Computers and Electrical Engineering 102 (2022) 108123

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

A Bacterial Foraging Based Smart Offloading for IoT Sensors in Edge Computing

Mohammad Babar [a,*], Ahmad Din [a], Ohoud Alzamzami [b], Hanen Karamti [c], Ahmad Khan [a], Muhammad Nawaz [d]

[a] Department of Computer Science COMSATS University Islamabad Abbottabad Campus, Pakistan
[b] Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
[c] Department of computer sciences, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, P.O.Box 84428, Riyadh 11671, Saudi Arabia.
[d] Institute of Management Sciences Peshawar, Pakistan.

ARTICLE INFO

Keywords:
Internet of things
Smart sensors
Edge computing
computation offloading
and resource scheduling

ABSTRACT

The main aim of edge computing is to facilitate the low power computational IoT sensors in processing heavy tasks. Sensors should be enough efficient to offload the required task and get back the computation results in the stipulated time. However, acquiring the desired quality of services (QoS) is a challenging task. Therefore, efficient resource management is inevitable to reduce the response time and communication cost. In this paper, a computation offloading algorithm based on nature-inspired multi-objective bacterial foraging optimization (MO-BFO) is proposed for load management over edge servers. The performance of the proposed MO-BFO algorithm is quantitatively evaluated against standard ant colony optimization (ACO), particle swarm optimization (PSO), and round-robin (RR) scheduler. The detailed results show that the proposed algorithm reduces the response time, communication cost, and ensures effective load management compared to its counterparts.

1. Introduction

In the current decennium, the sensor-based Internet of Things (IoT) is a source of attraction for both academia and industry. It transforms the traditional systems such as cities, transport, agriculture, and industries into smart cities, intelligent transportation, smart agriculture, and smart industry [1]. This transformation makes humans free from mental drudgery and leads to intelligent decisions with financial gain [2].

The Internet of vehicular things (IoVT), the internet of everything (IoE), and other IoT-based systems are revolutionizing the industry [3]. But the limited battery power, low computational capability, and limited lifetime of these smart IoT sensors face challenges to process real-time heavy tasks [4,5]. These systems are ineffective in a standalone capacity and thus need a potential solution.

One of the well-sought solutions to overcome these challenges is the cloud. The cloud has unlimited storage, huge processing power, and accommodates millions of applications to centrally process the IoT sensors requests [6]. The cloud centralized architecture leads to longer latencies due to physical and logical distances which makes it unviable to deal with IoT sensors [7].

It is important to bring the service provider closer to the user to improve the latency and response time. Thus, edge computing is

* Corresponding author.
E-mail address: mbabarcs@gmail.com (M. Babar).

https://doi.org/10.1016/j.compeleceng.2022.108123
Received 10 February 2022; Received in revised form 17 May 2022; Accepted 24 May 2022
Available online 1 July 2022

introduced to provide services to the user in a single hope distance [8]. The advent of edge computing with associated IoT brings the perception of smart systems into reality.

Edge computing provides the computation offloading facility to address the low computational issue of IoT sensors [9,10]. In computation offloading, a heavy task is transferred to the edge server for processing and the results are sent back to the offloading device. Thus, the offloading mechanism enables the IoT sensors to deal with complex tasks, saves energy, and extends their operational time [11].

Despite the advantages and successful implementation of computation offloading in many systems, it still faces many challenges such as task partitioning, synchronization, node discovery, resource provision and allocation, and dealing with information from external environments [12]. Thus, it is imperative to design an effective offloading technique to address the aforementioned challenges. This research focuses to devise a new offloading algorithm to accomplish the following objectives:

- To ensure seamless computation offloading in order to satisfy the stringent QoS requirements in delay-sensitive tasks.
- To effectively deal with the important parameters such as degree imbalance, response time, standard deviation, and communication costs to manage workload over the edge servers
- To effectively deal with a complex environment where the number of edge servers and IoT sensor nodes exponentially increases over time.

In this paper, a nature-inspired multi-objective bacterial foraging optimization (MO-BFO) based computation offloading algorithm is proposed for load management over edge servers to achieve the above-mentioned objectives. The performance of the proposed MO-BFO algorithm is quantitatively evaluated against standard ant colony optimization (ACO), particle swarm optimization (PSO), and round-robin (RR) scheduler

The rest of the article is arranged as follows: Section 2 provides a deep insight into the recent related work while Section 3 contains the proposed methodology. The experimental setup, detailed results, simulation testbed, and critical analysis are given in Section 4. Finally, the research work is concluded in Section 5.

2. Related Work

Computation offloading is considered a game-changer for IoT. Where a resource-poor IoT device offloads a complex task for processing to the edge server [13]. Computation offloading is divided into three different categories i.e. application offloading, component offloading, and virtual machine (VM) image offloading [14]. In application offloading an entire application is offloaded for processing to the edge. Whereas, in component offloading an entire class, method, or thread whichever is compute-intensive is offloaded to the edge for processing. In the VM image offload model, an image is migrated to the edge for processing [15].

In [16], the researchers analyze the computation offloading technique and its performance into two application types i.e. delay-sensitive and delay-tolerant. It has been concluded that the offloading decision is of vital importance. A well-thought computation offloading design reduces latency and saves energy, but an ineffective one can produce longer delays. Some of the considering points for a realistic computation offloading decision are latency requirements, network bandwidth, storage, load balancing, security, privacy, and energy efficiency are highlighted in their study [17].

