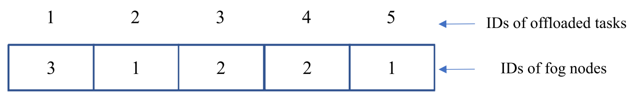

FIGURE 2. An example of a particle encoding.

During each iteration, the particle position and velocity are updated using Equations (16) and (17). However, each particle position value must be converted into a discrete numerical value using Equation (18).

$$p_{ij} = \begin{cases} \lfloor |p_{ij}| \rfloor & \text{if } 0 > p_{ij} \leq N_{nodes} \\ \lfloor |p_{ij}| \rfloor \% N_{nodes} & \text{otherwise} \end{cases} \quad (18)$$

where N_{nodes} is the number of fog nodes and $\lfloor |p_{ij}| \rfloor$ is the ceiling of the absolute value of p_{ij} for particle p_j.

The overall proposed PSO is shown in Algorithm 2. The algorithm starts by initializing the maximum number of iterations N_{iter}, the velocity update parameters, including $\alpha, \beta, and \omega$, and the mutation probability $P_{mutation}$. Further, the proposed PSO algorithm initializes the particle positions P_i and velocities V_i to random values. Each particle is evaluated through a fitness function that represents the quality of the solution that the particle encoding expresses. the fitness value is calculated using Equation (19).

$$Maximize f = \sum_{j=1}^{N_{nodes}} \frac{load_j}{R_j} \quad (19)$$

The higher the value of the fitness function is, the better the solution. After calculating the fitness of each particle, the local optimum of each particle is set. The best fitness among the swarm is set as the global optimum. The algorithm iterates for a defined number of iterations N_{iter}. During each iteration, each particle velocity and position is updated using Equations (16) and (17), and each particle position is converted to a discrete value using Equation (18). The PSO algorithm maintains the new local and global optima. Finally, a random mutation is applied to each particle. In the mutation process, the values of two random nodes of a particle are interchanged to generate a new task offloading of the particles, as shown in Figure 3. The mutation improves the local search capability and maintains the diversity of the generated new task offloading decision.

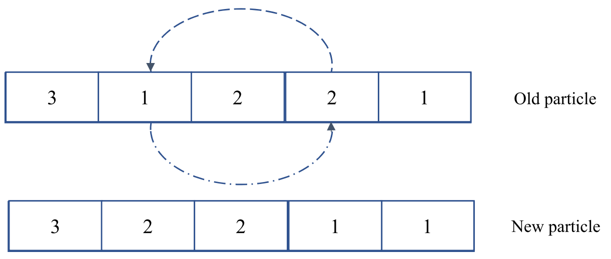

FIGURE 3. An example of particle mutation.

Algorithm 2 The Proposed Discrete PSO Algorithm

```
1:  Set the parameters $N_{iter}, N_{Particles}, \alpha, \beta, \omega, and P_{mutation}$
2:  Initialize each particle position $P_i$ with random task
    assignments
3:  Initialize each velocity $V_i$ with random values
4:  Calculate the fitness function for particle $P_i$ using
    Eq. (19)
5:  Set the local optimum $Pl_i$, best behavioral solution, for
    each particle $P_i$
6:  Set the global optimum $P_g$, best social solution for the
    swarm
7:  **for** each $iter$ in $N_{iter}$ **do**
8:      **for** each particle $P_i$ in $N_{Particles}$ **do**
9:          Calculate $V_i(t)$ for each particle using Eq. (16)
10:         Calculate $P_i(t)$ for particle $P_i$ using Eq. (17)
11:         Convert $P_i(t)$ to discrete values using Eq. (18)
12:         Calculate the fitness function for particle $P_i$ using
            Eq. (19)
13:         **if** current solution is the best among the previously
            explored solutions of $P_i$ **then**
14:             Update the local optimum $Pl_i$
15:         **end if**
16:         **if** $Pl_i < Pg$ **then**
17:             Update the global optimum $Pg = Pl_i$
18:         **end if**
19:         Apply the mutation operator on particle $P_i$ as fol-
            lows:
20:         **for** each $node_j$ in the particle $P_i$ **do**
21:             **if** $rand() < P_{mutation}$ **then**
22:                 $node_k$ = random index
23:                 $Mutation(node_k, node_j)$
24:             **end if**
25:         **end for**
26:     **end for**
27: **end for**
```

