Hindawi
Complexity
Volume 2021, Article ID 5563531, 12 pages
https://doi.org/10.1155/2021/5563531

Research Article

Intelligent Computation Offloading for IoT Applications in Scalable Edge Computing Using Artificial Bee Colony Optimization

Mohammad Babar,[1] **Muhammad Sohail Khan**,[1] **Ahmad Din**,[2] **Farman Ali**,[3] **Usman Habib**,[4] **and Kyung Sup Kwak**[5]

[1]*Department of Computer Software Engineering, University of Engineering and Technology, Mardan 23200, Pakistan*
[2]*Department of Computer Science, COMSATS University Islamabad (CUI), Abbottabad Campus, Islamabad 22010, Pakistan*
[3]*Department of Software, Sejong University, Seoul 05006, Republic of Korea*
[4]*National University of Computer & Emerging Sciences, Chiniot-Faisalabad Campus, Islamabad, Pakistan*
[5]*Department of Information and Communication Engineering, Inha University, Incheon 22212, Republic of Korea*

Correspondence should be addressed to Mohammad Babar; mbabarcs@gmail.com and Kyung Sup Kwak; kskwak@inha.ac.kr

Received 23 January 2021; Accepted 24 April 2021; Published 4 May 2021

Academic Editor: Ning Cai

Most of the IoT-based smart systems require low latency and crisp response time for their applications. Achieving the demand of this high Quality of Service (QoS) becomes quite challenging when computationally intensive tasks are offloaded to the cloud for execution. Edge computing therein plays an important role by introducing low network latency, quick response, and high bandwidth. However, offloading computations at a large scale overwhelms the edge server with many requests and the scalability issue originates. To address the above issues, an efficient resource management technique is required to maintain the workload over the edge and ensure the reduction of response time for IoT applications. Therefore, in this paper, we introduce a meta-heuristic and nature-inspired Artificial Bee Colony (ABC) optimization technique that effectively manages the workload over the edge server under the strict constraints of low network latency and quick response time. The numerical results show that the proposed ABC algorithm has outperformed Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Round-Robin (RR) Scheduling algorithms by producing low response time and effectively managing the workload over the edge server. Furthermore, the proposed technique scales the edge server to meet the demand of high QoS for IoT applications.

1. Introduction

Internet of Things (IoT) is reshaping the technological landscape of traditional systems. The concept of IoT-based smart system is making its way from dreams to reality [1]. Smart city, smart healthcare, and smart industry have grasped the researchers' attention at a monumental scale [2–5]. However, smart systems generate high volume of data in a short period of time and create several challenges such as data management, security, storage, and energy consumption [6]. In addition, the applications pertaining to these IoT-based systems are resource-constrained and require a crisp response, low latency, and high bandwidth, which are beyond their capabilities [7].

Cloud computing is considered as a resource-rich solution to the above problems. However, the inherent longer latencies of cloud computing make it nonviable. These longer latencies hinder the performance of IoT-based smart systems [8]. Edge computing offers computation, storage, and communication services at the edge of a network, resulting in low latency, high bandwidth, and energy-efficiency [9]. Both the edge and fog computing architectures have been used to handle resource-scarcity of IoT devices [10]. In this work, the objective behind the utilization of

edge-based architecture is to deploy edge as a micro data center that has the potential to provide cloud like services, even in the absence of the cloud. However, fog computing provides computing, storage, and other services through intermediate nodes such as routers and gateways, which are resource-limited. Edge computing utilizes computation offloading concept, where the resource-constrained IoT devices handle compute-intensive tasks to the edge server, execute the task, and send back the result to IoT devices. Computation offloading not only saves energy in IoT devices but also extends the lifetime of these devices [11]. Figure 1 shows edge computing architecture for IoT.

One way to achieve the required high QoS is via computation offloading application of edge computing [12]. However, computation offloading is rather a complex job, which enfolds the complexity of task scheduling, partitioning, migration, and latency. In addition, the offloading has a vital role in edge discovery, as well as selecting an appropriate edge node for computation.

The current IoT-based smart systems use large-scale sensors that generate a huge amount of data at the IoT deployment layer. The rapid processing of the generated data is very substantial. Therefore, computation offloading for the resource-constrained devices is quite significant. The objective of this research is to scale the edge server for delay sensitive tasks that demand stringent QoS requirements. A computation offloading technique selects a task from the IoT layer generated by IoT devices and offloads it to the edge server for execution. However, computation offloading at a large scale creates congestion on the edge server, which provides low QoS. Therefore, a resource scheduling mechanism for load balancing over edge servers is required to ensure the effective utilization of the edge resources, while considering communication cost and response time of the tasks. On the other hand, computation offloading is a nontrivial, challenging, and NP-hard problem, where its complexity is directly proportional to the increasing number of offloading tasks. Thereby, several studies proposed greedy algorithms to tackle computation offloading problem [13, 14], but the computation offloading still grows exponentially and became very challenging for traditional greedy algorithms. Therefore, in this paper, we proposed an Artificial Bee Colony- (ABC-) based computation offloading algorithm that effectively and seamlessly performs the process of computation offloading. The major contributions of this study are as follows:

(i) We devised a classical three-tier framework by integrating edge and cloud to simulate computation offloading process following strict energy and latency constraint for delay-sensitive tasks. An edge server is used in conjunction with cloud due to its higher computing power than edge servers. In the proposed framework, the inclusion of cloud further scales the edge server efficiently.

(ii) To effectively balance the workload over edge servers, we propose ABC optimization technique based on swarm intelligence, where the objective function is set to achieve the minimum computation cost and low latency for the offloaded tasks.

(iii) A computation offloading algorithm based on ABC is implemented for seamless computation offloading. The results exhibit that the proposed technique shows notable improvement in reducing the response time and efficient load balancing over edge nodes, compared to Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Round-Robin (RR) Scheduling.

The rest of the paper is structured as follows. Section 2 presents a detailed overview of the related work. Section 3 shows the edge-cloud integration framework that spans over the system model. In addition, it also shows the computation offloading algorithm based on Artificial Bee Colony. Section 4 presents the results and discussion of the proposed technique. Finally, Section 5 comprehends the conclusion of this research study.

2. Related Work

Smart systems are built using a large number of IoT devices, which generate huge amount of data in a short period of time. The generated data is sent to the cloud for aggregation, analytics, and computation [15–17]. Computation offloading to the cloud decreases the computation load from IoT devices. However, it produces excessive delay that violates the QoS requirements for real-time systems and applications [18]. Furthermore, it leads to inefficient use of resources and unnecessarily overloads back-haul links [19]. Edge computing brings down computation, storage, and other services in the closed proximity of the users, thereby meeting the strict QoS requirements for delay-sensitive applications.

