

Energy Centric Optimised Task Offloading in Fog Computing

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

Abstract—Nowadays, the vast amount of data produced by Internet of Things (IoT) devices requires computational power and storage capacity with significant energy consumption. Cloud computing offers ample storage and computing power but suffers from high transferring with high energy consumption resulting in increasing emissions of greenhouse gases such as CO_2 . Fog computing has become a supplementary approach, bringing computation closer to the data source on fog nodes to reduce transferring time with low energy usage. This paper presents a novel energy-centric optimised task offloading algorithm based on a meta-heuristic algorithm to optimise task allocation, distributing computational loads across fog nodes. Our approach considers multiple objectives, including energy consumption like computing, transfer, memory energy, and makespan, to enhance resource utilisation and meet the difficult requirements of modern IoT applications. Additionally, we introduce a new meta-heuristic algorithm with dynamic coefficients in the fitness function to address load distribution and low iteration, keeping CPU usage within an optimal range which is the most significant factor in decreasing energy consumption. The simulated result of our approach showed a better result by minimising the makespan and reducing the energy consumption as compared to single objective meta-heuristic algorithms.

Index Terms—Task scheduling · Offloading · Fog computing · Cloud computing · Meta-heuristic · Internet of Things (IoT)

I. INTRODUCTION

The growth of Internet of Things (IoT) technology has greatly raised the demand for effective data processing solutions. The rise of cloud computing has transformed the way software is written, delivered, and used and cloud services have steadily grown in popularity since their beginnings. While cloud computing provides significant storage and processing capabilities, the physical distance between cloud data centres and IoT devices causes latency issues, causing difficulties for time-sensitive IoT applications [1]. Many IoT applications require near real-time answers, which standard cloud infrastructures struggle to deliver due to intrinsic latency. Fog Computing emerges as a complement to Cloud Computing, with the goal of processing IoT-generated tasks closer to the data source on devices known as fog nodes. Fog Computing decreases latency and improves responsiveness in delay-sensitive applications by moving computing and storage closer to IoT devices. Furthermore, Fog Computing can increase security for sensitive IoT data by

reducing dependency on third-party cloud providers and storing data locally. The combination of Cloud and Fog computing can suit the different needs of IoT applications, but it poses the difficulty of efficient job offloading. To optimise resource use, effective work offloading in fog situations must take into account metrics such as energy usage and makespan. The complexity of this task needs powerful offloading algorithms that can dynamically distribute resources while minimising energy consumption and processing time [2].

Improper offloading systems leading to misloaded distribution across resources can result in excessive energy consumption and increased emissions of greenhouse gases such as CO_2 , ultimately contributing to global warming. Therefore, an efficient offloading system has become critical for reducing energy consumption in computational models. Ineffective management of load distribution on resources like fog nodes can cause a surge in CPU usage in some nodes, which is the most significant factor in rising energy consumption. By distributing the load efficiently across different nodes and keeping CPU usage within an optimal range, energy consumption can be reduced. In addition to CPU usage, other factors such as memory and transfer energy, while less significant, also play a role in optimising energy usage. Another critical parameter is makespan – the total execution time of all tasks – where minimising this can reduce overall processing time. However, it is important to note that an excessive reduction in makespan could lead to higher CPU usage, resulting in increased energy consumption. This research aims to establish a balance between makespan and key energy-related parameters such as computing, transfer, and memory energy, in order to achieve optimal energy efficiency.

In this context, we provide a meta-heuristic offloading technique meant to optimise task allocation over the edge and fog layers. Figure 1 illustrates how our approach uses a multi-layered architecture with edge and fog layers to balance the computational load and increase overall system performance. In this study, load distribution is acknowledged as a parameter for multi-objective optimisation. Determining the QoS objectives also heavily depends on user requirements [3]. Scheduling applications employ distinctive enhancement meta-heuristic techniques, such as the PSO, Crow Search Algorithm (CSA), etc, to optimise QoS objectives effectively.

We will use a meta-heuristic algorithm to reduce the energy and time consumption of offloading system by iteratively looking for optimal task distribution options, with a focus on some factors like computing, transfer, memory energy, and makespan. Important contributions of this paper include the following:

1) In order to significantly improve execution completion time, energy consumption, and resource utilisation in a heterogeneous edge and fog computing environment, we propose a novel meta-heuristic algorithm that optimises multiple objectives, including computing energy consumption, data transfer energy consumption, memory energy consumption, and makespan.

2) The second contribution of this paper is the introduction of a new meta-heuristic algorithm with some dynamic coefficients for every parameter in the fitness function. This algorithm can solve complex scheduling problems without the traditional constraints like resource limitation used in earlier studies, and it can also achieve an answer in less iteration.