Computation offloading in IoT remains a focus in numerous studies. An offloading technique discussed in [18] is deployed in an IoT environment that explores the impact of the response time of the application on computation offloading. The study also highlights a new parameter i.e., the ratio of input and output generation. An IoT-edge architecture presented in [10] focused on component offloading techniques to study the role of application partitioning, resource scheduling, and distributed execution. However, these investigations finding hardly consider the execution cost.

A resource allocation-based offloading algorithm is presented in [19] for the IoT environment. The compute-intensive task is processed on external edge devices and investigates the financial impact of computation power over offloading technique. A linear programming-based cyber-physical system [20] for smart health has been deployed. The study considers cellular nodes as fog nodes, which reflects an unrealistic offloading environment. Furthermore, Load balancing across edge nodes is not taken into account. Studies [21,22] also consider the computation cost in the context of IoT and produce desirable results, though focused on partial offloading schemes.

Several articles encompass computational offloading frameworks. These ensure the effective resource management of the available edge resources using different optimization techniques. Convex optimization is used in [23], and a game theory optimization to deal with a complex task like computation offloading decision. These frameworks obtained only suboptimal results in the MEC environment as it deals with the time-varying characteristics of the channel. It was also eminent that the longer communication and offloading overhead made them impractical.

A cross entropy-based scalable edge computing framework [24] has implemented the computation offloading separately at the IoT layer and edge layer to reduce the offloading and communication overhead. The end-to-end network latency is minimized, and the energy consumption of the computational nodes is also reduced. However, an increasing number of offloading requests quickly scale out the computation servers.

A particle swarm-based computation offloading technique is discussed in [11] to reduce the latency and communication cost of compute-intensive tasks using multi-objective optimization. The studies design a framework that is based on communication and computation cost, before optimizing the delay and computation cost of the offloading decision. Through detailed simulation, the offloading algorithm obtained a Pareto-optimal solution to the multi-objective optimization problem. It has been observed that the

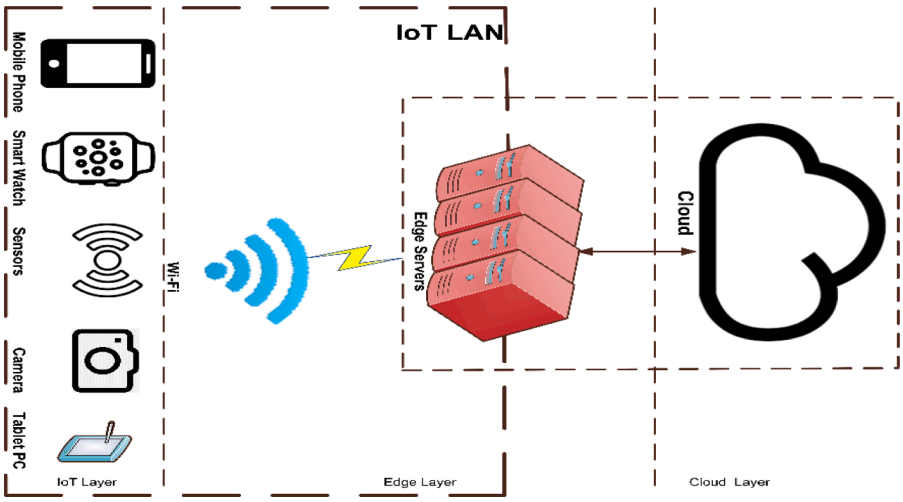

Fig. 1. A three-layer edge computing architecture.

Table 1
Annotation used in the system model.

Symbol	Meaning
d_i	IoT device / sensor number i
L^e	Computation time
L^o	Offloading latency
L^c	Communication time
h_{edge}	Computational power of edge device
λ_i	Frequency of sensor
P_{sen}	Data distribution generated by an IoT device
B_{LAN}	Bandwidth of LAN
P_{data}	Size of data transmitted
q_j	Offloading plan j
V_I	Power consumed by IoT in idle state
V_t	Energy required to transmit task j
$C(i)$	Step size
M_g	Number of generations
p	Search space dimension
L_j	Total execution time of task t_j

studies produce optimal results using a smaller number of computations offloading requests and their performance degrades as the number of offloading requests increases.

This research article deploys a three-tier hierarchical architecture using an IoT layer, an Edge layer, and a resource-rich cloud. Where a seamless computation offloading technique is introduced into it. The performance of the proposed multi-objective bacterial foraging optimization-based offloading algorithm is evaluated. We consider the response time of the application using loading balancing over edge servers in an IoT environment. The parameters such as communication cost, load balancing, standard deviation, degree of imbalance, and communication time are considered.

3. System Model

In this paper, the Multi-objective BFO (MO-BFO) algorithm has been deployed for computation offloading in an IoT-edge environment. Aim of this proposed offloading strategy is to reduce the task's service time, allowing devices to respond faster. IoT-Edge infrastructure has three layers: (1) IoT layer: a large number of smart IoT devices are connected to the LAN in clusters. (2) Edge Layer: Smart IoT devices of the IoT layer are connected to the edge layer. This layer consists of several edge nodes including a master node. Furthermore, these nodes can perform rudimentary analysis of data received form smart sensors due limited computational and storage resources.

The master node decides whether to move the task to (3) cloud tires or not. The cloud layer performs intensive analytics and provides massive/long-term data storage. Fig. 1 shows a three-layer architecture.

