IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Mathematical Modelling of Engineering Problems
Vol. 13, No. 4, April, 2026, pp. 741-750
Journal homepage: http://iieta.org/journals/mmep

Sequential Bitonic-Clustered Squirrel–Cat Hybrid Optimization for CPU-Only Batch Task Allocation in Fog Computing

Ahmed M. Al-Salih[1], Mehdi Ebady Manaa[1,2], Rasha Hussein Joudah[3], Ahmed J. Obaid[4*]

[1] Department of Information Networks, College of Information Technology, University of Babylon, Babylon 51001, Iraq
[2] Intelligent Medical System Department, College of Sciences, Al-Mustaqbal University, Babylon 51001, Iraq
[3] Department of Cybersecurity, College of Information Technology, University of Babylon, Babylon 51001, Iraq
[4] Faculty of Computer Science and Mathematics, University of Kufa, Najaf 540011, Iraq

Corresponding Author Email: ahmedj.aljanaby@uokufa.edu.iq

https://doi.org/10.18280/mmep.130411

Received: 18 December 2025
Revised: 28 March 2026
Accepted: 13 April 2026
Available online: 15 May 2026

Keywords:
fog computing, CPU-based task allocation, metaheuristic optimization, Squirrel Search Algorithm, Cat Swarm Optimization, Bitonic Clustering

ABSTRACT

Fog computing extends cloud computing by leveraging resources closer to end-users, enabling lower latency, enhanced security, and reduced operational cost for real-time applications. Efficient CPU-based batch task allocation in heterogeneous fog environments remains challenging due to device diversity. This study proposes a hybrid resource allocation strategy that integrates sequential Bitonic Clustering with metaheuristic optimization. Tasks are first grouped into priority-based clusters using a bitonic sorting approach, providing structured initial allocations. Subsequently, clusters are assigned to fog nodes through a hybrid metaheuristic combining the Squirrel Search Algorithm and Cat Swarm Optimization. Comparative evaluations against existing algorithms demonstrate that the proposed method improves allocation efficiency, reduces processing time, and enhances overall system performance under CPU-only constraints. These findings offer a practical approach for resource management in fog computing environments.

1. INTRODUCTION

Due to the rapid development of network devices, cloud computing has attracted many types of Internet users to provide services with high scalability and flexibility in various application areas [1, 2]. The number of Internet of Things (IoT) devices has increased rapidly due to the popularity of smartphones, wearable technology, and other sensors. As a result, cloud platforms have been proposed to connect these devices to the Internet so that the data they generate can be further processed and uploaded to a cloud server [3]. In general, a cloud server has unlimited processing power and storage. However, since it is geographically and/or conceptually distant from its users, uploading huge amounts of data to a cloud server consumes a lot of bandwidth and may cause problems like data loss or delay in rendering important application data.

Additionally, there is the potential for sensor data to grow exponentially from various sensors located in different areas, like video surveillance systems in a smart city or patient body sensors in health care scenarios. The advantages of combining IoT with cloud computing have been demonstrated and examined for numerous real-time uses such as smart healthcare, smart city, and other applications [4, 5]. The Cloud-IoT model allows various cloud servers with abundant resources to handle and calculate tasks produced by various apps [6, 7]. Despite the numerous advantages of cloud computing, IoT users face multiple difficulties such as unanticipated delays caused by extended transmission time from end device to cloud server, limited network bandwidth due to high number of requests, and diverse networks.

The traditional computing system, like the cloud environment, may not always be efficient in handling the large amount of data produced by various sensor devices spread across different locations [8]. Hence, when bandwidth is restricted and application quality of service (QoS) requirements are time-sensitive, like application deadlines, it is not feasible to offload all tasks to a centralized cloud server for different applications [9, 10]. In order to address the obstacles faced by IoT applications, it is necessary to expand cloud-like services to the network's edge to reduce the latency in data transmission when IoT devices send a request.

Hence, scholars have turned their attention to an emerging computing concept known as fog computing for resolving the issue of dependable and swift connectivity in upcoming scenarios [11]. Consequently, fog computing is incorporated into IoT-cloud systems. To meet QoS standards and enable high performance computing in the heterogeneous link system, computing, storage, communication, and control are brought closer to IoT devices.

The concept of fog computing expands on the cloud computing model, presenting a decentralized system of storage, networking, and computing resources to facilitate features like IoT device mobility and context awareness [12].

The majority of service requests from cloud servers can now be managed by the cloud computing platform, positioned

between actual IoT devices and cloud servers to enhance system performance in service latency, workload balancing, and resource utilization. Optimizing fog network resources is crucial for overall system performance, given the diverse nature of IoT devices and Fog network resources. The fog computing model offers various resources like computing, communication, and storage to handle IoT data and store it locally, rather than transmitting all data to the cloud [13]. The primary aim of fog computing is to enhance the effectiveness and performance of different IoT data gathered from sensor devices or applications by processing, analyzing, and storing it at the network's edge using diverse fog devices, instead of transferring it to a centralized device.

Several studies have proposed the best ways to distribute computing tasks across the network with task execution delay. Tasks are basically loaded on the best alternative nodes with the most resources (large storage capacity, high-speed processing, etc.) and reliable communication network conditions (latency, bandwidth across IoT devices, etc.) in many cases available. As a result, to achieve QoS measures, actions in the fog computing tier must be implemented with effective resource allocation and management. In order to efficiently allocate resources and perform tasks in specific computing fields, several new heuristic and meta-heuristic algorithms have been presented [14]. Among meta-heuristic algorithms, one of the most recent algorithms presented in this field is the flying squirrel optimization, which was proposed by Jain et al. [15]. Flying squirrels do not actually fly; instead, they glide using a cost-effective and energy-efficient method of locomotion. Squirrels are able to effectively use food resources by showing their changing search behavior. The searching starts when the squirrels begin to seek out food. In the autumn heat, squirrels search for food by flying between trees. In the process, they move around and investigate various parts of the forest. Because the warm weather allows for it, they are able to satisfy their daily energy requirements by eating the plentiful acorns that are readily available, leading them to consume them as soon as they come across them. Once squirrels have met their daily energy needs, they start looking for the best food source for the winter - specifically, coarse grain nutrients. Storing big seeds allows them to save the energy required in very harsh environments, decreasing expensive searching trips and ultimately boosting chances of survival.