V. EXPERIMENTAL EVALUATION

This section presents the experiments conducted to evaluate the proposed ACO task offloading algorithm for IoT-fog-cloud environments. A simulation is designed and developed using MATLAB to evaluate the proposed ACO task offloading algorithm. A simulation-based evaluation is adopted because it allows the environment parameters to be controlled and for the experiments to be repeated under different scenarios and constraints. The following subsections discuss the test scenario implemented in the simulation, the parameter setup, the evaluation criterion, and the evaluation of the proposed ACO-based algorithm for effective task offloading of IoT-fog-cloud environments against the RR algorithm.

A. IoT-FOG TEST-CASE ARCHITECTURE

An IoT test-case scenario is used to test the effectiveness of the proposed algorithm. The IoT-fog-cloud scenario consists of three layers, in which the first layers are composed of a large number of sensors. The sensors include three different IoT sensors sensing different task data such as temperature, traffic conditions, and surveillance video. 200-2000 IoT sensors are randomly distributed within a range of 100-300 meters of the fog nodes. Each sensor generates 250-1024KB data according to the sensor task. The tolerance delay of the tasks are 100 milliseconds. The second layer consists of 8-20 fog nodes that are used for data aggregation and the processing of the data workload generated by the first layer. It is assumed that on each fog node, there is a number of virtual machines for task offloading, where each virtual machine is used for offloading a certain class of IoT application. Finally, the cloud layer consists of one data center connected to the fog layer through the Internet.

B. PARAMETERS SETTING

The simulation was run on a PC with an Intel Core i7-4510 CPU at 2.60 GHz and 8 GB of RAM. In the experiments, the parameter settings are set to the values described in Table 2. The parameter settings are obtained by conducting several preliminary experiments.

TABLE 2. Parameter setting of the conducted experiments.

Parameter	Value
BW_c	12 MB/s
BW_l	300 MB/s
$M_{sensors}$	200-2000
N_{nodes}	8-20
λ_i	10-50
c	1-3
μ_{ij}	50-300
$Dsize_i$	250 kb-1 Mb
L_{ij}	U(2-20) ms
L_{jc}	30 ms
M_{ants}	30
N_{iter}	100
ρ	0.1
ρ_g	0.2
α	0.2
β	2

C. EVALUATION CRITERION

The experiments are conducted to evaluate the proposed ACO task offloading algorithm in terms of several evaluation metrics, namely, the average response time, the degree of

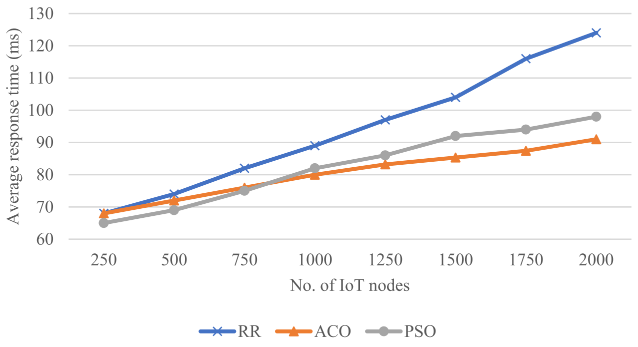

FIGURE 4. The average response time of the offloaded tasks.

imbalance, and the standard deviation of the load imbalance. The degree of imbalance shows the imbalance among the available fog nodes and is calculated using the following equation:

$$DI = \frac{Max(R_j) - Min(R_j)}{R_{average}}, \quad j = 1, 2, \ldots, N_{nodes} \tag{20}$$

The standard deviation of the response times of the offloaded tasks is used to evaluate the load distribution among the fog nodes, where a smaller value means highly balanced nodes and is calculated using the following equation:

$$SD = \sqrt{\frac{\sum_j (R_j - R_{average})^2}{N_{nodes}}} \tag{21}$$

D. EXPERIMENTAL RESULTS

The first test scenario shows the behavior of the proposed algorithms for small number of fog nodes, $N_{nodes} = 8$. The parameter settings are as follows: $c = 1$, $\lambda_i = 30$, and $\mu_{ij} = 200$. Figure 4 shows the average response times of the offloaded tasks for the proposed PSO and ACO offloading algorithms with different numbers of sensors, and the proposed algorithms are compared with the RR scheduler. In this scenario, the average average response time increases as the number of tasks is increased by adding more IoT sensors. The RR scheduler violates the set tolerant delay 100ms. The proposed ACO offloading algorithm maintains lower average response times than the PSO algorithm.