The IoT devices are virtually clustered based on location, ownership, and services. The offloading tasks are randomly generated from multiple IoT devices using multitasking nature. These offloading tasks are classified as delay-sensitive and delay-tolerant. The threshold for the delay-sensitive and delay-tolerant tasks is 100ms and 150ms respectively. The clustering technique at the IoT tier

controls the number of tasks offloaded to the edge tier. The delay-sensitive tasks are offloaded to the edge server for computation on priority, while the delay-sensitive tasks are entertained on a first come first serve basis.

Consider a scenario illustrated in Fig. 1, there are n IoT devices expressed as $D = \{d_1,d_2,d_3,\ldots\ldots\ldots d_n\}$, where each sensor d_i is working at frequency λ_i. The network latency in computation offloading is considered, as well as the computing time of the processing jobs and the data transmission time between the computing tasks. Thus, the task execution time is categorized into three categories, (a) computation time L^e, (b) the offloading latency L^o, and the communication time L^c. Table 1 lists all the notations used in this system model.

Computation time (L^e) for a task is computed using equation Eq. (1), if it is executed locally on a node:

$$L^e_{local} = \frac{w_{comj}}{h_{local}} \tag{1}$$

Where L^e_{local} is a computation time if a task is executed locally at the IoT node. Task j requires w_{comj} computational power, while IoT smart sensor needed h_{local} computational power. Computation time for the task can be computed using the following equation if it is executed at the edge node:

$$L^e_{edge} = \frac{w_{comj}}{h_{cdge}} \tag{2}$$

Where L^e_{edge} is a computation time if a task is executed at the edge node. Task j requires w_{comj} computational power, while edge device needed h_{edge} computational power. Computation time for the task can be computed using the following equation if it is executed at the cloud server:

$$L^e_c = \frac{w_{comj}}{h_c} \tag{3}$$

Where L^e_c is a computation time if the task is executed at the cloud server. Task j requires w_{comj} computational power, while the cloud server needed h_c computational power. For a job t_j, execution time $L^e(t_j)$ can be computed by using the following:

$$L^e(t_j) = \begin{cases} L^e_{local}, t_j = 0 \\ L^e_{edge}, t_j = 1 \\ L^e_c, t_j = 2 \end{cases} \tag{4}$$

The transmission cost between the smart sensor and edge device depends on the bandwidth of the LAN. It is computed using Eq. (5).

$$L^e_{edge} = \frac{P_{sen}}{B_{LAN}} \tag{5}$$

where P_{sen} is a data distribution generated by an IoT sensor. While B_{LAN} is the bandwidth of the LAN. The communication cost between the edge node and the cloud server is computed using Eq. (6), which depends on the bandwidth of the WAN.

$$L^c_{local} = \frac{P_{data}}{B_{WAN}} \tag{6}$$

Here P_{data} is the size of the data being transmitted. While B_{WAN} is the bandwidth of the WAN. However, the communication cost in the same computation environment is negligible. For the computation job t_j, communication cost $L^e(t_j)$ is computed by using the following:

$$L^c(t_j) = \begin{cases} 0, \ \textit{if task j is transmitted at the same layer} \\ L^c_{edge}, \ \textit{if task j is offloaded to the edge node} \\ L^c_{cloud}, \ \textit{if task j is offloaded to the cloud server} \end{cases} \tag{7}$$

For the computation task t_j, if the offloading plan q_j is employed, the offloading latency L^ocan be computed by

$$L^o(q_j) = \begin{cases} 0, q_j = 0 \\ L^o_{LAN}, q_j = 1 \\ L^o_{WAN}, q_j = 2 \end{cases} \tag{8}$$

where L^o_{LAN} is the latency of the LAN, while L^o_{WAN} is the latency of the WAN. However, latency at the same layer is negligible. Hence, the total execution time for a task t_j can be computed by Eq. (9):

$$L_j = L^o(q_j) + L^c(t_j) + L^e(t_j) \tag{9}$$

If a task t_j is offloaded using an offloading plan q_j, then the energy required for the computation of the task can be computed using the following equation:

$$E^o(q_j) = L^o(q_j) \times V_I \tag{10}$$

Algorithm 1
BFO Algorithm.

```
Initialization phase
repeat:
    Step 1: Chemotaxis Phase
        1(a): Tumble
        1(b): Swim
    Step 2:  Production Phase
    Step 3:  Elimination-Dispersal phase
until Max # of iterations reached
Output:  Best solution found
```