Computation offloading has largely attracted the researcher's attention, and a number of studies are conducted on edge computing domain. In [20], a three-tier edge-cloud integration framework has been deployed to reduce energy and latency of IoT devices. To identify the best edge server for computation offloading, the expected offloading and the propagation delay of different edge servers are considered to determine a threshold. If the expected delay is below the threshold, the request is accepted for offloading; otherwise, the task is given to the next feasible edge server. The edge server is further connected to the cloud for scalability, where the task is offloaded to the cloud if the edge server reaches its maximum. This work produces desirable results, but they did not consider the communication cost.

In [21], the authors focused on scheduling the problem of computation offloading over the edge server placed in the proximity of users. The offloading and computation decisions are made by the IoT device, while considering the battery life and response time for better quality of experience (QoE). They formulated the offloading problem as mixed-integer nonlinear programming (MINLP) and resolved it using branch and bound reinforcement learning technique. The proposed solution did not consider the load balancing over edge and it leads to scalability issue.

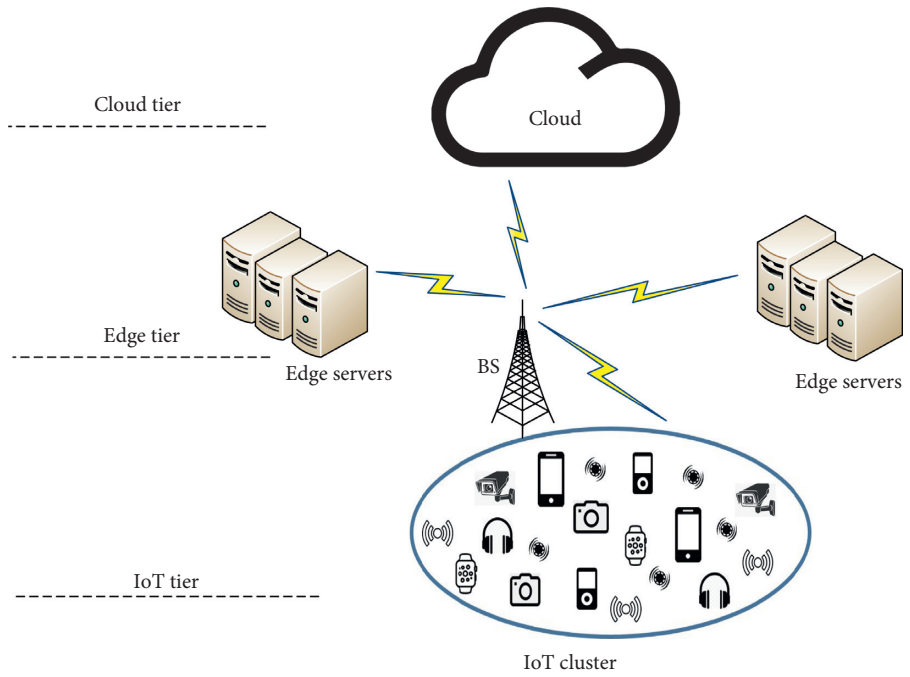

Figure 1: Edge computing architecture for IOT.

A distributed and scalable framework for wearable IoT devices is proposed in [22]. This existing system considered metrics such as response time, bandwidth, storage, and a number of tasks successfully completed for effective resource provisioning using recurrent learning. The response time is reduced by introducing a control layer between cloud and IoT layer, which generalizes the edge computing model but lacks practical computation offload environment. A dynamic computation offloading algorithm based on stochastic optimization technique is presented to deal with computation offloading problem [23]. The computation offloading problem is further divided into subproblems to achieve a minimum cost of the offloading process. They reduced the cost but did not consider the load balancing over multiple edge servers while migrating tasks.

The existing computation offloading frameworks comprised over cloud, edge, and IoT have been devised for seamless computation offloading [24–26]. These studies focused on reducing the communication overhead, response time, and bandwidth using machine learning algorithms such as Lyapunov optimization, Deep Supervised Learning (DSP), and Discrete Particle Swarm Optimization (DPSO). These frameworks have a well-defined objective for low latency and minimize energy consumption of IoT devices. However, this existing system focuses on edge-cloud framework instead of computation offloading process. A load balancing framework based on directed graph partitioning algorithms is proposed to balance load over edge node for in-network flexible resource provisioning and allocation [27]. The devised algorithm is inappropriate in the dynamic workload conditions. A cloud-edge integration architecture is introduced to deal with the scheduling problem of bag-of-tasks applications using Modified Particle Swarm Optimization (MPSO) [28]. The similar problem is handled using Genetic Algorithm (GA) [29]. The main goals of these studies were to reduce the operating cost and remote processing time of the task. However, these studies have not focused on communication cost and latency incurred by the computation offloading process.

Several studies use different clustering techniques to protect the edge server from bottleneck and cope with the scalability issue. For instance, a CNN-based fused tile partitioning (FTP) is presented to distribute the workload over edge servers [30]. A PSO-based multiclustering technique in a semiautonomous edge-IoT environment is proposed in the account of reducing processing and communication delays to distribute the load over edge servers [31]. A graph-based edge clustering technique and software defined network- (SDN-) based multicluster overlay (MCO) are utilized to optimize task size, number of servers, required channels for communications, best channel allocation for effective load distribution, and scaling the edge server [32, 33]. However, these studies produce additional communication overhead and add more latency while making computation offloading decision.

The existing studies reveal that the computation offloading process is very complex and challenging. It consumes extra energy and incurs latency, while intercommunicating between devices and servers [5, 6]. A single device is responsible for making computation offloading decisions, which consumes more energy and causes fast battery drain. In addition, the existing edge server makes computation offloading decision independently in the IoT environment [34–37], where the edge server is overwhelmed with many requests, creating congestion over the edge server, and originates scalability issue. The existing approaches do not attain the high QoS requirements of IoT applications. Nonetheless, they reflect a trade-off between QoS and the scale of offloading requests.

In our work, we design a dynamic and decentralized task execution through computation offloading. To accomplish the above-mentioned objective, a three-tier edge-cloud integration framework is designed for a successful computation offloading process. One of the major advantages of the proposed layered-based architecture is a robust service discovery. The IoT system is designed using a large number of devices and servers, where searching for the right resource for the IoT device is quite challenging. A social Internet of things (SIoT) clustering approach [38] is deployed at IoT layer that performs the task of aggregation and resource management. The SIoT not only controls the number of offloading tasks sent to the edge server and protects the edge server from bottleneck but also creates an association between offloading task and the resource allocation. This association finds the right resource for executing a particular task, hence reducing the latency of the offloading task.