3) An efficient offloading system for distributing tasks among numerous computing fog nodes. The goal of task distribution is loading efficiently across different nodes and keeping the CPU usage within an optimal range, resulting in reduced energy consumption. Excessive load on the CPU more than optimal range leads to a sharp increase in energy consumption. In this paper, energy consumption is reduced by utilizing a new fitness function that distributes the tasks across all nodes.

The rest of this paper is organised as follows. The section II presents existing studied research and analysis of existing task scheduling algorithms in Fog Computing. The section III illustrates the proposed system model and algorithms. The section IV explains experimental parameter setting, simulation results and comparison of the proposed algorithm with existing algorithms. The conclusion and future works are presented in the section V.

II. RELATED WORKS

This section has covered the architecture and different fog computing task offloading algorithms that are currently in use. By managing the resource diversity and interconnection of fog nodes through a fog layer, resource-aware partitioning technique, fog computing expands the capabilities of cloud computing while minimising resource waste and optimising application service placement [4].

Khaledian et al. [5] proposed an energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment, where a hybrid Particle Swarm Optimization and Simulated Annealing algorithm (PSO-SA) is used for prioritising tasks and improving fitness function. They followed a user and company satisfaction, as users prefer to have their demands answered quickly and without delay, and firms who provide services prefer to have their costs decreased. The energy consumption for memory and data transfer was not taken into account, despite the fact that their computing energy usage carries significant costs. In their approach, a successful IoT task scheduling technique reduces energy usage while meeting the deadline and job priority requirements to address the aforementioned issues. They considered task scheduling as

a mixed-integer linear programming (MILP) problem, with the goal of completing IoT activities before the stated dates while prioritising priority and limiting energy consumption. They also consider deadline breach time.

Arri et al. [6] proposed an energy optimization-based trade-off scheme for job scheduling in fog computing platforms, where tasks are assigned to Virtual Machines (VMs) according to their CPU use, RAM, and storage capacity. They used an Artificial Neural Network (ANN) in conjunction with the Artificial Bee Colony (ABC) algorithm as an optimization tool to develop a quick and reliable mechanism for virtual machine (VM) allocation targeted at effective task completion. In their approach, the Modified Best Fit Decreasing (MBFD) algorithm has constraints that the ABC algorithm attempts to overcome by optimizing the merging of virtual machines (VMs) for both newly received requests and VMs that are fully utilized. By doing this, it is made sure that virtual machines are sent to the fewest possible fog servers, which lowers the possibility of assigning too many or incorrect tasks to mobile manipulators. Despite taking into account the CPU, RAM, and energy consumption in their fitness function, it has not been included time as a decision-making factor.

You et al. [7] presented an effective edge computing task offloading approach, where heterogeneous edge servers receive tasks produced by smart edge devices from the Industrial Internet of Things (IIoT). They used a discrete Particle Swarm Optimization (PSO) technique to solve the compute offloading strategy's lowest delay issue. In their approach, they used a technique that combined low latency and energy efficiency in their dynamic multi-local calculations and offloading decisions. They did not account for memory energy use, even though they took computation and transmission energy usage into consideration. The relative efficiency of the PSO-based offloading technique was demonstrated through simulation studies, whereby it was compared with the Genetic Algorithm (GA) and Simulated Annealing Algorithm (SA). Akraminejad et al. [8] presented a multi-objective Crow Search Algorithm (CSA) for optimising both makespan and costs in scientific cloud workflows (CSAMOMC), where using dynamic scheduling strategies in virtualised environments offers various benefits to cloud service users and providers. They followed an efficient resource management that decreases power consumption, enhances utilisation, and accelerates task completion by reducing makespan. In their approach, the initial population used a combination of Breadth-First Search and Depth-First Search, intending to reduce computational costs by attaining faster convergence in the search for potential solutions. Their work intended to improve task execution and resource provisioning by providing a novel CSA, which is inspired by the relevance of cloud workflow scheduling as well as the necessity to reduce costs and timelines without considering the energy consumption.

Delavar et al. [9] proposed a heuristic method (HDECO) for optimizing energy and cost in cloud processes that use multiple instances. To prevent several servers from turning on or off, an intelligent threshold detector was created to monitor

CPU utilization and determine when to alter the CPUs. They tested the novel threshold detector on the Google dataset but did not consider scientific procedures or meta-heuristics. In their approach, energy and cost were decreased by using virtual machine migration by considering hybrid parameters, where a dynamic integration of energy and cost-aware VMs based on predicted resources and heterogeneous hosts was used.