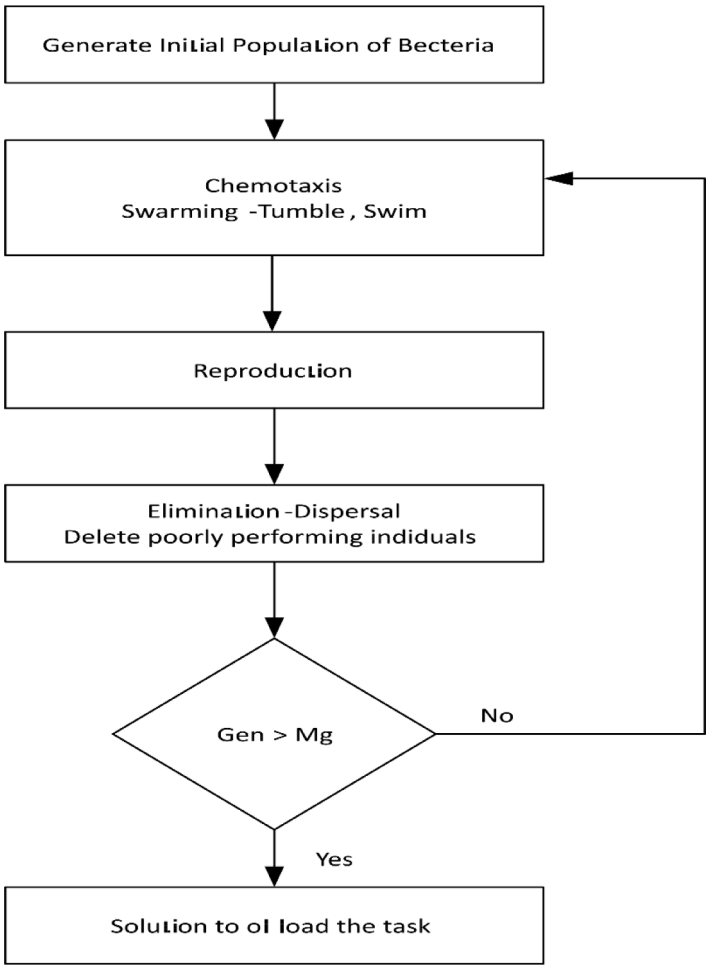

Fig. 2. Flow Diagram of BFO enabled Computation Offloading Algorithm.

where $V_I$is the power consumed by the IoT device when it is in an idle state. If the job is executed by an IoT device locally then there will be no offloading energy cost. However, if a job is executed locally then the energy required to complete this task can be computed by the following equation.

$$E^o(q_j) = \begin{cases} L^e_{local} \times V_{U,} q_j = 0 \\ L^e_{edge} \times V_{I,} q_j = 1 \\ L^e_{cloud} \times V_{I,} q_j = 2 \end{cases} \tag{11}$$

The energy required for the transmission of the task among the layers can be computed as follow:

$$E^c(q_j) = L^c(t_j) \times V_t \tag{12}$$

where V_t is the energy required for the transmission of the task j. Hence the total energy required for the task t_j.

$$E_j = E^o(q_j) + E^c(t_j) + E^e(t_j) \tag{13}$$

3.1. Objective Function

Two main decisions regarding offloading a job in hand to the edge device or cloud server should be modeled properly. To this end, the objective is to minimize service time/delay (L) of each offloading scheme, and energy consumption E, simultaneously. It can be characterized as a multi-objective optimization problem as follows:

$$min(L_j, E_j)$$

$$1 \leq j \leq M$$

$$s.t..\frac{\sum^k H_k}{H_t - H_{avg}} \leq E_{cap} \tag{14}$$

where E_{cap} is the capacity of edge device d, H_k is an average of response-time if the edge layer has only one device to perform the computation of all tasks offloaded to the edge.

3.2. Bacterial Foraging Optimization (BFO)

BFO is a foraging behavior-based swarm intelligence (SI) optimization algorithm. Besides a chemotactic strategy, it has other foraging behavior of bacteria including reproduction, swarming, dispersal, and elimination. A high-level outline of the BFO is depicted in Algorithm 1 and a holistic overview of the computation offloading algorithm is represented in Fig. 2.

In this section, we discuss the computation-offloading algorithm based on multi-objective BFO. The objective functions deal with the energy required to complete the task and service time. Furthermore, the service time is composed of computation time and network delays. This multi-objective optimization aims to discover a solution that could minimize overall network latency and computational cost. The inputs of the proposed algorithm are computational devices (at each layer) and a set of tasks. The decision variables are L_i E_j and ξ (contains a list of jobs).

3.2.1. Chemotaxis Phase

Bacteria move using their flagellum. A bacterium moves in random directions to find nutrients. This random movement is accomplished by alternatively tumbling and swimming. It can generate these modes by simultaneously moving flagella clockwise and counter-clockwise moving clockwise and counter-clockwise. During the course, if a bacterium comes across a nutrient gradient, it swims more than tumbling. This random search can be modeled using the following equation.

$$\sigma(i) = \frac{\Delta(i)}{\sqrt{\Delta(i)^T \Delta(i)}} \tag{15}$$

where $\Delta(i)^p$ is a randomly generated vector of p-dimension. Its elements in within intervals of -1 and 1. The position of i^{th} bacterium is defined by $\varnothing^i(j,k,l)$. The j, k, and l represent counters for Chemotaxis, reproduction, and elimination and dispersal respectively. The bacterium adjusts its position using the following equation:

$$\varnothing^i(j+1,k,l) = \varnothing^i(j,k,l) + C(i)\sigma(i) \tag{16}$$

where parameter $C(i)$ is a step size. A tumble that generates search direction is modeled using Eq. (15). A swim defines the movement component. It can be computed using Eq. (16). If the new position is better than the previous position means that the bacterium has encountered a nutrient gradient, therefore, the swim will be repeated N_s times.

3.2.2. Reproduction Phase

After the chemotaxis phase, some of the bacteria may have performed poorly in searching for nutrients. Therefore, it is essential to get rid of some poorly performing agents. BFO is using reproduction to remove deficient individuals and to retain good agents. It helps to speed up the search for Pareto-front. For this purpose, the lifetime fitness of each bacterium is computed using the fitness function. The population of bacteria is sorted based on the fitness of individuals. Poor half of the population is deleted.