We propose an ABC optimization technique that balances the workload over the edge server, provides the right resource for offloading device, and exploits low latency interconnections between IoT device and server. The proposed framework can be effectively utilized for IoT devices, where the task is executed under strict energy with the required latency to meet the high QoS requirements, which is very unlikely to get using the traditional cloud. Finally, a computation offloading algorithm based on ABC is proposed to provide an efficient computation offloading facility that searches to find new resources for task execution. The objective function measures the network latency and execution time of the task to achieve minimum service time. The detailed discussion on ABC Algorithm and computation offloading technique is provided in Section 3.

3. Edge-Cloud Integration Framework for Computation Offloading

In this section, we briefly describe the system model. In addition, we also provide the detailed overview of the Artificial Bee Colony optimization technique and a novel ABC-based computation offloading algorithm.

3.1. System Model. There are three layers in the IoT-based edge infrastructure as shown in Figure 2. The first layer is IoT layer, where a large number of sensors are connected to LAN in clusters. These clusters are connected to the second layer called edge layer, which contains multiple edge nodes. These nodes perform basic analytics on the data received from the sensors. However, the computational and storage capabilities of these nodes are limited. The third layer is the cloud layer. The master node of second layer is responsible for deciding whether to offload the task to the cloud layer or not. It has powerful computing and storage resources to perform heavy analytics and large/long-term data storage. Table 1 expresses the list of symbols and notations used in the system model.

The IoT layer has N nodes $\{S_1, S_2, \ldots, S_n\}$, where each sensor/mote S_i is working at frequency λ_i. Edge layer has M nodes $\{Eg_1, Eg_2, \ldots, Eg_m\}$. If the computation job is performed in the IoT node without offloading it to the fog or cloud, then the service time for a job is computed using the following equation:

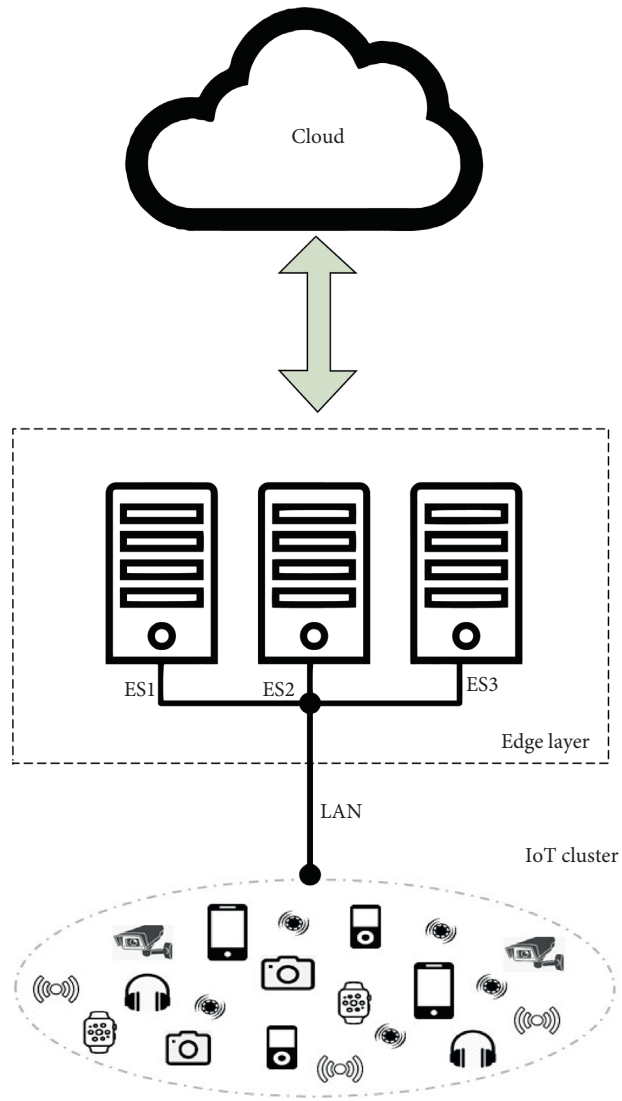

Figure 2: Edge-computing-based system model.

Table 1: Symbols and notations used in the system model.

Symbols	Descriptions
S_n	IoT nodes n
Eg_m	Edge servers m
ST_i	Service time of IoT node i
T_{com_i}	Computation requirement of task
λ_i	Frequency of sensor
C_{sen}	Computation capacity of sensor
NL_{me}	Network latency (IoT to edge)
B_{me}	Bandwidth (IoT node to edge)
D_m	Data size
C_{mote}	Energy required for task execution
EC_{BW}	Bandwidth (edge to cloud)

$$ST_n = \frac{T_{\text{com}_i}}{C_{\text{sen}}}, \tag{1}$$

where ST_n is a service time of the task, T_{com_i} is computation required to complete the task i, and C_{sen} is computational capability of the mote (IoT sensor node). The communication cost between mote and edge node is computed using equation (2), which depends on transfer capacity and network latency.

$$\text{MoteEdgeComm} = NL_{\text{me}} + \frac{D_m}{B_{\text{me}}}, \tag{2}$$

where NL_{me} is network latency between mote and edge, D_m is data generated by the mote/sensor, and B_{me} is the bandwidth between mote and the edge. If the computation is performed in the IoT node, the energy (E_m) required for the computation of the job is given by the following equation:

$$E_m = ST_n * C_{\text{mote}}, \tag{3}$$

where C_{mote} is an energy consumed by processing units in unit time. The service time of the job offloaded to the edge node is calculated using the following equation:

$$ST_{EN} = \text{MoteEdgeComm} \times \frac{T_{\text{com}_i}}{C_{\text{EdgeNode}}}. \tag{4}$$

In the above equation, C_{EdgeNode} is clock frequency of the edge node. The task offloaded to the edge node required energy for its completion, which is calculated using the following equation:

$$E_{\text{edge}} = 2 \times \frac{D_m}{B_{\text{me}}} + \frac{T_{\text{com}_i}}{C_{\text{EdgeNode}}}. \tag{5}$$

The service time for job offloading to the cloud is computed using the following equation:

$$ST_{\text{cloud}} = \text{EdgeCloudComm} + \frac{T_{\text{com}_i}}{C_{\text{cloud}}}, \tag{6}$$

where EdgeCloudComm is composed of the cloud latency and time required to send data to cloud from the edge node. It is calculated using the following equation:

$$\text{EdgeCloudComm} = NL_{EC} + \frac{T_{\text{com}_i}}{C_{\text{EdgeNode}}}, \tag{7}$$

where NL_{EC} is network latency between the edge node and cloud, while EC_{BW} is bandwidth between edge and cloud. Therefore, the energy consumption for the offloaded task to the cloud is calculated using the following equation:

$$E_{\text{cloud}} = 2 \times \frac{D_m}{EC_{BW}} + \frac{T_{\text{com}_i}}{C_{\text{cloud}}}. \tag{8}$$