The next test scenario shows the behavior of the proposed algorithms using larger number of fog nodes, $N_{nodes} = 20$, and larger data rates, $\lambda_i = 50$. The experiment is set for one application class $c = 1$, fog node service rate $\mu_{ij} = 200$. Figure 5 shows that the proposed ACO offloading algorithm maintains lower response times as the number of IoT sensors increases, and does not violate the set tolerant delay.

Figure 6 shows the average response times of the offloaded tasks during the iterations of a single run of the proposed algorithm with the following settings: $c = 3$, $\lambda_i = 10$ for c = 1, $\lambda_i = 5$ for c = 2, $\lambda_i = 1$ for c = 3, $N_{nodes} = 20$, $\mu_{ij} = 100$ for $c = 1$, $\mu_{ij} = 50$ for $c = 2$, and $\mu_{ij} = 10$ for $c = 3$. The average response times of all the ants are recorded during each iteration.

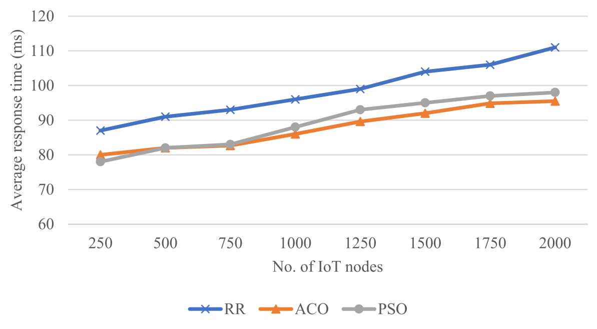

FIGURE 5. The average response time of the offloaded tasks.

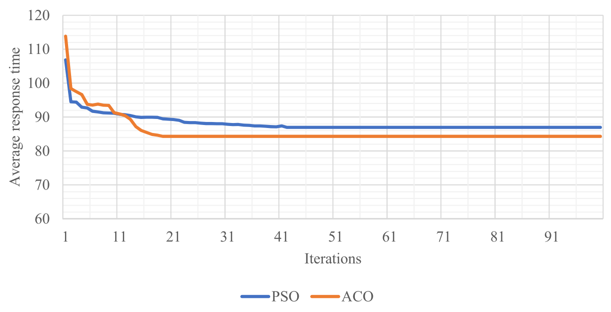

FIGURE 6. The values of the objective function during a single run of the proposed algorithm.

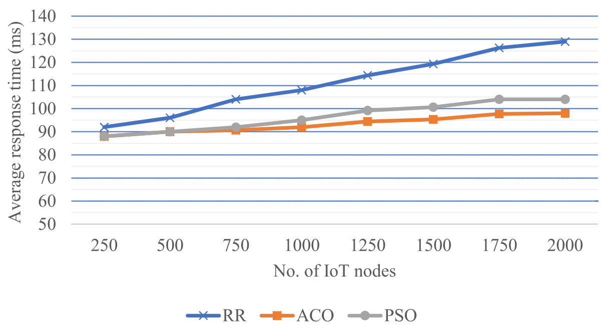

FIGURE 7. The average response time of the offloaded tasks.

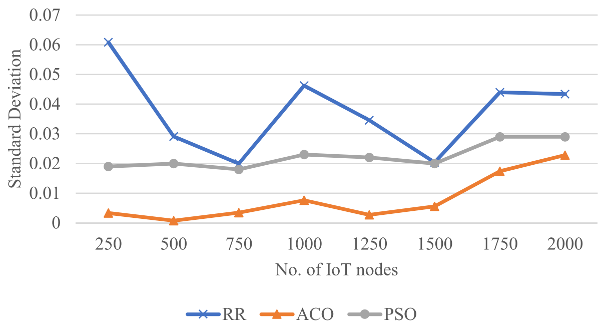

FIGURE 8. The standard deviation of the response times on the fog nodes.