In winter, the decrease in deciduous forest cover results in a higher predation risk, causing flying squirrels to decrease their activity without hibernating. At the conclusion of winter, flying squirrels resume activity. This process is ongoing throughout the lifespan of a flying squirrel and is fundamental to Squirrel Search Algorithm (SSA). To make the mathematical model simpler, we take into account the following assumptions:

(a) There are n number of flying squirrels in a forest and it is assumed that there is one flying squirrel in a tree.
(b) Each flying squirrel searches for food individually and optimizes the use of available food resources by displaying a dynamic foraging behavior.
(c) There are only three types of trees in the forest. Common tree, oak tree (food source of common oak) and hickory tree (food source of large oak).
(d) The desired forest area is assumed to include three common oak trees and one large oak tree.

2. RELATED WORKS

Ghaffari [16] proposed an ant colony-based technique to assign tasks to virtual machines that require the least time and cost. There are three steps in the proposed algorithm. First, tasks were classified based on cost and completion time upon arrival. The identified tasks were ranked based on cost and time constraints in the second step. Ant colony method was used in the third stage to give work to virtual machines.

The problem of assigning tasks to virtual machines was initially expressed as a linear scheduling model, and a load balancing strategy was proposed for assigning tasks to virtual machines using the Hungarian Algorithm Based Binary Policy (HABBP) algorithm in this study [17]. The next step was to use a genetic algorithm Genetic Algorithm Based Virtual Machine Placement (GABVMP) to deal with the virtual machine placement problem. According to the simulation results, this algorithm performs better than the first fitting and random placement algorithms in terms of allocation cost parameter. The genetic algorithm may converge to a local optimum, but its convergence speed is slow.

Li et al. [18] integrated particle swarm optimization and fuzzy clustering techniques to classify resources and limit the resource search range. Source qualities and characteristics were first standardized and normalized. Finally, they presented a technique for resource scheduling based on fuzzy clustering. According to the simulation results, the proposed algorithm performs better than the traditional fuzzy algorithm in terms of convergence speed.

A heuristic technique for scheduling fog computing tasks based on particle swarm and ant colony optimization was presented in study [19]. This technique enables the scheduling of end devices with low processing power, making it suitable for real-time operation and efficient processing. The proposed algorithm performs well in terms of completion time, and reliability compared to Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and round robin.

A Multi-Objective Simplified Swarm Optimization (MOSSO) technique is presented in the study [20] to address the scheduling problem of fog computing. Simple Swarm Optimization (SSO) served as the basis for MOSSO, a multi-objective optimization technique. SSO was a population-based stochastic optimization technique that was easy to use and effective. Compared with the Multi-Objective Particle Swarm Optimization (MOPSO) technique, the MOSSO method produced better results with reduced processing rate and cost.

Saad et al. [21] proposed multi-objective task scheduling in fog computing with a hybrid of Genetic Algorithm and Particle Swarm Optimization (GA-PSO) for big data in fog computing environment. They achieve effective exploration and exploitation of the search space, which leading to improved performance execution time, response time, and completion time. The authors concluded an improved search approach superior in results for GA exploration skills and PSO's exploitation characteristics.

The proposed work to allocate computing tasks for fog nodes using Gorilla Troops Optimization with Skip Salp Swarm Algorithm (GTO-SSSA) in this study [22]. The primary goal of GTO-SSSA is to efficiently address the complex challenges of the task scheduling problem in a fog computing environment. This involves optimizing task allocation across fog nodes while simultaneously considering multiple objectives, including reducing task completion times, minimizing execution time, and improving productivity. The

GTO-SSSA model demonstrates improved performance in fog computing, consistently outperforming comparable models across various task sizes and volumes, while achieving significant reductions in task completion times. Comparison results indicate that the performance improvements achieved by GTO-SSSA compared to other models reflect substantial gains in task scheduling efficiency, ranging from 0.87% to 17.83%.

Liu et al. [23] used machine learning techniques to investigate resource allocation in IoT networks using edge computing. In order to organize IoT users into different clusters based on user preferences, user-centered clustering is investigated in this study. While the lowest-priority cluster completes the computing tasks locally, the highest-priority cluster is responsible for offloading the computing tasks and working on the edge server.

Deng et al. [24] first coded the state of the service delivery system and the resource allocation plan, then set the resources allocated for the service with their Markov Decision Process model and the benefit of reliability with the good degree that Service Level Agreement (SLA) can demonstrate. Using a reinforcement learning approach, a learned resource allocation policy is derived from this. A trained policy can always allocate resources in the safest possible ways by dynamically building plans that fit the current system conditions. Through a series of experiments on the YouTube request dataset, it has been shown that the edge service delivery system using this strategy performs at least 21.72% better than the baselines.

A technique for wireless network resource allocation for 6G applications was presented by Lakhan et al. [25]. This network is a hybrid wireless sensor network and resource allocation is done using a mechanism called MAS. By introducing clustering technology in data mining, the entire 6G system is divided into separate clusters to optimize the use of scarce resources and provide better control over different types of sensor nodes in the hybrid network. An established machine learning method called Deep Neural Network (DNN) provides several layers of partitioning algorithms in local and remote areas. The main advantage of layers is their ability to provide a wide range of options when multiple parameters change during runtime. Therefore, DNN is more effective in situations where there is system uncertainty during program partitioning during runtime. However, DNNs offer superior strategies for many current approaches to decision making.

Diamanti et al. [26] used different degrees of available computing power across the network to create an incentive mechanism that tries to change the preferences of self-service users from the lower fog computing layer to the upper layer and at the same time consider their delay. The edge server calculates users' packets based on multidimensional contract theory modeling, which is used to handle the heterogeneity of users with respect to the unique multidimensional characteristics of their applications (such as delay tolerance/sensitivity). This means that the amount of work that each user initially delivered to the edge is represented by their effort, allowing additional work to be submitted and processed in the fog. The goal of this game is to increase the efficiency of resource use. Through modeling and simulation, the incentive system and the framework for resource allocation are evaluated based on their performance in different conditions.

Several recent studies show that meta-heuristic optimisation coupled with learning-based controllers can jointly improve QoS and security in fog-edge systems. Baburao et al. [27] proposed a study that investigates the use of PSO for improving resource allocation in 5G networks focuses on resource utilization, and overall network efficiency in highly interconnected environments. The proposed work [28] combines PSO with a deep deterministic policy gradient agent to reach near-optimal, real-time resource allocation in 5G cores. These converging findings underpin the design choices detailed in the next section.