The total communication cost for offloading the job to cloud using IoT node is computed by the following equation:

$$\text{TotalCommCost} = \text{EdgeCloudComm} + \text{MoteEdgeComm}. \tag{9}$$

3.1.1. Objective Function. It is important to handle two main decisions of whether to offload the task to edge or cloud. Therefore, the objective function is used to minimize the energy consumption (E) and service time delay (ST) of each offloading scheme. The first decision is about offloading the job to the edge:

$$s_i = \begin{cases} 0, & \text{if job } i \text{ is offloaded to the edge,} \\ 1, & \text{if job } i \text{ is exectuted using mote computational resource.} \end{cases} \tag{10}$$

Similarly, the edge might have limited resources to fulfil the computational requirements of the job. Therefore, it may decide to further offload the task to the cloud. The following variables in decision allow offloading the job to the cloud:

$$t_i = \begin{cases} 0, & \text{if job } i \text{ is offloaded to the cloud,} \\ 1, & \text{if job } i \text{ is offloaded to the edge.} \end{cases} \tag{11}$$

For ABC optimization algorithm, ST_i and E_i are required to be normalized using the two following equations:

$$ST_{\text{current}} = \frac{ST_{\text{current}} - ST_{\text{min}}}{ST_{\text{max}} - ST_{\text{min}}}, \tag{12}$$

$$E_{\text{current}} = \frac{E_{\text{current}} - E_{\text{min}}}{E_{\text{max}} - E_{\text{min}}}. \tag{13}$$

For all K jobs, total energy consumption E_{total} is calculated using equation (15), and the total delay for offloading the task to the cloud is computed using the following equation:

$$ST_{\text{total}} = \sum_{n=1}^{K} \left(ST_n(s_i) + ST_{\text{edge}}(s_i) + ST_{\text{cloud}}(1 - s_i)(1 - t_i) \right), \tag{14}$$

$$E_{\text{total}} = \sum_{n=1}^{K} \left(E_n(s_i) + E_{\text{edge}}(s_i) + E_{\text{cloud}}(1 - s_i)(1 - t_i) \right). \tag{15}$$

Therefore, the objective function is

$$F_{\text{obj}} = \delta(ST) + (1 - \delta)(E), \tag{16}$$

where ST is the service time and E is energy consumption for the offloading scheme.

$$ST = \min\left(ST\left(l_j, \text{job}_r\right)\right), \quad \text{for } 1 \le j \le M,\ 1 \le r \le k, \tag{17}$$

$$E = \min\left(E\left(l_j, \text{job}_r\right)\right), \quad \text{for } 1 \le j \le M,\ 1 \le r \le k, \tag{18}$$

where $\delta \in [0, 1]$ is weight to prioritize the elements of the objective function and $(E(l_j, \text{job}_r))$ is energy consumption of the task job_r by the node at level l_j. The objective function has following constraint on the edge layer:

$$\frac{\sum_{j=1}^{k} H_k}{H_k - H_{\text{avg}}} \le Eg_{\text{cap}}, \tag{19}$$

where Eg_{cap} is a load capacity of the edge node j and H_k is an asthmatic mean of response time if it is the only edge that serves all the offloaded jobs.

3.2. Artificial Bee Colony (ABC) Algorithm. The ABC was proposed by Dervis Karaboga [39]. It is a swarm-intelligence-based optimization algorithm, which contains three types of bees. The first type is scout bees that search for new sources of food randomly, thus ensuring exploration. The second type is onlooker bees that choose a food source by observing the dance of employed bee. The third type is employed bees that are linked to the food source, thus ensuring exploitation. Scout bees and onlooker bees are not linked to any specific food source. Therefore, they are usually called unemployed bees. A general outline of the ABC algorithm is shown in Algorithm 1.

In this section, we presented the ABC algorithm for computation offloading at IoT edge. The objective function measures the service time (network latency and time required for job completion) and energy consumption for the solution provided by the optimization algorithm. The main purpose is to minimize the objective function, which searches for minimum computational cost and job latency. There are three decision variables: s_i, t_i, and ξ (contains a list of jobs). The input for the algorithm is a set of jobs and nodes.

3.2.1. Initialization Phase. The population (nodes) is represented by vectors x_n, which is initialized by bees using the following equation:

$$x_{nj} = lb_j + \text{rand}(0,1) \times \left(ub_j - lb_j\right), \tag{20}$$

where ub_j and lb_j are the upper and the lower bounds of the parameter, respectively.

3.2.2. Employed Bee Phase

(1) New Solution. The employed bee searches for new nodes (y_{mi}) with more resources in a neighbourhood. The new neighbour node y_{mi} can be found by the the following equation:

$$y_{nj} = x_{nj} + \tau_{nj} \times \left(x_{nj} - x_{pj}\right), \tag{21}$$

where τ_{nj} is a function that generates a random number between −1 and 1 and x_{pj} is a randomly chosen node in a neighbourhood.

(2) Greedy Selection. Fitness of new node y_{mi} is calculated. If its fitness is high, then x_{nj} y_{mi} is memorized.

3.2.3. Onlooker Bee Phase

(1) Probability Calculation Based on Fitness. Onlooker bees choose the node probabilistically based on information provided by the employed bee nodes. The onlooker bees choose node x_n using the probability p_n as shown in the following equation:

$$p_{nj} = \frac{F_{nj}(x_n)}{\sum_{n=1}^{k}(F_k)(x_n)}, \tag{22}$$

where F_n is fitness function, which is computed by using the following equation:

$$F_n = \begin{cases} \dfrac{1}{1 + F_{\text{obj}}}, & \text{if } F_{\text{obj}} \geq 0, \\ 1 + \left|F_{\text{obj}}\right|, & \text{if } F_{\text{obj}} < 0. \end{cases} \tag{23}$$

(2) New Solution for Onlooker Bee. Once a node is chosen for the onlooker bee probabilistically, a new neighborhood node y_{mi} is determined using equation (20).

(3) Greedy Selection. At this stage, y_{mi} and x_{nj} are compared to each other. If a new node in neighborhood y_{mi} has high fitness value, then current node y_{mi} is memorized.

3.2.4. Scout Bee Phase. Scout bees ensure exploration and choose a node randomly. An employed bee becomes a scout bee if it fails to improve its solution in a limit (number of trails). Figure 3 and Algorithm 2 present the flowchart and algorithm of ABC-based computation offloading technique, respectively.