Chafi et al. [10] proposed a novel particle swarm optimization (PSO)-based technique for workflow time and energy optimization in heterogeneous fog computing environments. The algorithm optimizes task allocation in a workflow across available host and fog resources. In their approach, a multi-objective scientific workflow scheduling technique attempted to reconcile competing goals such as energy consumption and scheduling reliability. While this approach stresses the complexities of fog-cloud work scheduling, it does not conduct a comparative analysis with other multiobjective optimization strategies. They achieved effective resource use and increased workflow performance by dynamically changing host and fog resources based on job allocation and using a fitness function that balances time and energy constraints. The experimental setup required adjusting several parameters within iFogSim to recreate realistic. Varshney et al. [11] also used Ant Colony Optimization (ACO) to efficiently schedule workflows in Fog-Cloud environments. ACO's capacity to discover near-optimal solutions makes it ideal for dynamic and developing situations. They used the ACO algorithm to improve task mobility among fog nodes and reduce response time. They used the Java-based FogWorkflowSim simulator to examine the performance of standard scheduling algorithms such as FCFS, SJF, and RR in cloud and hybrid cloud-fog environments.

III. PROPOSED METHOD

This section presents the design and implementation of our multi-objectives offloading algorithm. The algorithm aims to optimise the allocation of tasks in the available fog resources. In the following, we will first present an example of the offloading problem. Then, we will introduce the problem in a more formal manner and present our proposed algorithm, which is designed to solve it.

A. Offloading problem

The fog computing, illustrated in Figure 1, comprises a three-tier architecture including an edge layer, a fog layer, and a cloud layer. In this model, edge devices as task generators such initiate task execution requests directed towards the higher computing layer like the fog layer. These requests from various edge devices are aggregated and managed by a scheduler situated within the fog layer. The scheduler on the fog layer decides where the tasks will be executed.

In fog computing, tasks are offloaded based on optimising various evaluation parameters to achieve a distribution of tasks and the best performance in higher layers with more capacities. The goal of task offloading is to find the optimal solution for allocating a number of tasks to available nodes in the fog layer.

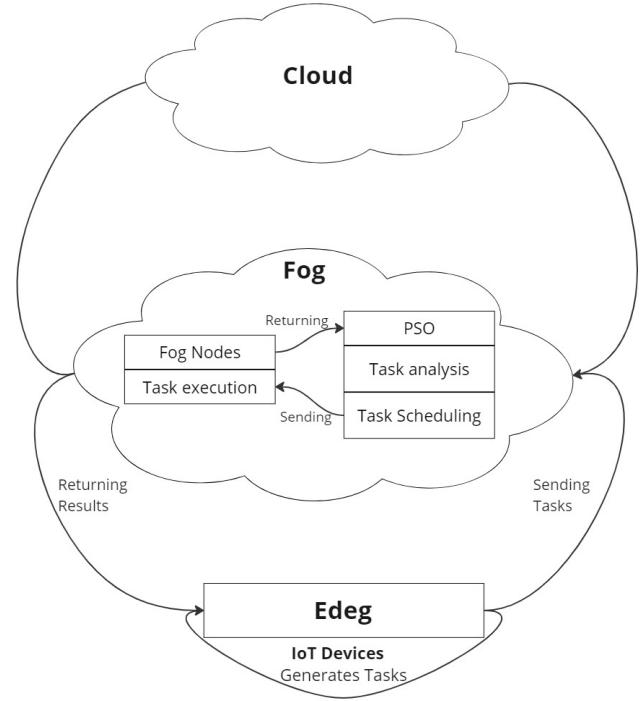


Fig. 1. Edeg-to-cloud offloading system

To address this problem, the proposed model considers the following assumptions:

- All submitted tasks are independent.
- Tasks are generated by edge devices.
- Energy consumption of fog nodes in the minimum power – the machine/device is turned off, but still plugged into the energy source – mode is considered negligible.

B. Problem formulation

In order to optimise the amount of time and energy used for processing tasks, an algorithm is proposed in this paper. The algorithm leverages the power of multiple objectives, utilising available time, CPU and RAM resources effectively. The best answer is found by using meta-heuristic methods. Depending on our goals, we can customise any meta-heuristic algorithm using its fitness function. According to Equation (1), the fitness function in our method is designed to minimise resource consumption (RC), including time T_i^t and energy consumption T_i^e that is defined as a collection of Tasks = { T_1, T_2, \dots, T_w }.