3. PROBLEM STATEMENT AND SYSTEM MODEL

The system considers a static task allocation problem in a heterogenous environment. The problem considers a set of independent tasks $T = \{T_1, T_2, T_3, \ldots, T_n\}$, and a set of computing nodes $N = \{N_1, N_2, N_3, \ldots, N_n\}$, and task ($T_i$), is associated with workload w_i, while each node N_j has a CPU processing capacity c_j. Accordingly, the execution time of task T_i on node N_j is defined in equation $p_{ij} = w_i/c_j$, where, p_{ij} denotes the execution time required to process task T_i on node N_j.

The optimization problem is to address task-to-node allocation problem $x_i = j$, which assigns each task to exactly one node. The model assumes batch/offline task availability, sequential execution of tasks in each node, and parallel operation across nodes. Moreover, CPU is the only resources dimensions in the current implementation. The memory, storage, bandwidth, migration, and communication overhead are out of the scope of the present model. The proposed approach minimizes a combined objective involving makespan and load balancing by incorporating the standard deviation of node loads $L_j = \sum_{i:x_i=j} \frac{w_i}{c_j}$, where w_i is the computational workload of task T_i, and c_j is the processing capacity of N_j.

4. THE PROPOSED METHOD

The proposed algorithm in this study is to use Squirrel meta-heuristic search algorithm considering several goals in the process of mapping virtual resources to physical hosts and combining it with clustering. The correct use of the combination of several methods brings better results, covers the disadvantages of each method with the help of another method, and thus benefits from the advantages of each method. The meta-exploratory method presented in this study consists of a combination of two techniques. It actually combines a biological method with an efficient technique. Multi-objective SSA and bitonic algorithm along with Cat Swarm Optimization (CSO) are the strategies used in the proposed method. In the proposed approach to build a multi-objective scheduling algorithm, the Squirrel Search method is combined with a bitonic sorter that enhances load balancing. The steps of the algorithm in the proposed approach are as follows:

(a) First, user requests are entered into the system, and then these requests are categorized and used for scheduling. Bitonic algorithm is used to categorize and cluster requests.
(b) The values of the parameters of the current scheduling solution, including the request load, are given as input to the bitonic sorting algorithm.
(c) Finally, after classifying the requests, the classified requests are given as input to the multi-objective SSA to calculate the suitability of the computing resource using the objective function. The input values are used

to calculate the objective function and obtain values related to the main scheduling objectives. The new scheduling information is maintained and the search space for scheduling solutions is updated after the target values are verified. The results of the new scheduling solution are then compared with the previous solutions to see if it is superior.

(d) The previously mentioned steps are repeated a predetermined number of times until requests are best allocated to computer resources.

Algorithm 1. Proposed Bitonic-Guided SSA-CSO (BG-SSA-CSO) for Task-to-Node Allocation

Input:
$T = \{t_1, t_2, ..., t^M\}$ // task set
$N = \{n_1, n_2, ..., n^K\}$ // fog-node set
w_i // workload of task t_i
c_j // CPU capacity of node n_j
Output:
$X^* = [x_1, x_2, ..., x^M]$ // final allocation, $x_i \in \{1, 2, ..., K\}$
Steps:
1: Generate bitonic seed X_bit by ordering T according to w_i and N according to c_j
2: For each task t_i,
Assign it to node n_j that minimizes:
$score_{ij} = F_j + (w_i / c_j) + \lambda |r_i - r_j|$
// F_j: predicted finish time of node n_j
// r_i, r_j: resource demand of task t_i and node n_j
3: Initialize SSA population Ω_SSA using X_bit, mutated seeds, and random feasible solutions
4: Evaluate each $X \in$ Ω_SSA using the fitness function f(X)
5: For iter = 1 to I_SSA do
a. Update SSA solutions toward the best and elite solutions.
b. Apply seasonal diversification if needed.
c. Repair each solution to valid node indices.
d. Re-evaluate f(X).
6: End For
7: Let X_SSA be the best SSA solution
8: Let E_SSA be the elite SSA solutions
9: Initialize CSO population Ω_CSO using X_bit, X_SSA, and E_SSA
10: Evaluate each $X \in$ Ω_CSO using the same fitness function f(X)
11: For iter = 1 to I_CSO do
a. Update cats using seeking mode or tracing mode.
b. Repair each solution to valid node indices.
c. Re-evaluate f(X).
12: End For
13: Let X_CSO be the best CSO-refined solution
14: If f(X_CSO) < f(X_SSA) then
$X^* = X\_CSO$
15: Else
$X^* = X\_SSA$
16: End if
17: Return X^*

The initial population is formed which includes requests and hosts is shown in Figure 1. Then the requests are sorted and categorized based on bitonic sort. Categorized requests are given as input to the cat swarm algorithm. Algorithm inputs include categorized requests (flying squirrels) and hosts (food). Then, in the next step, we will examine the objective function of the SSA algorithm based on Eqs. (1) and (2). If the hosting has the desired conditions, the allocation of categorized requests is done, otherwise the steps are repeated. At the end, if all the requests are finished, the algorithm is terminated and the desired outputs including energy and delay are displayed. Algorithm 1 shows the main pseudo-code for task-to-node allocation task scheduling.

Figure 1. Flowchart of the proposed method

4.1 Bitonic method for classification

Bitonic sorting is actually a sorting method in which half of the numbers are sorted in ascending order and the other half of the numbers are sorted in descending order. With the help of this structure, considering that the load balance is important and vital in the proposed method, by using this method, we intend to classify the requests and create balanced groups and then assign the created groups to the appropriate resources using the SSA algorithm. In order to categorize the requests using the structure of the bitonic method, according to the load of each request and its length, an ascending-descending set is created based on the length of the requests or descending-ascending. After sorting the requests by length, we need to classify the requests. In order to categorize the requests, since according to the bitonic algorithm, half of the numbers are ascending (or descending) and the other half are descending (or ascending), we extract categories of requests using the list obtained in the first step that the total capacity of each batch is almost balanced. As a result, to create clusters and create balanced clusters, first the first number from the first half is placed in the same category with the first number from the

second half, and then the second number from the first half is placed in the same category with the second number from the second half and so it will continue with the creation of categories. For example, if the length of requests includes numbers (100, 200, 300, 400, 200, 300), using the bitonic sorting method and using the ascending, descending mode, these numbers are sorted as follows: 100, 200, 300, 400, 300, 200.