3.2.3. Elimination and Dispersal Phase

In a swarm foraging, with a change in a food source, individuals are required to be dispersed. Therefore, the dispersal step is performed after a few generations e.g., after the k reproduction step (N_{re}). Each individual is dispersed with a probability of $0 \leq P_{ed} \leq 1$. The individual who chooses to disperse is eliminated and a new agent is generated.

Algorithm 2
MO-BFO algorithm to offload computation to Edge / Cloud.

Step 1 : Initialization
p ← search space dimension
S ← # of bacteria
S_r ← # of bacteria for reproduction
r ← scaling factor
M_g ← # of generations
N_s ← swim
$C(i)$ ← The length of run for each run or tumble
Step 2: Generate random initial population of bacteria $\varphi^i(j;G)\forall(i=1;2;\cdots;S)$
Step 3: Evaluate each bacterium $\varphi^i(j;G)\forall(i=1;2;\cdots;S)$
Step 4: $Gen \leftarrow 1, k \leftarrow 1, j \leftarrow 1$
$whileGen < M_g do$
$whilek < Sdo$
$whilej < N_{hc} do$
Step 5: Chemotactic Step – perform the following each bacterium in S:
5(a): Compute fitness using Eq (20)
5(b): save last value, it may help to find better values through a run
5(c): Tumble: Generate p-dimensional vector $\Delta(i)^p$ in ranges 1 and -1
5(d): Move: compute the new position of bacterium i using equation # 18
5(e): Compute fitness using Eq (20)
5(f): Swim: let $n = 0$
$whilen < N_{swim} do$
$n = n + 1$
If $J_{current} < J_{last}$ **then**
$J_{last} = J_{current}$
compute new position of bacterium i using Eq (16) and $\emptyset^i(j+1,k,l)$
else:
$n = N_{swim}$
end if
end while
$j = j + 1$, $i = i + 1$, and go to set 5(a)
end while
Step 6: Reproduction Step-
6(a): Calculate the health value of i^{th} bacterium using eq (21)
6(a) sort all individuals using Pareto non-dominance
6(b) Eliminate $S_r = \frac{S}{2}$ poorly performing bacteria based on non-dominated sorting
6(c) duplicate remaining good individuals
end while
Step 6: Elimination-Dispersal Step – generate a non-dominated set of solutions based on fitness in previous steps. Replace old solutions with a new solution if new solutions are improved.
end while
Output the best solution identified

3.2.4. Multi-objective BFO Algorithm

Single objective problems could be solved using BFO, as it can search single optimum solution. However, the multi-objective problem can be approached using an extension of the BFO algorithm called MO-BFO. It discovers a set of all possible values that satisfies all objective functions. To this end, the concept of Pareto dominance is integrated to find only non-dominated solutions. This non-dominated set of solutions would be close to the real Pareto front.

Bactria moves along the nutrient gradient as a swarm. Swarming behavior is widespread in nature and multi-agent systems. Bacteria for instance E. coli have highly sensitive sensors to detect protein molecules and nutrient profiles. Once it is signaled by a high level of nutrients, it releases attractant, which triggers aggregate behavior, then bacteria swarm together. Otherwise, it releases a

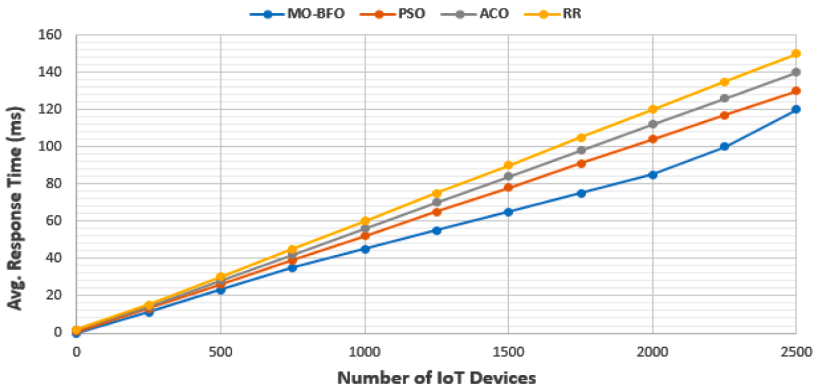

Fig. 3. Avg. response time of the offloaded task using a single server.

repellent signal to avoid an unfavorable environment. The swarming behavior of bacteria could be modeled using attraction and repulsion functions.

$$J_{BB}(\varnothing, Q(j,k,l)) = \sum_{i=1}^{S} J_{BB}^{i}\left(\varnothing, \varnothing^{i}(j,k,l)\right) = J_a^S + J_r^S \tag{17}$$

where, $J_{BB}(\varnothing,Q(j,\ k,\ l))$ is the time-varying cell-to-cell attraction-repulsion cost function. $Q(j,\ k,\ l)$ represents the position of the bacterium. The cost for this bacterium $\varnothing^{i}(j,k,\ l)$ at this position is computed using $J_{\varepsilon}(i,j,\ k,\ l)$. While S is the population of bacteria. J_a^S is a total attraction force.

$$J_a^S = \sum_{i=1}^{S}\left[- d_a exp\left(-w_a \sum_{n=1}^{q}\left(\varnothing_n - \varnothing_n^i\right)^2\right)\right] \tag{18}$$

where, $d_a$and w_a are the depth and width of attraction signal released by bacteria, respectively. While q represents several parameters in the cost function required to be optimized. J_r^S is a total repulsive force.