$$\text{Min} \sum_{i=0}^w T_i^e + T_i^t \quad (1)$$

The time of execution of a task is determined by multiplying the time required to complete the task at a specific node by the cost associated with that node. The total resource consumption for the task is then calculated based on its computing, transfer, memory value, and execution time on the selected node. Communication time is not explicitly included in

the calculation; instead, they are indirectly accounted for within the makespan. The overall optimisation of the task offloading is obtained by summing the four various factors of fog nodes, where $\text{FN} = \{\text{FN}_1, \text{FN}_2, \dots, \text{FN}_z\}$, represented as:

$$\begin{aligned} F(X) &= \sum_{i=0}^z \sum_{j=0}^w \\ &\alpha * \text{FN}_{ij}^{ce} + \beta * \text{FN}_{ij}^{te} + \gamma * \text{FN}_{ij}^{me} + \delta * T_{ji}^{et} \end{aligned} \quad (2)$$

The fitness function – used on the meta-heuristic algorithms – denoted as $F(X)$, Equation 2, plays a pivotal role in assessing system performance. The coefficients $\alpha, \beta, \gamma, \delta$ represent the weights for node i and task j assigned to computing energy FN_{ij}^{ce} , transfer energy FN_{ij}^{te} , memory energy FN_{ij}^{me} , and execution time (T_{ji}^{et}), respectively. In this context, these coefficients are used as follows:

α : Weight assigned to computing energy

β : Weight assigned to transfer energy

γ : Weight assigned to memory energy

δ : Weight assigned to execution time

$$\alpha + \beta + \gamma + \delta = 100 \quad (3)$$

As it is shown in equation 3, the sum of these coefficients is typically constrained to equal 100 to ensure the total weight is distributed proportionally among all factors.

$$\text{FN}^{ce} = \sum_{i=0}^z N_i^{cu} * E_i^{cu} \quad (4)$$

The calculation of Equation 4 involves multiplying the number of compute units utilised N_i^{cu} by the energy usage per compute unit E_i^{cu} of node i for processing allocated tasks based on the energy model in section III-C. This calculation helps in making decisions about node selection, ensuring that nodes with insufficient processing energy capacity, which could fail to complete the task or shut down midway, are not chosen.

$$\text{FN}^{te} = \sum_{i=0}^z N_i^{bw} * E_i^{bw} \quad (5)$$

Furthermore, equation 5 shows the energy consumption per bandwidth E_i^{bw} multiplied by the number of bandwidth used N_i^{bw} , which can influence decisions based on the data transfer energy consumption. This ensures that tasks are allocated to the appropriate node with minimal energy requirements for data transfer based on the energy model in section III-C.

$$\text{FN}^{me} = \sum_{i=0}^z \frac{1}{2} C_i V_i^2 f_i \quad [12] \quad (6)$$

$$T^{et} = \sum_{j=0}^w \frac{CU_j^t}{\text{MIPS}} \quad (7)$$

Equation 6 describes the energy consumed by memory, referred to as memory energy FN_i^{me} for node i . High memory energy consumption can result in certain nodes not being assigned tasks, as it may be inefficient to keep tasks in memory for extended periods while waiting for previous tasks to be processed. Instead, tasks should remain in the queue until they can be efficiently processed which consumes energy. Moreover, the execution time in millions of instructions per second (MIPS) indicates that one million tasks are handled in a second, which we compute the time based on the MIPS capability of the device and the task resource unit requirement CU_j^t in equation 7. To further explain our approach to minimising energy consumption and execution time, we integrate a detailed energy model, in section III-C, monitoring power usage across various computing nodes and transfer links, ensuring precise tracking and optimisation of resource efficiency throughout the infrastructure. This facilitates time-based decision-making, guaranteeing that nodes with high processing or transfer time usage are not given tasks. As a result, tasks are distributed to more appropriate nodes, resulting in an even distribution of duties.

C. Modelling energy consumption

Our energy model for infrastructure energy use assigns a unique energy model to each computing node and transfer link to enable precise energy consumption monitoring across cloud, fog nodes, edge devices, and their transfer connections including WAN and Wi-Fi. Edge devices can connect to fog nodes and one another over Wi-Fi. Cloud and fog nodes use WAN to transport data, and the quantity of energy needed can vary based on the infrastructure that the devices and nodes use. Our computing energy FN_{tot}^{ce} returns the total computing energy usage at any point in time, calculated as the sum of energy static E_{st} and energy dynamic E_{dy} usage for all fog nodes FN_i^{ce} . Our transfer energy FN_{ij}^{te} includes the amount of energy consumption for transfer tasks.