4. Results and Discussion

In this section, we discuss the results achieved using our proposed framework. A three-tier edge-cloud integration framework is proposed, where IoT devices are placed at Tier-1, where data are generated from multiple devices in multitasking manner. Tier-2 comprises edge servers, and Tier-3 includes a resource-rich cloud. The hierarchical representation of the proposed framework helps efficiently utilize the resources and distinguish the responsibility of each tier. A simulation-based edge-cloud integration test-bed is designed using MATLAB. The simulation setup not only provides the opportunity to conduct the experiment in the control environment using a preferred set of parameters but also allows us to repeat the experiment under different scenarios and constraints. Thereby, we have evaluated the performance of the proposed Artificial Bee Colony computation offloading algorithm against the metaheuristic algorithms such as Particle Swarm Optimization, Ant Colony Optimization, and Round-Robin Scheduling [10]. A list of parameters with their corresponding values acquired by conducting several preliminary experiments is provided in Table 2.

The metrics for the evaluation of the proposed offloading algorithm are degree of imbalance and standard deviation in response time, while observing the load of edge node. The degree of imbalance among edge nodes is calculated using the following equation:

(1) Step 1: Initialization Phase
(2) **repeat**
(3) Step 2: Employed bees' phase
(4) Step 3: Onlooker bees' phase
(5) Step 4: Scout bees' phase
(6) Step 5: Memorize the best solution achieved so far
(7) **until** maximum cycle number reached
(8) Output the best solution identified

Algorithm 1: ABC algorithm.

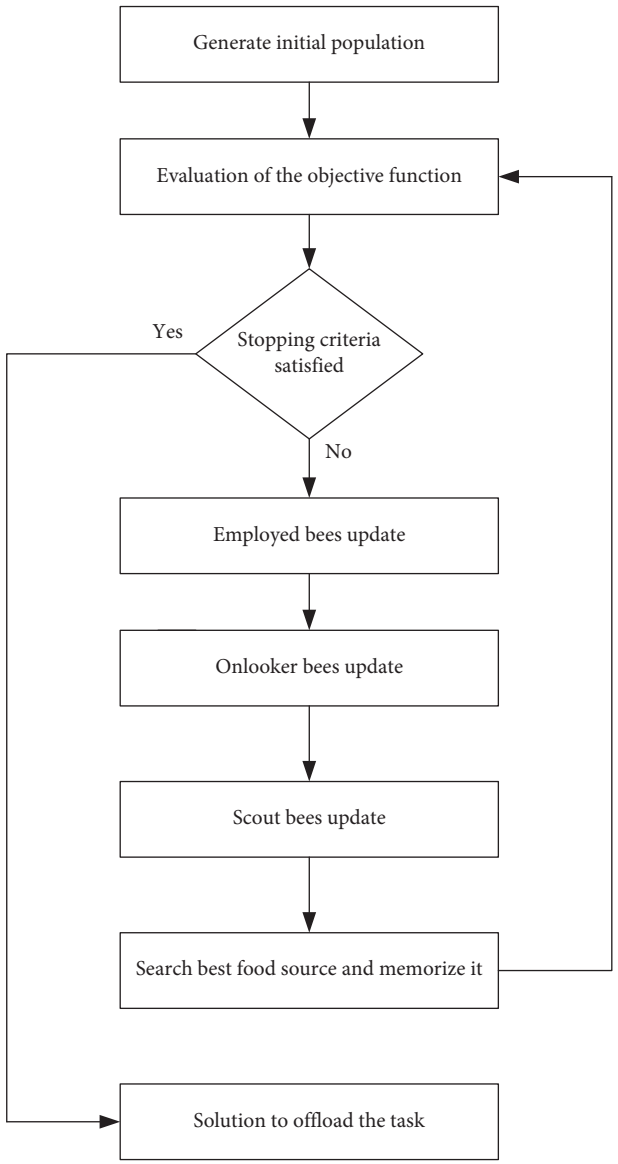

Figure 3: Flow chart of computation offloading framework.

$$LI = \frac{\max(H_k) - \min(H_k)}{H_{\text{avg}}}. \tag{24}$$

The IoT devices at Tier-1 generate the tasks, selected for computation offloading to the edge server. This approach leverages the IoT devices to save energy and make them capable of handling compute-intensive tasks. The edge tier is placed between IoT tier and cloud, which takes the data load generated by IoT devices and executes the offloaded tasks using a number of edge servers. However, each server has several virtual machine (VM) instances that ensure the successful execution of the offloaded tasks, where each VM handles a different class of IoT applications. The cloud is a resource-rich solution to IoT and is connected to the edge server through Internet for achieving scalability in the edge server. The performance analysis of computation offloading based on ABC algorithm is evaluated using three-tier edge-cloud framework for seamless and successful execution of task offloading between IoT, edge, and cloud. In this experiment, we have considered the response time, standard deviation, and degree of imbalance as a set of parameters. Using these parameters, we tested the performance of ABC-based computation offloading algorithm under different scenarios. However, the degree of imbalance reflects the inequality among multiple edge servers, and the standard deviation of the response time exhibits the load balance between edge servers. The degree of imbalance is expressed using equation (24).

Figure 4 shows the performance of the ABC task offloading algorithm in scenario 1, where the number of edge server Eg_n is 3, total number of IoT nodes M_n is 250, and primary server rate is 100. The two delay-sensitive and delay-tolerant applications are generated from IoT nodes. It has been observed that the proposed ABC task offloading algorithm outperforms the counterparts ACO, PSO, and RR Algorithms by keeping the response time well below the defined latency threshold. The ABC algorithm satisfies the QoS requirement of both delay-sensitive and delay-tolerant application. However, the RR scheduler degrades its performance by violating the latency requirements of 100 ms. In scenario 2, experimental parameters are changed by increasing the number of edge servers Eg_n to 5 and server rate to 300 with a gradual increase in number of IoT devices. In Figure 5, the proposed ABC task offloading algorithm is compared with ACO, PSO, and RR scheduler. The achieved results show that the proposed algorithm maintains the low response time even with the increasing number of computation offloading requests of IoT devices.