In the above example, the bitonic method is of the ascending-descending type, where the first half of the numbers are ascending and the second half are descending numbers. Based on this example, the first three numbers are ascending and the next three are descending. As a result, the classification is done in this way that first the first number from the ascending half is placed in the first category and the first number from the descending half is placed in the first category. After that, to create the second category, the second number from the first half is placed in the same category with the second number from the second half, and the next categories are created in the same way. For example, the categories created are as follows: Group 1 = {100, 400}, Group 2 = {200, 300}, Group 3 = {300, 200}.

The categories created based on the proposed method, as it is known, have an almost balanced load. In fact, with this solution, the categories created are almost balanced, because at the beginning, half of the data is ascending and the other half is descending, when the first number of each half in one category and the second number of each half in the next category and so it is placed until the next categories, as a result, we create balanced categories with this structure. After creating balanced categories, it is necessary to map the created categories to the appropriate resource. For this purpose, we will use the multi-objective SSA algorithm.

4.2 Squirrel Search Algorithm

The structure of the proposed method is to use the multi-objective SSA algorithm to map requests. In this regard, in this research, a new strategy for the efficient allocation of program resources in the combined cloud and fog paradigm has been introduced and formulated. In order to express the proposed method, the multi-objective SSA algorithm is used to map the virtual resources to the hosts after categorizing the requests using the provided bitonic method. In fact, a solution using the SSA multi-objective algorithm has been proposed for proper scheduling for the categorized requests from the previous step to be effective in reducing the energy consumption by creating a balanced load of resources and preventing the overflow of resources and their underload by reducing the number of active servers, while avoiding service quality violations and delays. For this purpose, there is a need for an objective function that makes a more appropriate allocation to be effective in improving the efficiency of resources by balancing the load. The steps of the proposed algorithm to find the optimal location are as follows:

(a) **First step: Input data:** The input data in the SSA algorithm are squirrels and food. In the proposed solution, in order to model the SSA algorithm, squirrels form user requests, and the appropriate host location is suitable for hosting the squirrels, which are considered as the input of the problem. The purpose of the proposed method is to find the best location based on the objective function.

(b) **Second step: Calculation of the objective function:** The objective function of the proposed method is to find the ideal place for squirrels to search for food. Therefore, the value of the objective function is calculated and, taking into account the objective function, the movements of the response set (categorized requests) are determined. The objective function of the proposed method considers many objectives to find computing resources with more free resources and less energy. When grouping requests are to be assigned to a computing resource, the constraint must be checked that the resources have the capacity to host the requests of each group. Consequently, the following restriction is considered for this purpose. The processing capacity of each node must be positive $C_j > 0$, and the workload of each task must be positive $W_i > 0$, where each task must be assigned to exactly on computing node. Based on this limitation, the hosts that have the necessary positive for both of processing and workload and each task request must be assigned to exactly computing node. After considering the constraints, we examine the objective function in the proposed method. For this purpose, the objective function is shown as Eq. (2).

$$f(x) = C_{max} \times \sigma(L) \tag{1}$$

where, L defined in problem statement Section 3, and the system makespan.

$$C_{max} = \max_j L_j \tag{2}$$

In the above formula, two basic constraints are considered for the objective function. In the first criterion, the processing capacity of each computing node and the workload of each task must be positive the goal of the optimization process is to minimize the system makespan while maintaining balanced workloads among computing nodes. Therefore, the fitness function is defined as shown in Eq. (1). The system makespan is defined as the maximum load among all computing nodes, as expressed in Eq. (2).

The proposed method's objective function is to find a valid computing node for task allocation according to its objective function while minimizing the overall system's makespan and keeping workloads on each node as balanced as possible. Each candidate solution will represent a specific mapping of the tasks that are to be executed onto the available computing nodes. The optimization process evaluates all nodes and determines an output based on the lowest value of the overall system makespan.

Therefore, this algorithm looks for a task-to-node assignment that minimizes the highest finish time while preventing an overloaded node. The algorithm attempts to minimize the makespan and provide a balanced load across the nodes by using the standard deviation of the node load as a measure of load balancing. The proposed approach enhances the task allocation efficiency and improves the balance among the computing resources in the system.

(a) **Third step: Correcting the position:** according to the objective function presented in the previous step, in the SSA algorithm, at the time of moving towards the food, after the new position and location for the squirrel has been examined and evaluated based on the objective function, if the new position is better than the previous solution, the squirrels move to the new position (computational resource) and update their position.

Otherwise, it remains in its current position.

(b) **Fourth step: Checking the end condition:** In this step, if the squirrels have finished, the program ends, otherwise, steps 2 to 4 are repeated.

4.3 Cat Swarm Optimization algorithm

In the CSO technique, their two main behaviors are modeled with two sub-models called tracking and searching mode. With a method of combining these two modes in a defined ratio, the CSO algorithm shows good performance. In this algorithm, like PSO, the location of cats is an answer, and this algorithm solves optimization problems by using cats and modeling their behavior. In the optimization of the swarm of cats, it is first decided how many cats to use. Each cat has a position that has M dimension. In addition to this position, each cat has a speed for each dimension and a fitness value that indicates the fitness level of that cat. This fitness is obtained by the fitness function. Every cat also has a flag, which is used to indicate whether the cat is in search or tracking mode. In this study, the optimization method of cat swarming is modified so that resources may be allocated to jobs in the cloud infrastructure efficiently, hence reducing system makespan for users' requests to be executed.

A group of cats will be employed in the suggested method; some will be in search mode while others are in tracking mode. Based on the cat's current condition, each cat represents a resource task mapping. The mapping cost is reduced when the cats are given the fit value. A fresh group of cats is chosen to be in the search state at the start of each iteration. In the end, the optimal place for cats will indeed be the mapping that produces the lowest cost when compared to other mappings. In light of this, the subsequent analysis begins with a sub-model that looks at search and tracking modes in order to simulate the cat's search behavior for the target, which in this case is

4.3.1 Search mode

The four main factors in each cat's search mode are defined as follows:

(a) Search Memory Pool (SMP): This factor is defined to define the search memory size for each cat, which shows the positions searched by the cat. The cat chooses one of the positions in its memory based on its fitting functions.

(b) Change Dimension Count (CDC): This factor determines the number of dimensions that will change.

(c) Search Range of Dimension (SRD): This factor shows the rate of change for the selected dimension. In search mode, if a dimension is selected to change, the difference between its new and old value will not be outside the range defined by the SRD.

(d) Selected Position Candidate (SPC): This factor is a Boolean variable that determines whether the point where the cat is currently standing is one of the movement candidates or not. This value will not affect the SMP value.