$$J_r^S = \sum_{i=1}^{S}\left[- h_r exp\left(-w_r \sum_{n=1}^{q}\left(\varnothing_n - \varnothing_n^i\right)^2\right)\right] \tag{19}$$

where, h_r and w_r is the height and width of repulsive signal release by bacteria, respectively. There are two cost functions to be solved for each function.

$$J(i,j,k,l) = J_{\varepsilon}(i,j,k,l) + J_{BB}\left(\varnothing^{i}(j,k,l), Q(j,k,l)\right) \tag{20}$$

The total health of bacterium i during chemotaxis is computed using the following equation.

$$J_i^{health} = \sum_{j=1}^{N_{hc}} J(i,j,k,l) \tag{21}$$

where N_{hc} is some steps for chemotaxis. Bacteria are sorted based on J_i^{health}. Bacteria with the lowest health value are given a higher chance to survive. Like other population-based algorithms, diversity in the population is essential for convergence in MO-BFO. However, a reproduction based on the Pareto front may reduce diversity. Therefore, to ensure diversity in the population a dominant solution cannot be to eliminate more than one-quarter from a particular group. The result of MO-BFO can be a set of non-dominated solutions, a decision-maker module selects the required number of solutions randomly. The proposed algorithm is illustrated in Algorithm 2.

4. Experimental Results

This section briefly describes the results achieved in this research study. We have evaluated the performance of the Multi-objective BFO-based computation offloading algorithm in a classical edge enabled IoT setup. The simulation setup is developed using MATLAB. We have kept the environment generic that depict a common architecture adopted in most of the IoT-based smart systems. It consists of sensors, mobile phones, and other handheld devices that make a viable IoT system. The simulation-based research study eases the researchers to run the simulation redundantly under different scenarios and keep the parameters used in the simulation under control.

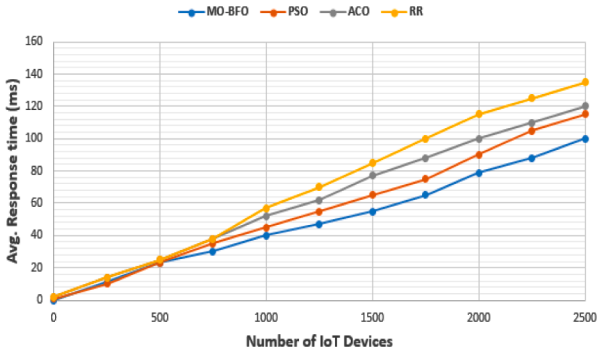

Fig. 4. Avg. response time of the offloaded task using 4 servers.

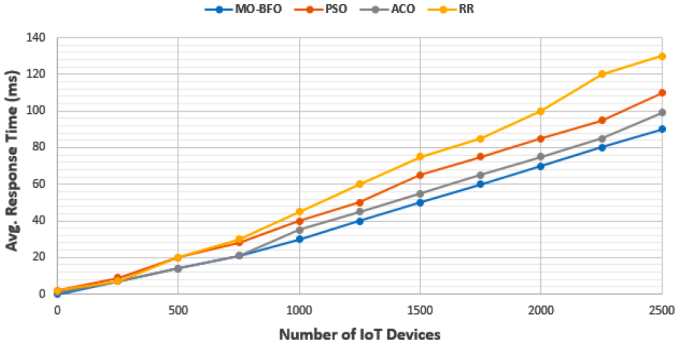

Fig. 5. Avg. response time of the offload task using 2 server.

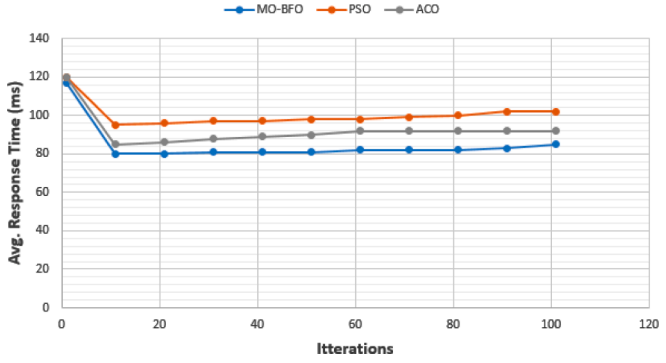

Fig. 6. Performance comparison of the objective function using a single run.

The hardware used to run the simulation includes a PC having a processing capability of 2.60 GHz Intel Core i7 and primary memory of 8 GB. The number of IoT nodes ranges from 500 – to 2500 comprises of sensors and other IoT nodes. The number of edge servers is 1-4, wherein in scenario 1, we have deployed a single edge server. In scenario 2, we have increased the computational resources to two edge servers, and 4 servers in scenario 3. The task size for offloading ranges from 250Kb to 1MB which includes computationally intensive code for execution. The performance of the MO-BFO computation offloading algorithm is tested using these scenarios. In our experimental setup, we increase the number of IoT devices from 2000 to 2500 to satisfy the requirements of a standard smart system. The delay-sensitive tasks consider a threshold of 100ms and a delay-tolerant of 150ms respectively.

In this experiment, we have examined the MO-BFO algorithm against the standard meta-heuristic nature-inspired swarm intelligence techniques namely PSO, ACO, and RR. We have considered the following parameters i.e., average response time, the standard deviation, degree of imbalance, and average response time of the single run of the iteration of the observed algorithms.