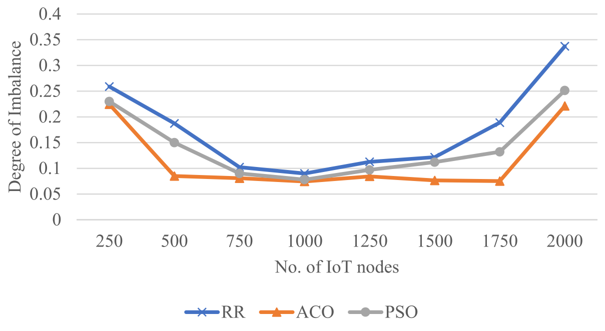

FIGURE 9. The degree of the imbalance of the offloaded tasks.

The convergence of the proposed ACO algorithm is faster than that of the PSO algorithm: the best value is reached immediately after iteration number 17. This result indicates that the proposed ACO algorithm explored the search space of possible solutions and reached the best value in a reasonable amount of time compared to the PSO algorithm.

In the following test scenario, three different sensors tasks belonging to three different application classes, $c = 3$. Each task has different data rates $\lambda_i = 10$ for c = 1, $\lambda_i = 20$ for c = 2, $\lambda_i = 40$ for c = 3. The number of fog nodes is $N_{nodes} = 10$. The service rate for each task is $\mu_{ij} = 150$ for $c = 1$, $\mu_{ij} = 100$ for $c = 2$, and $\mu_{ij} = 50$ for $c = 3$. Figure 7 shows that the proposed ACO offloading algorithm maintains lower response times as the number of IoT sensors increases compared to both the PSO and the RR algorithm. In this scenario, both RR and PSO violates the set tolerant delay. This scenario shows that the proposed ACO task offloading algorithm is able to make the best offloading decision compared to the proposed PSO algorithm.

Figure 8 shows the standard deviation of the response times with increasing number of IoT nodes using the previous parameter settings. The standard deviation of the response times is the variation among response times of all offloaded tasks from the average response time. The figure clearly shows that the proposed ACO task offloading algorithm enhanced the standard deviation compared to the RR algorithms.

Figure 9 shows the degree of imbalance for the RR algorithm and the proposed algorithm with increasing number of IoT nodes using the same previous parameter settings. The figure shows that the proposed ACO algorithm maintains fewer values. This means that the proposed algorithm balances the workload over the fog nodes effectively.

VI. CONCLUSION AND FUTURE WORK

This paper is motivated by a smart city scenario in which a large number of distributed IoT sensors produce a large amount of data to be processed to make decisions and perform coordinated actions. Fog computing is presented as essential in the smart city scenario because the fog provides a better response time for IoT applications by allowing such applications to take advantage of fog computing's low latencies rather than using the cloud for processing. A formal model of IoT task offloading on the fog nodes is provided. The formal model considers the network latency and the service rate of the fog nodes. Furthermore, two nature-inspired meta-heuristic task offloading algorithms, namely, ACO and PSO, are proposed based on the provided formal model. The experimental results show that the proposed ACO task offloading algorithm provides a significant improvement in IoT application response times and effectively balances the tasks over the fog nodes.

In future work, this research can be extend in two ways. First, a multi-objective optimization that includes power consumption, communication cost, and computation cost will be explored. Furthermore, future research will consider IoT application that may involve a collaboration between multiple IoT sensors. As a result of such collaboration, there will be dependencies between the IoT tasks produced by multiple IoT sensors. In that case, the optimization objective should consider the dependencies between the IoT tasks. Second, considering a dynamic scenarios which considers dynamic changes in the sensor nodes or the produced data rates. The IoT sensor nodes can be non-stationary and mobility is introduced. Further, the data production rate of the IoT sensor node can be dynamic as well where adaptive sampling techniques at the sensor level might produce dynamic data rate.

REFERENCES

[1] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. Workshop Mobile Big Data*, New York, NY, USA, 2015, pp. 37–42, doi: 10.1145/2757384.2757397.

[2] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 945–953.

[3] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*, M. Gris and G. Yang, Eds. Berlin, Germany: Springer, 2012, pp. 59–79.

[4] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Services*, New York, NY, USA, 2010, pp. 49–62, doi: 10.1145/1814433.1814441.

[5] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the Internet of Things," in *Proc. 2nd ACM SIGCOMM Workshop Mobile Cloud Comput. (MCC)*, New York, NY, USA, 2013, pp. 15–20, doi: 10.1145/2491266.2491270.