The general algorithm when the cat is in search mode is as follows:

The first step is to create j copies of the current position of the kth cat (Cat_k) where $j = SMP$. If the value of SPC is true, then $j = (SMP - 1)$ and then it keeps the current position as one of the candidates. In the second step, for each copy, according to the CDC, the SRD is increased by a percentage of the current values and added to the previous values. In the third step, the fitness function is calculated for all candidate points. In the fourth step, if all fitness functions are not exactly equal, the probability of selecting each candidate position is calculated according to the normalized fit of that position, and otherwise, the probability of selecting all candidate positions is set equal. The following relationship shows how to calculate the normalized fit of a candidate point according to the fitness level of that point.

$$P_i = \frac{|FS_i - FS_b|}{FS_{max} - FS_{min}}, \quad 0 < i < j \tag{3}$$

where,

P_i: Probability related to current candidate cat_i

FS_i: cat_i fit value

FS_{max}: The maximum value of the fitting function

FS_{min}: Minimum value of the fitting function

Considering that in the proposed solution, the main goal of the merit function is to find the minimum solution, and in other words, its main goal is to reduce the system makespan, so in this case $FS_b = FS_{max}$. Finally, in the last step, a position to move is randomly selected from the candidate positions and the cat_k position is changed.

4.3.2 Tracking mode

The following solution is used to model the cat's behavior while tracking the target. When the cat goes into tracking mode, it moves according to its speed in each dimension and searches the local space by moving to the best position. The tracking process can be described in the following three steps:

The first step: updating the speed for each dimension ($v_{k,d}$) according to the following relationship:

$$v_{k,d} = v_{k,d} + r_1 \times c_1 \times \left(x_{best,d} - x_{k,d}\right), \quad d = 1, 2, \ldots, M \tag{4}$$

where, $x_{best,d}$ shows a position of the cat that has the highest amount of fitness and $x_{k,d}$ is the position of the kth cat. c_1 is a fixed number and r_1 is a random number in the interval [0, 1].

The second step: It is checked that the speeds are within the defined range. If there was a higher speed, it will be replaced with the maximum possible value in the desired range.

The third step: updating the position of the cat according to the following relationship:

$$x_{k,d} = x_{k,d} + k_{k,d} \tag{5}$$

In other words, at this stage, the set of answers obtained from the best location of the cat in the current iteration is updated.

The main goal of scheduling tasks is the optimal use of available resources so that the basis for increasing service quality is provided. In this research, service quality means reducing the time of executing tasks, which in this way provides the basis for reducing costs and also minimizing the time of executing users' requests. In order to reduce the costs of providing services to users, cloud service providers have different policies and in fact, based on the type of user and desired services, they adopt different policies. Algorithm 2 shows the main step of the hybrid SSA-CSO framework.

Algorithm 2. Hybrid SSA–CSO Optimization Procedure

Input:

- Task set $T = \{t_1, t_2, \ldots, t^M\}$
- Node set $N = \{n_1, n_2, \ldots, n^K\}$
- Population size P

Output:

- Optimal task-to-node assignment $X^* = [x_1, x_2, \ldots, x^M]$, $x_i \in \{1, \ldots, K\}$

Steps:

1. **Bitonic-guided grouping and population initialization:** Sort and group tasks based on workload characteristics and initialize the candidate solutions.
2. **Population evaluation:** Generate an initial population of candidate solutions mapping tasks to nodes and calculate the objective function for each solution.
3. **Solution refinement using SSA:** Apply the SSA to refine the candidate solutions for improved task-to-node allocation.
4. **Final optimization using CSO:** Use the CSO algorithm with the SSA-refined solutions as the initial population to further optimize the task assignment and obtain the final optimal solution X*.

5. EVALUATION OF RESULTS

In this section, we evaluate the efficiency of the proposed method. Therefore, we compare the performance of the proposed method with one of the latest algorithms presented in this research field, namely Enhanced Dynamic Resource Allocation Method (EDRAM) [27]. The proposed method and the compare method are implemented using Python programming language. The results are presented in the form of makespan, response time and standard of deviation (STD) of load. Also, two scenarios are considered for evaluation. The first scenario, which is for the change in the number of tasks, has been evaluated for 10 nodes and the number of tasks in the range [50–150]. The purpose of this scenario is to check the scalability of the compared methods. In the second scenario, which is for the change in the number of nodes, evaluations have been performed for the number of nodes between [5–15] and 100 tasks.

The implemented Python framework is a batch/offline simulator rather a general event-driven or time-stepped simulator. For any candidate solution, the simulator first used task-to-node assignment, insert each task into local queue on its assigned fog node.

Task execution on each node is sequential and is computed using a local node clock. Synthetic workloads and node capacities are generated using seeded pseudo-random number generators. In the default experimental setting, 100 tasks and 10 nodes are created per run; task workloads are sampled uniformly from 10^5 to 10^7, and node CPU capacities are sampled uniformly from 10^4 to 10^6. This seeded generation process ensures that all experiments are reproducible.

Table 1 summarizes the reproducibility settings of the implemented batch/offline CPU-only simulation framework the settings used in the simulation.

In this section, the results of the proposed method, which is a combination of CSO algorithm and Squirrel Search, are described as CSO-SS. In the following, the results of the scenarios are presented in the form of the mentioned evaluation criteria.

Table 1. Simulation settings

Parameter	Value
Number of nodes	5–15
Number of tasks	50–150
Tasks size	10^5–10^7 Instructions
Processor speed	10^4–10^6 MIPS
Population size	100
Iteration	50
CSO-SMP	5
CSO-SPC	False
CSO-CDC	0.8
CSO-SRD	0.15
Simulator	Batch/offline assignment-and-evaluation simulator
Queue model	One local queue per node
Queue policy	SPT (default), FIFO supported
Scheduling trigger	Once per batch after solution generation

Note: Cat Swarm Optimization = CSO; Search Memory Pool = SMP; Selected Position Candidate = SPC; Change Dimension Count = CDC; Search Range of Dimension = SRD; Shortest Processing Time = SPT; First In First Out = FIFO.

5.1 Makespan

Makespan is the amount of time it takes for all tasks to be completed by fog nodes. Figures 2 and 3 of makespan show the compared methods in two scenarios.