In Figure 6, we present the performance of ABC algorithm. It is witnessed that the proposed algorithm maintains lower response time of the offloaded tasks in the single run of

```
(1) Step1: Initialization
(2) q ← # of employed bees, r ← # of onlooker bees
(3) Dp ← # dimension of problem
(4) StoppingCriteria ← Max. # of iterations allowed
(5) Create an initial population using Equation (20)
(6) Evaluate the fitness of the population
(7) repeat
(8)    Step 2: Employed bees' phase
(9)    k = 1
(10)   while k < q do
(11)          Compute new solution using Equation (21)
(12)          Compute the fitness value of new solution using Equation (23)
(13)          if fit(y_nk) > fit(x_nk) in a neighborhood then
(14)                x_nk = y_nk, and trail_n = 0
(15)          else
(16)                Increase trail_n by 1
(17)          end if
(18)          k = k + 1
(19)   end while
(20)   Step 3: Onlooker bees' phase
(21)   Dp = r, s = 1, k = 0
(22)   while s < Dp do
(23)       Generate a random number pr such that r ∈ [0, 1]
(24)       Calculate the probability p_nj using Equation (22)
(25)       if pr < p_nj then
(26)             Compute new solution using Equation (21)
(27)             Compute the fitness value of new solution using Equation (23)
(28)             if fit(y_nk) > fit(x_nk) in a neighborhood then
(29)                   x_nk = y_nk, and trail_n = 0
(30)             else
(31)                   Increase trail_n by 1
(32)             end if
(33)       end if
(34)       s = s + 1
(35)       k = k + 1
(36)       if k > Dp then
(37)             k = 1
(38)       end if
(39)   end while
(40)   Step 4: Scout bees' phase
(41)   if trail > limit then
(42)   Initialize randomly chosen solution using Equation (20)
(43)   end if
(44) Step 5: Memorize the best solution achieved so far
(45) until maximum cycle number reached
(46) Output the best solution identified
```

Algorithm 2: ABC to offload computation to the edge/cloud.

Table 2: Simulation parameters.

S. no.	Parameters	Value
1	No. of IoT devices	**250–2000**
2	Edge server	**Core i7 (2.6 GHz, 8 GB RAM)**
3	IoT cluster radius	**100-300 meters**
4	Task size	**250 Kb–1 MB**
5	Latency	**100 ms**
6	No. of servers	**3–8**
7	Communication parameters	**3GPP**

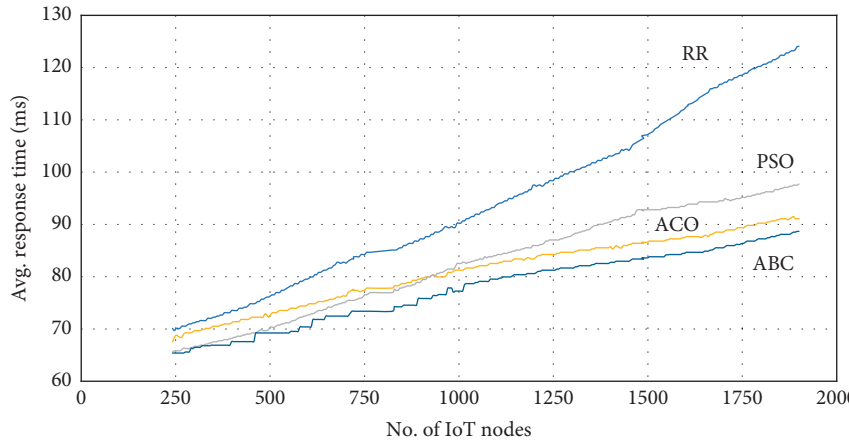

Figure 4: Average response time of the offloading tasks.

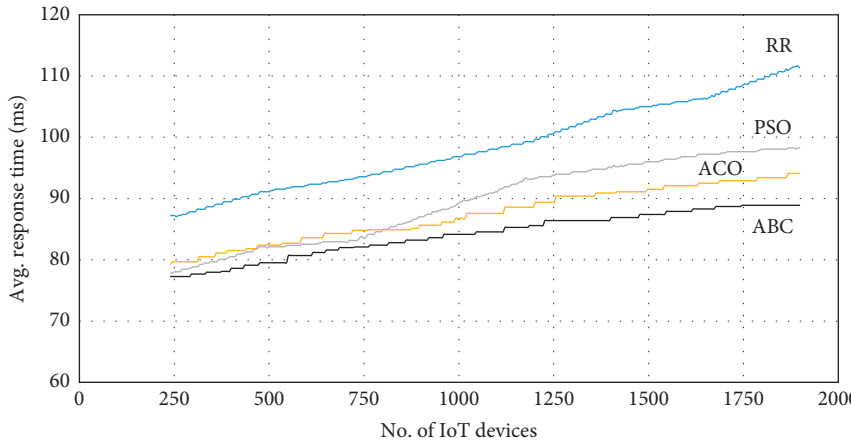

Figure 5: Average response time of the offloaded tasks.

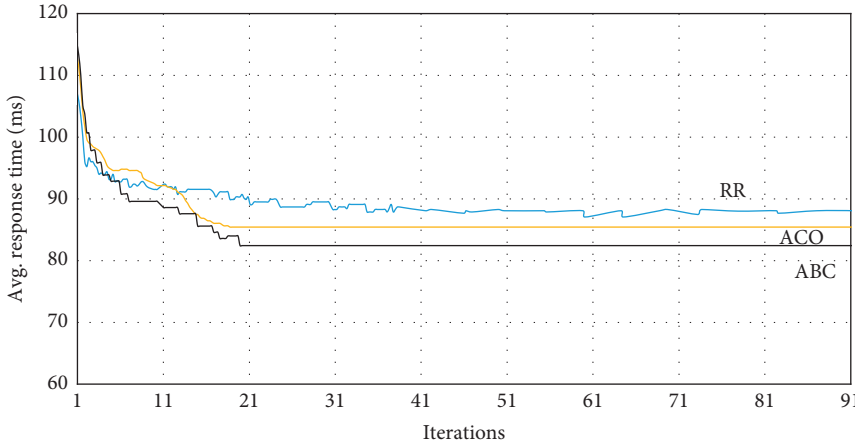

Figure 6: Average response time of the offloaded tasks.

the ABC algorithm. In scenario 3, the experimental setup is deployed with the following parameter settings: $Eg_n = 8$, server rate = 500, and $M_n = 2000$. However, the number of applications is increased to 3. This setup is objectively designed to simulate real IoT-based smart system environment, where heavy traffic is generated by IoT nodes.