The average response time of the offloaded computation is shown in Fig. 3. The statistics obtained via scenario 1 reflect that the MO-BFO algorithm produces the lowest average response time compared to ACO, PSO, and RR scheduler and satisfies the QoS requirements of the delay-sensitive tasks. However, a rapid increase in the response time is witnessed as the number of IoT devices

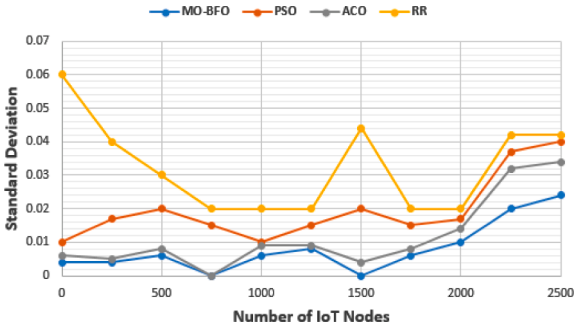

Fig. 7. Standard Deviation of Response time on the edge server.

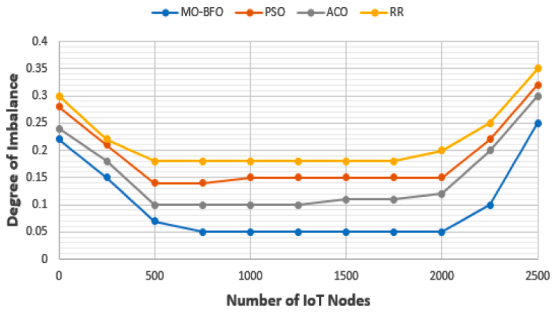

Fig. 8. Degree of imbalance in offloaded tasks.

approaches 2200. This is because the edge infrastructure utilizes a single server and therefore quickly scales out.

Figs. 4 and 5, highlight the average response time of the offloaded tasks computed by the edge server i.e., two and four respectively. It is observed that the proposed MO-BFO algorithm outperforms the PSO, ACO, and RR scheduling algorithms as it converges in a short time to find the right resource for the execution of the offloaded task. The MO-BFO algorithm minimizes the average response time for delay-sensitive tasks compared to others. A 100ms response time using two computational resources with a density of 2500 IoT devices is recorded and none of the offloaded tasks violates the strict latency requirements. The performance of the computational offloading algorithm produces the lowest response time of 85ms which is good enough for delaying critical applications and applications relating to smart systems.

The performance comparison of the MO-BFO, PSO, and ACO is given in Fig. 6. Our findings discovered that the proposed BFO-based computation algorithm explores the search space quickly as the bacterium swims through the search space and reaches the best value for the objective function hastily. This behavior reflects the faster convergence as MO-BFO reaches the best solution on the 15th iteration than the PSO on 21st and ACO on the 18th iteration.

Fig. 7 explains the statistics of standard deviation using defined scenarios with an increased number of IoT devices and offloaded tasks. The standard deviation is acquired through observing all the offloaded tasks with variations in response time to deduce the average response time. MO-BFO algorithm produces improved standard deviation in comparison to PSO, ACO, and RR. However, swarm intelligence-based optimization technique such as PSO, ACO, and BFO remains persistent in obeying QoS requirements for delay-critical applications.

The workload of the edge server is depicted in Fig. 8, which elaborates the performance comparison of MO-BFO, PSO, ACO, and RR scheduling algorithms. The proposed BFO-based algorithm maintains a straight line with an increased number of IoT devices in the graph, which resembles that it upholds smaller numbers. Therefore, indicating the steadiness in workload distribution amongst the edge servers as compared to the listed scheduling algorithms. This feature shows the ability of edge-enabled architecture to support a large number of IoT devices and effective resource management over edge server scales to compute the growing number of offloading requests.

5. Conclusion

Edge computing exploits its proximity feature to deal with a large amount of data and offers data analytics, storage, and content caching services. To deal with real-time delay-sensitive tasks the IoT-based systems require low response time, which is challenging to achieve. This research study focuses on the design of an IoT-based smart system which uses a variety of heterogeneous IoT sensors that tends to produce huge data instantaneously. A three-tier edge-based architecture for the seamless computation offloading process is designed using MO-BFO. The quantitative results show that the proposed algorithm reduces the response time, minimizes communication cost, and efficiently deals with edge server resources load management. The proposed algorithm is able to serve the delay-

sensitive tasks under the defined threshold, which reflects its suitability for applications with stringent QoS requirements. In the future, this research work will be extended to investigate the energy consumption cost of IoT devices in a multiuser and multitasking environment.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2022R192), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia

References

[1] Xue H, Huang B, Qin M, Zhou H, Yang H. Edge Computing for Internet of Things: A Survey. In: 2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics); 2020. p. 755–60.
[2] Varghese B, Wang N, Nikolopoulos DS, Buyya R. Feasibility of fog computing. In: Handbook of Integration of Cloud Computing, Cyber Physical Systems and Internet of Things. Springer; 2020. p. 127–46.
[3] Jiang X, Yu FR, Song T, Leung VC. A Survey on Multi-Access Edge Computing Applied to Video Streaming: Some Research Issues and Challenges. IEEE Commun Surv Tut 2021;23(2):871–903.
[4] HaddadPajouh H, Dehghantanha A, Parizi RM, Aledhari M, Karimipour H. A survey on internet of things security: Requirements, challenges, and solutions. Internet of Things 2021;14:100129.
[5] Hansen EB, Bøgh S. Artificial intelligence and internet of things in small and medium-sized enterprises: A survey. J Manuf Syst 2021;58:362–72.
[6] Welsh T, Benkhelifa E. On resilience in cloud computing: A survey of techniques across the cloud domain. ACM Comput Surv (CSUR) 2020;53(3):1–36.
[7] F. Firouzi, B. Farahani, and A. Marinšek, "The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT)," Information Systems, p. 101840, 2021.
[8] Babar M, Khan MS, Ali F, Imran M, Shoaib M. Cloudlet Computing: Recent Advances, Taxonomy, and Challenges. IEEE Access 2021;9:29609–22.
[9] Khan WZ, Ahmed E, Hakak S, Yaqoob I, Ahmed A. Edge computing: A survey. Future Gener Comput Syst 2019;97:219–35.
[10] Lin L, Liao X, Jin H, Li P. Computation offloading toward edge computing. Proc IEEE 2019;107(8):1584–607.
[11] Babar M, Khan MS, Din A, Ali F, Habib U, Kwak KS. Intelligent Computation Offloading for IoT Applications in Scalable Edge Computing Using Artificial Bee Colony Optimization. Complexity 2021;2021.
[12] Deng X, Sun Z, Li D, Luo J, Wan S. User-centric computation offloading for edge computing. IEEE Internet Things J 2021.
[13] 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, p. 102781, 2020.
[14] Maleki EF, Mashayekhy L, Nabavinejad SM. Mobility-aware computation offloading in edge computing using machine learning. IEEE Trans Mob Comput 2021.
[15] Ale L, Zhang N, Fang X, Chen X, Wu S, Li L. Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning. IEEE Trans Cogn Commun Netw 2021;7(3):881–92.
[16] Gasmi K, Dilek S, Tosun S, Ozdemir S. A survey on computation offloading and service placement in fog computing-based IoT. J Supercomput 2021:1–32.
[17] Saleh H, Saber W, Rizk R. Mobile Computation Offloading in Mobile Edge Computing Based on Artificial Intelligence Approach: A Review and Future Directions. In: International Conference on Advanced Machine Learning Technologies and Applications; 2022. p. 593–603.
[18] Dai B, Niu J, Ren T, Atiquzzaman M. Towards Mobility-Aware Computation Offloading and Resource Allocation in End-Edge-Cloud Orchestrated Computing. IEEE Internet Things J 2022.
[19] Babar M, Sohail Khan M. ScalEdge: A framework for scalable edge computing in Internet of things–based smart systems. Int J Distrib Sens Netw 2021;17(7). 15501477211035332.
[20] H. K. Apat, K. Bhaisare, B. Sahoo, and P. Maiti, "Energy efficient resource management in fog computing supported medical cyber-physical system," in 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), 2020, pp. 1–6.
[21] Guo Y, Zhao R, Lai S, Fan L, Lei X, Karagiannidis GK. Distributed machine learning for multiuser mobile edge computing systems. IEEE J Sel Top Signal Process 2022.
[22] Alam T, Ullah A, Benaida M. Deep reinforcement learning approach for computation offloading in blockchain-enabled communications systems. J Ambient Intell Humaniz Comput 2022:1–14.
[23] Barbarossa S, Sardellitti S, Di Lorenzo P. Communicating while computing: Distributed mobile cloud computing over 5G heterogeneous networks. IEEE Signal Process Mag 2014;31(6):45–55.
[24] Babar M, Khan MS, Habib U, Shah B, Ali F, Song D. Scalable Edge Computing for IoT and Multimedia Applications Using Machine Learning. Hum-Centric Comput Inf Sci 2021;11.

Mohammad Babar received his master's degree in data telecommunication and networks from the University of Salford, U.K., in 2010. He is currently pursuing the Ph. D. degree from the University of Engineering and Technology Peshawar. His research interests include cloud computing, edge computing, Fog computing, and the Internet of Things.

Ahmad Din is currently assistant professor in COMSATS University Islamabad (CUI), Abbottabad Campus, Pakistan. He received his Ph.D. in Mechatronics / Artificial Intelligence from Politechnico di Torino, Italy. His research interests include collective and swarm intelligence, artificial and natural swarms, machine learning, complex systems, and agriculture robotics.

Ohoud Alzamzami is currently an Assistant Professor at the Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia. She received her Ph.D. degree in Computer Science from the College of Engineering and Computer Science at Florida Atlantic University. Her research interests include IoT, vehicular and wireless sensor networks, and protocol design and optimization.

Hanen Karamti obtained her Ph.D. degree in Computer Science from University of Sfax, Tunisia, in cooperation with the University of La Rochelle (France) and the University of Hanoi (Vietnam). Karamti is assistant professor at Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia. Her areas of interest are multimedia systems, Image retrieval, health informatics, and big data analytics.

Ahmad Khan received the Ph.D. degree from the National University of Computer and Emerging Sciences (FAST- NU), Islamabad, Pakistan, in 2015. He is currently an Assistant Professor with the Department of Computer Science, COMSATS University Islamabad (CUI), Abbottabad Campus. His research interests include computer vision, machine learning, and evolutionary algorithms.

Muhammad Nawaz earned his doctorate from Brunel University West London. He obtained his MSc (Computer Science) and MS in Information Technology from the University of Peshawar-Pakistan, and he is currently the Head of the Ph.D. and MS-Computer Sciences Department at IMSciences Peshawar-Pakistan.