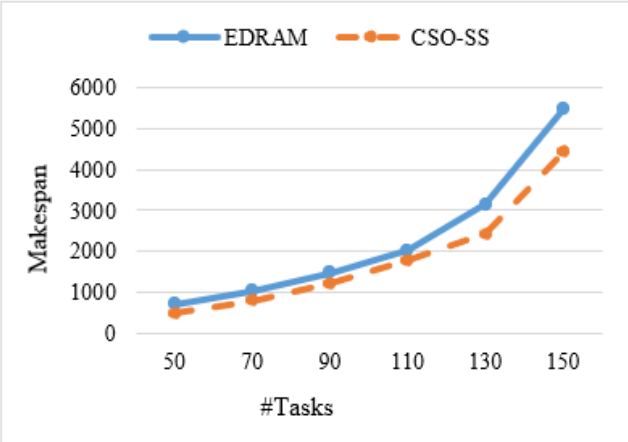

Figure 2. Makespan vs. number of tasks

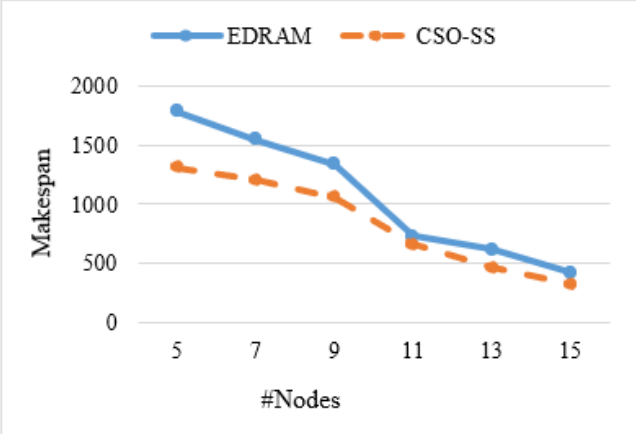

Figure 3. Makespan vs. number of nodes

The results presented in these figures show that the proposed method has been able to obtain a lower makespan than the EDRAM method in both scenarios. Makespan of the proposed method in the first scenario between 11% and 32% and in the second scenario between 9% and 27% is less than the EDRAM method.

5.2 Waiting time

Waiting time is the time between receiving the task and starting. Figures 4 and 5 contain the waiting time of the compared methods in both scenarios, which prove that the SS-CSOS method responded to the tasks faster than the EDRAM method. So that the response time of the proposed method in the first and second scenarios is 19% to 34% and 7% to 32% better than the compared method, respectively.

5.3 Standard of deviation of load

This criterion indicates the balance in the load distribution in fog. The more evenly the tasks are distributed among the nodes, the more efficient the fog is. Figures 6 and 7 show STD of load in two scenarios.

The STD of load results show the superiority of the CSO-SS method was repeated over 20 independent runs using different random seeds. So, the CSO-SS method has a lower STD in the distributed load among the nodes. The proposed method has between 28% and 35% less STD in the first scenario and between 4% and 19% in the second scenario in the distributed load.

Table 2 shows the main results for the hybrid CSO-SS proposed method compared with three widely baselines methods, namely Round Robin, Min-Max, and PSO to provide a fair and comprehensive evaluation.

The comparisons results show that the proposed CSO-SS method outperforms round robin in all tested node setting, and provide better performance that PSO in term of makespan, load distribution, and average task start time in several scenario. The min-min archives the lowest makesspan under the parameters setting as shown in Figure 8.

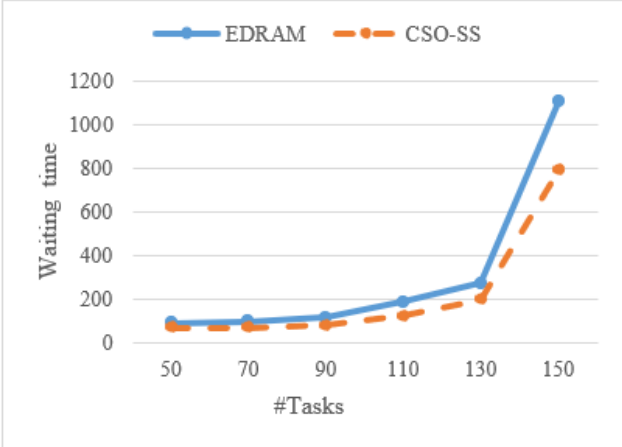

Figure 4. Waiting time vs. number of tasks

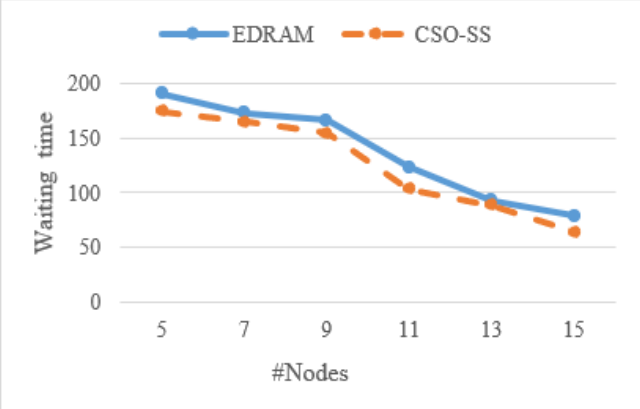

Figure 5. Waiting time vs. number of nodes

Table 2. Performance comparison of round robin, min-max, Particle Swarm Optimization (PSO), and proposed Cat Swarm Optimization-Squirrel Search (CSO-SS)

Number of Nodes	Method	Makespan_mean	StdLoad_mean	AvgStart_mean
5	Round Robin	7285.0	2862.634285	1353.4790
5	Min-Min	420.62	89.314601	99.2866
5	PSO	1807.2	619.688039	586.1176
5	CSO-SS	1773.5	588.658243	194.7770
7	Round Robin	4601.0	1538.682179	636.7922
7	Min-Min	157.62	61.011612	24.5225
7	PSO	1110.6	390.697145	211.7515
7	CSO-SS	1200.4	417.854366	64.8513
9	Round Robin	3462.0	1118.325065	570.3356
9	Min-Min	153.72	61.692446	25.3955
9	PSO	1064.0	326.626173	223.7351
9	CSO-SS	1028.9	325.176003	77.7550
11	Round Robin	3484.0	1172.878929	628.8381
11	Min-Min	171.74	74.573569	29.2563
11	PSO	1163.5	374.471727	244.8827
11	CSO-SS	1061.0	342.286987	80.6821
13	Round Robin	2931.0	979.518952	495.6834
13	Min-Min	98.94	38.139499	15.7987
13	PSO	954.1	301.626042	190.3893
13	CSO-SS	909.0	277.440116	65.6973
15	Round Robin	2218.0	692.525535	314.1555
15	Min-Min	80.22	33.599130	11.0895
15	PSO	761.0	229.950965	118.1706
15	CSO-SS	771.9	228.876190	48.5342