[6] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, Sep. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762118306349

[7] Q. Wang and S. Chen, "Latency–minimum offloading decision and resource allocation for fog–enabled Internet of Things networks," *Trans. Emerg. Telecommun. Technol.*, Jan. 2020, Art. no. e3880. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3880

[8] A. Bhattacharya and P. De, "A survey of adaptation techniques in computation offloading," *J. Netw. Comput. Appl.*, vol. 78, pp. 97–115, Jan. 2017, doi: 10.1016/j.jnca.2016.10.023.

[9] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Netw. Appl.*, vol. 18, no. 1, pp. 129–140, Apr. 2012, doi: 10.1007/s11036-012-0368-0.

[10] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the Internet of Things," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 17–24.

[11] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 1, pp. 108–119, Jan. 2017.

[12] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing towards balanced delay and power consumption," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1171–1181, Dec. 2016.

[13] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016.

[14] Y.-C. Chen, Y.-C. Chang, C.-H. Chen, Y.-S. Lin, J.-L. Chen, and Y.-Y. Chang, "Cloud-fog computing for information-centric Internet-of-Things applications," in *Proc. Int. Conf. Appl. Syst. Innov. (ICASI)*, May 2017, pp. 637–640.

[15] Y.-L. Jiang, Y.-S. Chen, S.-W. Yang, and C.-H. Wu, "Energy-efficient task offloading for time-sensitive applications in fog computing," *IEEE Syst. J.*, vol. 13, no. 3, pp. 2930–2941, Sep. 2019.

[16] M. Huang, W. Liu, T. Wang, A. Liu, and S. Zhang, "A cloud-MEC collaborative task offloading scheme with service orchestration," *IEEE Internet Things J.*, to be published.

[17] Y. Liu, Z. Zeng, X. Liu, X. Zhu, and M. Z. A. Bhuiyan, "A novel load balancing and low response delay framework for edge-cloud network based on SDN," *IEEE Internet Things J.*, to be published.

[18] S. Zahoor, N. Javaid, A. Khan, B. Ruqia, F. J. Muhammad, and M. Zahid, "A Cloud-Fog-Based smart grid model for efficient resource utilization," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2018, pp. 1154–1160.

[19] S. Zahoor, S. Javaid, N. Javaid, M. Ashraf, F. Ishmanov, and M. Afzal, "Cloud-fog-based smart grid model for efficient resource management," *Sustainability*, vol. 10, no. 6, p. 2079, Jun. 2018. [Online]. Available: https://www.mdpi.com/2071-1050/10/6/2079

[20] S. A. A. Naqvi, N. Javaid, H. Butt, M. B. Kamal, A. Hamza, and M. Kashif, "Metaheuristic optimization technique for load balancing in cloud-fog environment integrated with smart grid," in *Advances in Network-Based Information Systems,*, L. Barolli, N. Kryvinska, T. Enokido, and M. Takizawa, Eds. Cham, Switzerland: Springer, 2019, pp. 700–711.

[21] S. Ningning, G. Chao, A. Xingshuo, and Z. Qiang, "Fog computing dynamic load balancing mechanism based on graph repartitioning," *China Commun.*, vol. 13, no. 3, pp. 156–164, Mar. 2016.

[22] S. Bitam, S. Zeadally, and A. Mellouk, "Fog computing job scheduling optimization based on bees swarm," *Enterprise Inf. Syst.*, vol. 12, no. 4, pp. 373–397, Apr. 2017, doi: 10.1080/17517575.2017.1304579.

[23] H. T. T. Binh, T. T. Anh, D. B. Son, P. A. Duc, and B. M. Nguyen, "An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment," in *Proc. 9th Int. Symp. Inf. Commun. Technol. (SoICT)*, New York, NY, USA, 2018, pp. 397–404, doi: 10.1145/3287921.3287984.