We recorded the average response time of the bees. The ABC algorithm minimizes the objective function by discovering minimum computational cost and low latency of each offloaded task. The probabilistic calculation and memorization of the fitness value reach the minimum fitness value immediately in the 11th iteration. The proposed ABC

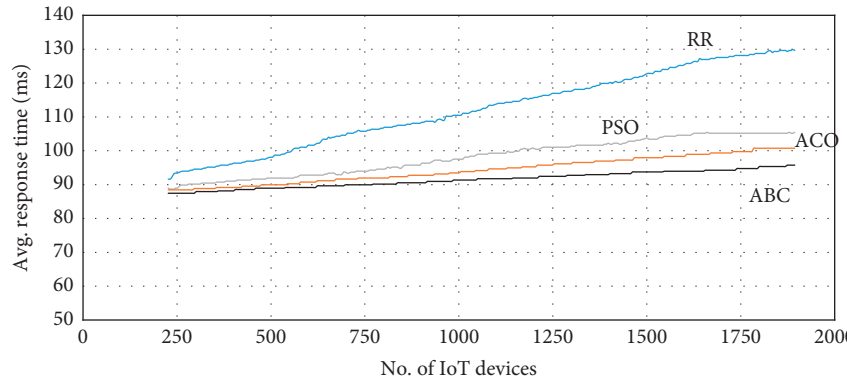

Figure 7: Average response time of the offloaded tasks.

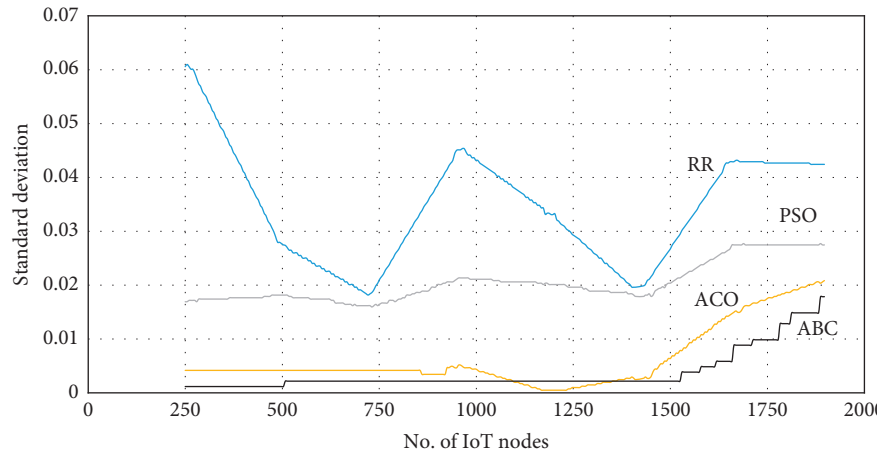

Figure 8: Standard deviation of the response time on the edge nodes.

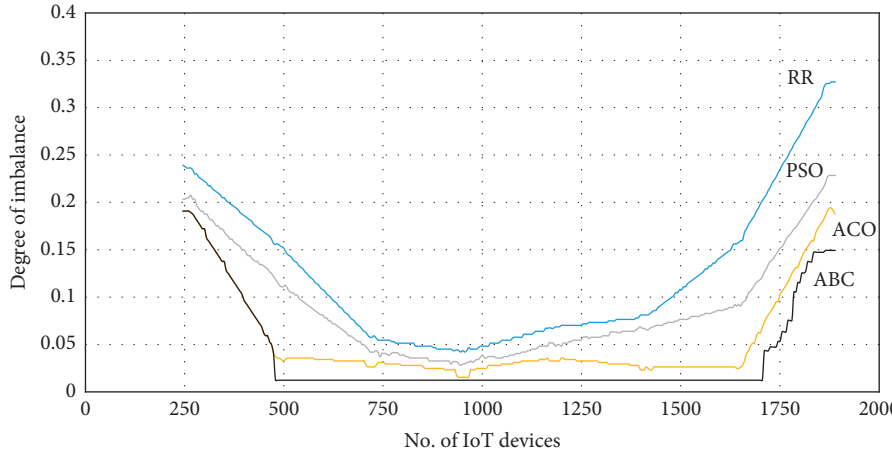

Figure 9: Degree on imbalance of the offloaded tasks.

algorithm explores the search space quickly, resulting in faster convergence compared to the ACO and PSO for the best possible solution. In addition, it achieves the best value in a short period of time.

In scenario 4, we changed a set of parameters by introducing three different classes of applications using three different types of sensors. Each sensor generates different data rate according to the task with having number of edge servers $Eg_n = 8$, the server rate = 500, and number of IoT nodes $M_n = 2000$.

In Figure 7, the results exhibit that the ABC task offloading algorithm maintains the low response time because

the onlooker bee probabilistically selects and memorizes the successful node while looking for other probabilistic solutions and outperforms the RR, PSO, and ACO algorithms. Figure 8 reflects the behaviour of the proposed ABC algorithm by considering the standard deviation. The standard deviation is the variation between the average response times of all the tasks that are offloaded for remote execution. As the number of IoT devices increases, the proposed algorithm shows an improved performance in the standard deviation. However, the standard deviation grows exponentially while exceeding 1750 IoT devices, which is due to the inherent issue of scalability in edge computing. These results are achieved through the same parameters mentioned in scenario 4. Figure 9 exhibits the degree of imbalance of the offloaded tasks over edge servers while increasing the number of offloading tasks from IoT layer using scenario 4. It is observed that the proposed ABC algorithm minimizes the objective function, produces low values, and reflects that the workload is effectively distributed among the edge servers.

5. Conclusion and Future Work

Smart city is designed using a large number of IoT devices. These devices are resource-limited and their applications are resource-intensive, which require high QoS. Nonetheless, they produce a large amount of data in a short period of time. Cloud can be a feasible solution for it, but the inherent longer latency makes it nonviable. Edge computing is a potential solution that resides in the close proximity of the users. It offers low latency, high bandwidth, crisp response, and reliability for the resource-limited IoT devices. Therefore, in this paper, we proposed a three-tier edge-cloud integration architecture and utilized a classical computation offloading technique for seamless task offloading process. A metaheuristic and nature-inspired ABC optimization technique is used to balance the workload over edge servers while considering network latency and service rate of the edge servers. The numerical results exhibit that the proposed ABC algorithm produced notable improvement in response time of IoT applications. In addition, the results also ensure that the workload over edge servers is managed effectively, which scales the edge server for entertaining maximum number of offloaded tasks.

In the future, we aim to extend this work to LTE and 5G communication architecture using multiobjective optimization, including communication cost and energy consumption cost.

Data Availability

The data used to support the findings of this study are included within the article. The used simulations software and its details are mentioned in the Results section that help to reach conclusions.

Disclosure

Mohammad Babar and Farman Ali are the co-first authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Mohammad Babar and Farman Ali contributed equally to this work.

Acknowledgments

This work was supported by the National Research Foundation of Korea-Grant funded by the Korean Government (Ministry of Science and ICT, NRF-2020R1A2B5B02002478).