Table 3. Performance comparison of all algorithms over 20 independent runs

Algorithm	Makespan (Mean ± Standard Deviation)	Load Imbalance (Mean ± Standard Deviation)	Average Waiting Time (Mean ± Standard Deviation)	Average Response Time (Mean ± Standard Deviation)
Random	1273.97 ± 963.22	378.04 ± 285.00	93.99 ± 47.38	120.17 ± 57.97
Min-Min	114.23 ± 18.77	25.37 ± 12.35	33.48 ± 5.64	42.94 ± 7.46
Cat Swarm Optimization (CSO)	131.55 ± 19.47	25.19 ± 8.48	33.38 ± 5.65	43.44 ± 7.27
Bitonic-Guided-Squirrel Search Algorithm-CSO (BG-SSA-CSO)	118.84 ± 18.37	13.45 ± 7.79	32.17 ± 5.62	42.51 ± 7.24

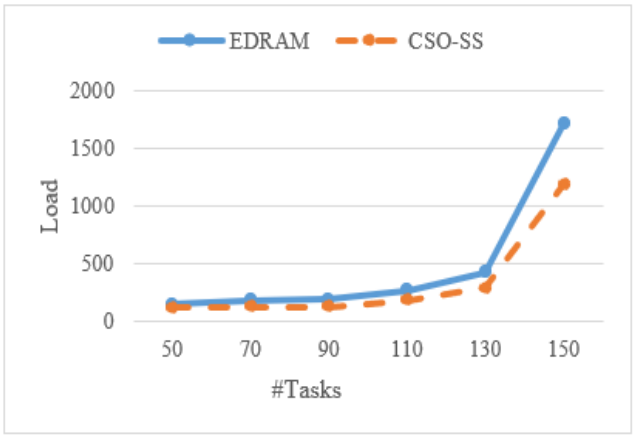

Figure 6. Standard of deviation (STD) of load vs number of tasks

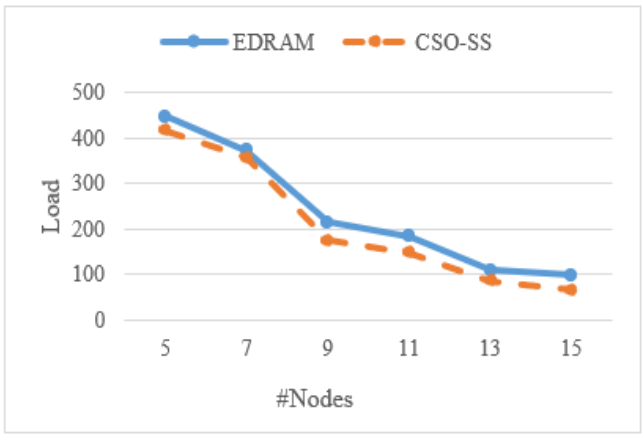

Figure 7. Standard of deviation (STD) of load vs. number of nodes

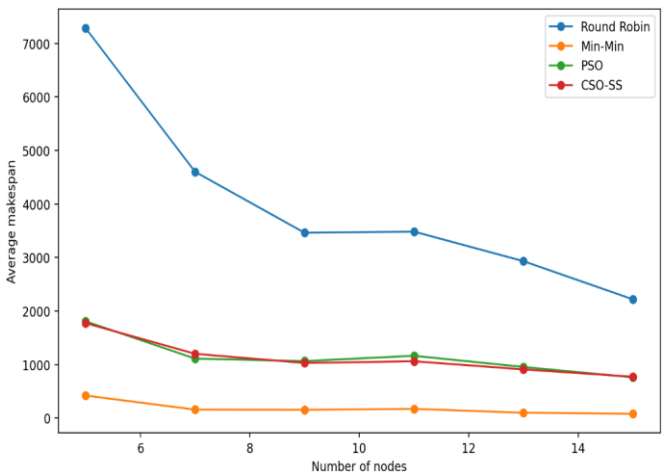

Figure 8. Benchmark comparison of scheduling methods

All experiments were repeated over 20 independent runs using different random seeds. The obtained results are reported as mean ± standard deviation for makespan, load imbalance, average waiting time, and average response time. Table 3 shows the performance of all algorithms over 20 independent runs.

The proposed hybrid CSO-SSA method time complexity is approximately $(O(pop \times iter \times |T| \times |N|))$, where, pop is the population size, $iter$ the number of iterations, $|T|$ denotes the number of tasks, and $|N|$ presents the number of computing nodes involved in evaluating each candidate scheduling solution.

6. CONCLUSION

In this study, we have used the combination of a bitonic-based grouping stage to sort and classify tasks before optimization and an evolutionary algorithm squirrel search and CSO to find an optimal resource allocation in in a fog-inspired heterogeneous environment. The proposed system uses batch CPU-only/ task-to-node allocation by squirrel search and CSO to cluster and search the resource allocation problem space and find the optimal solution. To evaluate the performance, the proposed method is implemented along with the EDRAM method presented in study [27] using Python programming language and other baseline methods Round Robin, Min-Min, and PSO. Evaluation criteria including makespan, response time and STD of load were considered. The results of the evaluations in the form of two scenarios that examine the effect of the number of tasks and nodes, show that the proposed method has performed much better than the EDRAM method.

The results of the proposed method of this study show that the combined use of clustering and evolutionary algorithms can provide promising results in the field of resource allocation in fog computing. Therefore, testing the new evolutionary algorithm and clustering methods based on neural networks will be among our future works.

REFERENCES

[1] Iorga, M., Goren, N., Feldman, L., Barton, R., Martin, M., Mahmoudi, C. (2018). Fog computing conceptual model (No. NIST Special Publication (SP) 500-325). National Institute of Standards and Technology.