[24] B. M. Nguyen, H. Thi Thanh Binh, T. The Anh, and D. Bao Son, "Evolutionary algorithms to optimize task scheduling problem for the IoT based Bag-of-Tasks application in Cloud-Fog computing environment," *Appl. Sci.*, vol. 9, no. 9, p. 1730, Apr. 2019.[Online]. Available: https://www.mdpi.com/2076-3417/9/9/1730

[25] A. Mebrek, L. Merghem-Boulahia, and M. Esseghir, "Efficient green solution for a balanced energy consumption and delay in the IoT-Fog-Cloud computing," in *Proc. IEEE 16th Int. Symp. Netw. Comput. Appl. (NCA)*, Oct. 2017, pp. 1–4.

[26] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jul. 2018, pp. 66–73, doi: 10.1109/edge.2018.00016.

[27] C. Canali and R. Lancellotti, "GASP: Genetic algorithms for service placement in fog computing systems," *Algorithms*, vol. 12, no. 10, p. 201, Sep. 2019. [Online]. Available: https://www.mdpi.com/1999-4893/12/10/201

[28] P. Schneider, N. Castell, M. Vogt, F. R. Dauge, W. A. Lahoz, and A. Bartonova, "Mapping urban air quality in near real-time using observations from low-cost sensors and model information," *Environ. Int.*, vol. 106, pp. 234–247, Sep. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0160412016310741

[29] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

[30] X.-D. Xue, X.-D. Cheng, B. Xu, H.-L. Wang, and C.-P. Jiang, "The basic principle and application of ant colony optimization algorithm," in *Proc. Int. Conf. Artif. Intell. Edu. (ICAIE)*, Oct. 2010, pp. 358–360.

[31] A. Nayyar and R. Singh, "Ant colony optimization - computational swarm intelligence technique," in *Proc. 3rd Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2016, pp. 1493–1499.

[32] I. Chaouch, O. B. Driss, and K. Ghedira, "A modified ant colony optimization algorithm for the distributed job shop scheduling problem," *Procedia Comput. Sci.*, vol. 112, pp. 296–305, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050917316769

[33] S. H. H. Madni, M. S. Abd Latiff, M. Abdullahi, S. M. Abdulhamid, and M. J. Usman, "Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment," *PLoS ONE*, vol. 12, no. 5, May 2017, Art. no. e0176321, doi: 10.1371/journal.pone.0176321.

[34] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, Dec. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000013000627

[35] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, "Cloud task scheduling based on load balancing ant colony optimization," in *Proc. 6th Annu. Chinagrid Conf.*, Aug. 2011, pp. 3–9.

[36] M. K. Hussein, M. H. Mousa, and M. A. Alqarni, "A placement architecture for a container as a service (CaaS) in a cloud environment," *J. Cloud Comput.*, vol. 8, no. 1, May 2019, doi: 10.1186/s13677-019-0131-1.

[37] H. R. Boveiri, R. Khayami, M. Elhoseny, and M. Gunasekaran, "An efficient swarm-intelligence approach for task scheduling in cloud-based Internet of Things applications," *J. Ambient Intell. Hum. Comput.*, vol. 10, no. 9, pp. 3469–3479, Oct. 2018, doi: 10.1007/s12652-018-1071-1.

[38] J. Fan, X. Wei, T. Wang, T. Lan, and S. Subramaniam, "Deadline-aware task scheduling in a tiered IoT infrastructure," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–7.

[39] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[40] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Phys. A, Stat. Mech. Appl.*, vol. 391, no. 6, pp. 2193–2196, Mar. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378437111009010

[41] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4. Nov. 1995, pp. 1942–1948.

[42] M. Clerc, *Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem*. Berlin, Germany: Springer, 2004, pp. 219–239, doi: 10.1007/978-3-540-39930-8_8.

MOHAMED K. HUSSEIN received the B.Sc. degree in computer science from Alexandria University, Egypt, in 1996, and the M.Sc. degree from Ain Shams University, Egypt, in 2003. He is currently pursuing the Ph.D. degree with The University of Manchester, U.K., in 2008. He joined the Department of Computer Science, Suez Canal University, where he is currently an Associate Professor. His research interests include operating systems, cloud computing, and fog computing.

MOHAMED H. MOUSA received the B.Sc. degree in computer science and pure mathematics from Ain-Shams University, Egypt, in 1996, the M.Sc. degree from Helwan University, Egypt, in 2001, and the Ph.D. degree from Claude Bernard University Lyon1, France, in 2007. He is currently an Associate Professor with the Department of Computer Science, Suez Canal University, Egypt. His research interests include high performance computing and GPU computing.