References

[1] K. Sha, T. A. Yang, W. Wei, and S. Davari, "A survey of edge computing-based designs for iot security," *Digital Communications and Networks*, vol. 6, no. 2, pp. 195–202, 2020.

[2] L. U. Khan, I. Yaqoob, N. H. Tran, S. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge computing enabled smart cities: a comprehensive survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10200–10232, 2020.

[3] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: a survey," *Mobile Networks and Applications*, vol. 25, pp. 1–24, 2020.

[4] C. K. M. Lee, Y. Z. Huo, S. Z. Zhang, and K. K. H. Ng, "Design of a smart manufacturing system with the application of multi-access edge computing and blockchain technology," *IEEE Access*, vol. 8, pp. 28659–28667, 2020.

[5] W. Yan, Z. Wang, H. Wang, W. Wang, J. Li, and X. Gui, "Survey on recent smart gateways for smart home: systems, technologies, and challenges," *Transactions on Emerging Telecommunications Technologies*, vol. 31, Article ID e4067, 2020.

[6] I. Lee and K. Lee, "The internet of things (IoT): applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[7] M. Babar, M. S. khan, F. Ali, M. Imran, and M. Shoaib, "Cloudlet computing: recent advances, taxonomy, and challenges," *IEEE Access*, vol. 9, pp. 29609–29622, 2021.

[8] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. ur Rasool, and W. Dou, "Complementing IoT services through software defined networking and edge computing: a comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1761–1804, 2020.

[9] K. Jain and S. Mohapatra, "Taxonomy of edge computing: challenges, opportunities, and data reduction methods," *Edge Computing*, Springer, New York, NY, USA, pp. 51–69, 2019.

[10] M. K. Hussein and M. H. Mousa, "Efficient task offloading for IoT-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020.

[11] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, vol. 169, Article ID 102781, 2020.

[12] Q.-H. Nguyen and F. Dressler, "A smartphone perspective on computation offloading–a survey," *Computer Communications*, vol. 159, pp. 133–154, 2020.

[13] F. Wei, S. Chen, and W. Zou, "A greedy algorithm for task offloading in mobile edge computing system," *China Communications*, vol. 15, no. 11, pp. 149–157, 2018.

[14] F. Wang, J. Xu, and Z. Ding, "Multi-antenna NOMA for computation offloading in multiuser mobile edge computing systems," *IEEE Transactions on Communications*, vol. 67, no. 3, pp. 2450–2463, 2018.

[15] T. Zheng, J. Wan, J. Zhang, C. Jiang, and G. Jia, "A survey of computation offloading in edge computing," in *Proceedings of the 2020 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pp. 1–6, Hangzhou, China, October 2020.

[16] F. Ali, S. El-Sappagh, S. M. R. Islam et al., "An intelligent healthcare monitoring framework using wearable sensors and social networking data," *Future Generation Computer Systems*, vol. 114, pp. 23–43, 2021.

[17] F. Ali, S. El-Sappagh, S. M. R. Islam et al., "A smart healthcare monitoring system for heart disease prediction based on ensemble deep learning and feature fusion," *Information Fusion*, vol. 63, pp. 208–222, 2020.

[18] C. Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, "Toward computation offloading in edge computing: a survey," *IEEE Access*, vol. 7, pp. 131543–131558, 2019.

[19] G. Orsini, D. Bade, and W. Lamersdorf, "Context-aware computation offloading for mobile cloud computing: requirements analysis, survey and design guideline," *Procedia Computer Science*, vol. 56, pp. 10–17, 2015.

[20] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: towards minimizing delay in the internet of things," in *Proceedings of the 2017 IEEE International Conference on Edge Computing (EDGE)*, pp. 17–24, Honolulu, HI, USA, June 2017.

[21] J. Luo, X. Deng, H. Zhang, and H. Qi, "QoE-driven computation offloading for edge computing," *Journal of Systems Architecture*, vol. 97, pp. 34–39, 2019.

[22] H. Fouad, N. M. Mahmoud, M. S. E. Issawi, and H. Al-Feel, "Distributed and scalable computing framework for improving request processing of wearable IoT assisted medical sensors on pervasive computing system," *Computer Communications*, vol. 151, pp. 257–265, 2020.

[23] Y. Chen, N. Zhang, Y. Zhang, and X. Chen, "Dynamic computation offloading in edge computing for Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4242–4251, 2019.

[24] Z. Zhao, R. Zhao, J. Xia et al., "A novel framework of three-hierarchical offloading optimization for MEC in industrial IoT networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5424–5434, 2020.

[25] H. Zhao, S. Deng, C. Zhang, W. Du, Q. He, and J. Yin, "A mobility-aware cross-edge computation offloading framework for partitionable applications," in *Proceedings of the 2019 IEEE International Conference on Web Services (ICWS)*, pp. 193–200, Milan, Italy, July 2019.

[26] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: a deep learning approach," in *Proceedings of the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–6, Montreal, Canada, October 2017.

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

[28] 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," *Applied Sciences*, vol. 9, no. 9, p. 1730, 2019.

[29] 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 *Proceedings of the 2018 Ninth International Symposium on Information and Communication Technology*, pp. 397–404, Da Nang, Vietnam, December 2018.

[30] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[31] S. Azimi, C. Pahl, and M. H. Shirvani, "Particle swarm optimization for performance management in multi-cluster IoT edge architectures," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science, CLOSER 2020*, pp. 328–337, Prague, Czech Republic, February 2020.

[32] M. Bouet and V. Conan, "Mobile edge computing resources optimization: a geo-clustering approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 787–796, 2018.

[33] R. Bruschi, F. Davoli, P. Lago, and J. F. Pajo, "A multi-clustering approach to scale distributed tenant networks for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 499–514, 2019.

[34] R. Yu, J. Ding, S. Maharjan, S. Gjessing, Y. Zhang, and D. H. Tsang, "Decentralized and optimal resource cooperation in geo-distributed mobile cloud computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 1, pp. 72–84, 2015.

[35] R. Yu, X. Huang, J. Kang et al., "Cooperative resource management in cloud-enabled vehicular networks," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 12, pp. 7938–7951, 2015.

[36] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2015.

[37] W. Labidi, M. Sarkiss, and M. Kamoun, "Joint multi-user resource scheduling and computation offloading in small cell networks," in *Proceedings of the 2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 794–801, Abu Dhabi, UAE, October 2015.

[38] A. M. Ortiz, D. Hussein, S. Park, S. N. Han, and N. Crespi, "The cluster between internet of things and social networks: review and research challenges," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 206–215, 2014.

[39] D. Karaboga, "Artificial bee colony algorithm," *Scholarpedia*, vol. 5, no. 3, p. 6915, 2010.