[2] Garg, S.K., Versteeg, S., Buyya, R. (2013). A framework for ranking of cloud computing services. Future Generation Computer Systems, 29(4): 1012-1023. https://doi.org/10.1016/j.future.2012.06.006

[3] Sun, H., Yu, H., Fan, G., Chen, L. (2020). Energy and time efficient task offloading and resource allocation on the generic IoT-fog-cloud architecture. Peer-to-Peer

Networking and Applications, 13(2): 548-563. https://doi.org/10.1007/s12083-019-00783-7

[4] Manogaran, G., Varatharajan, R., Lopez, D., Kumar, P.M., Sundarasekar, R., Thota, C. (2018). A new architecture of internet of things and big data ecosystem for secured smart healthcare monitoring and alerting system. Future Generation Computer Systems, 82: 375-387. https://doi.org/10.1016/j.future.2017.10.045

[5] Fortino, G., Parisi, D., Pirrone, V., Di Fatta, G. (2014). BodyCloud: A SaaS approach for community body sensor networks. Future Generation Computer Systems, 35: 62-79. https://doi.org/10.1016/j.future.2013.12.015

[6] Rimal, B.P., Choi, E., Lumb, I. (2009). A taxonomy and survey of cloud computing systems. In 2009 Fifth International Joint Conference on INC, IMS and IDC, Seoul, Korea (South), pp. 44-51. https://doi.org/10.1109/NCM.2009.218

[7] Botta, A., De Donato, W., Persico, V., Pescapé, A. (2016). Integration of cloud computing and internet of things: A survey. Future Generation Computer Systems, 56: 684-700. https://doi.org/10.1016/j.future.2015.09.021

[8] Klas, G.I. (2015). Fog computing and mobile edge cloud gain momentum: OpenFog Consortium, ETSI MEC and cloudlets. Technical Report, OpenFog Consortium.

[9] Arkian, H.R., Diyanat, A., Pourkhalili, A. (2017). MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. Journal of Network and Computer Applications, 82: 152-165. https://doi.org/10.1016/j.jnca.2017.01.012

[10] Karatas, F., Korpeoglu, I. (2019). Fog-based data distribution service (F-DAD) for internet of things (IoT) applications. Future Generation Computer Systems, 93: 156-169. https://doi.org/10.1016/j.future.2018.10.039

[11] Apat, H.K., Nayak, R., Sahoo, B. (2023). A comprehensive review on internet of things application placement in fog computing environment. Internet of Things, 23: 100866. https://doi.org/10.1016/j.iot.2023.100866

[12] Sadeeq, M.M., Abdulkareem, N.M., Zeebaree, S.R., Ahmed, D.M., Sami, A.S., Zebari, R.R. (2021). IoT and cloud computing issues, challenges and opportunities: A review. Qubahan Academic Journal, 1(2): 1-7. https://doi.org/10.48161/qaj.v1n2a36

[13] Bonomi, F., Milito, R., Zhu, J., Addepalli, S. (2012). Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, pp. 13-16. https://doi.org/10.1145/2342509.2342513

[14] Ahmed, R., Rangaiah, G.P., Mahadzir, S., Mirjalili, S., Hassan, M.H., Kamel, S. (2023). Memory, evolutionary operator, and local search based improved Grey Wolf Optimizer with linear population size reduction technique. Knowledge-Based Systems, 264: 110297. https://doi.org/10.1016/j.knosys.2023.110297

[15] Jain, M., Singh, V., Rani, A. (2019). A novel nature-inspired algorithm for optimization: Squirrel search algorithm. Swarm and Evolutionary Computation, 44: 148-175. https://doi.org/10.1016/j.swevo.2018.02.013

[16] Ghaffari, E. (2019). Providing a new scheduling method in fog network using the ant colony algorithm. In Collection of Articles on Computer Science. http://www.colloquiam.com/public/Ghaffari_2019a.

[17] Akintoye, S.B., Bagula, A. (2019). Improving quality-of-service in cloud/fog computing through efficient resource allocation. Sensors, 19(6): 1267. https://doi.org/10.3390/s19061267

[18] Li, G., Liu, Y., Wu, J., Lin, D., Zhao, S. (2019). Methods of resource scheduling based on optimized fuzzy clustering in fog computing. Sensors, 19(9): 2122. https://doi.org/10.3390/s19092122

[19] Wang, J., Li, D. (2019). Task scheduling based on a hybrid heuristic algorithm for smart production line with fog computing. Sensors, 19(5): 1023. https://doi.org/10.3390/s19051023

[20] Yeh, W.C., Lai, C.M., Tseng, K.C. (2019). Fog computing task scheduling optimization based on multi-objective simplified swarm optimization. Journal of Physics: Conference Series, 1411(1): 012007. https://doi.org/10.1088/1742-6596/1411/1/012007

[21] Saad, M., Enam, R.N., Qureshi, R. (2024). Optimizing multi-objective task scheduling in fog computing with GA-PSO algorithm for big data application. Frontiers in Big Data, 7: 1358486. https://doi.org/10.3389/fdata.2024.1358486

[22] Arulkumar, V., Lathamanju, R., Nithya, T., Rajendran, T. (2025). Enhancing task scheduling process in fog computing using GTO-SSSA: A metaheuristic approach. Journal of Intelligent Systems & Internet of Things, 14(1): 114-128. https://doi.org/10.54216/JISIoT.140109

[23] Liu, X., Yu, J., Wang, J., Gao, Y. (2020). Resource allocation with edge computing in IoT networks via machine learning. IEEE Internet of Things Journal, 7(4): 3415-3426. https://doi.org/10.1109/JIOT.2020.2970110

[24] Deng, S., Xiang, Z., Zhao, P., Taheri, J., Gao, H., Yin, J., Zomaya, A.Y. (2020). Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method. IEEE Transactions on Industrial Informatics, 16(9): 6103-6113. https://doi.org/10.1109/TII.2020.2974875

[25] Lakhan, A., Mastoi, Q.U.A., Elhoseny, M., Memon, M.S., Mohammed, M.A. (2022). Deep neural network-based application partitioning and scheduling for hospitals and medical enterprises using IoT assisted mobile fog cloud. Enterprise Information Systems, 16(7): 1883122. https://doi.org/10.1080/17517575.2021.1883122

[26] Diamanti, M., Charatsaris, P., Tsiropoulou, E.E., Papavassiliou, S. (2022). Incentive mechanism and resource allocation for edge-fog networks driven by multi-dimensional contract and game theories. IEEE Open Journal of the Communications Society, 3: 435-452. https://doi.org/10.1109/OJCOMS.2022.3154536

[27] Baburao, D., Pavankumar, T., Prabhu, C.S.R. (2023). Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. Applied Nanoscience, 13(2): 1045-1054. https://doi.org/10.1007/s13204-021-01970-w

[28] Abdullah, A.A., Manaa, M.E. (2025). Optimizing resource allocation in 5G networks through enhanced particle swarm optimization leveraging deep deterministic policy gradient techniques. Intelligent Decision Technologies, 19(5): 3340-3359. https://doi.org/10.1177/18724981251350319