The model considers the following computing energy levels, including idle energy, which is running without carrying out tasks, maximum energy, which is the CPU fully utilised, and minimum energy, which is the device turned off but plugged in. It supports two energy-draining behaviours dynamic, using linear interpolation between idle and maximum energy, and static, which can handle any energy value. The dynamic model is the default for high-energy states, handling idle energy with minimum values and defining the consumption range by subtracting idle from maximum energy [13]. The electronic system like fog node and edge device, equation 8, involves voltage (V measured in volts) and metered electric current (I measured in amps) values that can be used to calculate the electric energy (P measured in watts) like $1W = 1V \cdot 1A$ [12].

$$P = V * I \quad (8)$$

For example, total energy consumption in equation 9 can frequently be accurately modelled based on the dynamic – entity's current load C_t – and static energy. In equation 10,

calculate dynamic energy consumption, the variable σ indicates the incremental energy per unit of load.

$$FN_{tot}^{ce} = E_{st} + E_{dy} \quad (9)$$

$$E_{dy} = C_t \sigma \quad (10)$$

For instance, in network equipment in equation 11, σ denotes the energy consumed by a number of instructions per second transferred (J/bit). When modelling resource-constrained compute nodes or transfer links that have a maximum load C_{max} with energy usage C_{max} , σ can be defined as:

$$\sigma = \frac{E_{max} - E_{st}}{C_{max}} \quad (11)$$

D. Multi-objective proposed algorithm

As shown in Figure 2, once the tasks are generated, the meta-heuristic algorithm like Particle swarm optimisation (PSO) initialises parameters, including the swarm size, which determines the number of particles in the PSO swarm, and velocity limits, which govern the particles' movement and exploration capabilities. A key element of meta-heuristic algorithms is the fitness function, which we can tailor to meet our goals. In the context of multi-objective optimization, we can make well-informed decisions by taking into account a number of parameters, including energy and makesapn. Moreover, coefficients are set to balance the exploration and exploitation abilities of the algorithm in the fitness function that can consider the effectiveness of each parameter to make a decision. For example, if we require low energy consumption, the coefficient in those fitness function parameters should be higher, as this can result in rejecting nodes with high energy consumption. Once the PSO parameters are initialised, the algorithm proceeds to the update phase, where it dynamically adjusts fog resources according to the current task allocation. In this phase, the algorithm takes into account several indirect parameters, including computational capabilities, storage capacity, and network bandwidth that have effects on our objectives.

The idea behind the meta-heuristic algorithm is to find the optimal solution by comparing the cost of each annoyance with its predecessor after it has occurred. The coefficients α, β, γ , and, δ in the fitness function $F(X)$ represent the relative importance of different factors in evaluating the fitness of a solution. Therefore, employing coefficients can aid in obtaining the optimal solution with less iteration, which may result in the usage of less computer power and a faster algorithm. The coefficients in the fitness function can influence how tasks are allocated to different nodes in the fog layer. If the coefficients prioritise energy efficiency too strongly, some nodes may be left in a low-power state, meaning they are not used to their full potential or may take longer to complete tasks due to their lower power consumption. On the other hand, the coefficients can also help ensure that tasks are spread evenly across all available nodes, balancing the load and preventing some nodes from being overworked while others are underutilised. This balance

helps optimise overall performance and energy efficiency. This indicates tasks have been sent to all devices leading to spreading workload over the network. In this context, these coefficients are used as follows:

The parameters of the coefficients can be chosen and optimised using an ascent-based optimisation. We can adjust the coefficient to have less of a negative impact on the distribution of resources based on previous results, such as computing energy, transfer energy, memory energy, and makespan. For example, we can modify the coefficients in the direction of the gradient ascent – for maximising – if the latency and execution time were increased by the prior data and learning rate, a tiny positive number that regulates the step size. In this case, the coefficients are initialised to the same value. Based on the previous values of $\alpha_{prev}, \beta_{prev}, \gamma_{prev}, \delta_{prev}$ as well as previous results from factors like FN^{ce} , FN^{te} , FN^{me} , and T^{et} , a gradient ascent approach is used to estimate these coefficients. Every freshly computed coefficient is restricted to be between 0 and 100. For instance, if a specific parameter like time has used too many resources, we will use a gradient ascent methodology to upwards coefficient for execution time. Thus, nodes with lengthy execution duration can be excluded from consideration for other tasks as a result of this modification. It is necessary to update the coefficients repeatedly based on new results. In the following, we present the pseudo-code of the proposed algorithm:

Algorithm 1 Gradient Ascent Algorithm For Calculation of Coefficient

Input:

Previous coefficients: $\alpha_{prev}, \beta_{prev}, \gamma_{prev}, \delta_{prev}$

Previous results: $FN_{prev}^{ce}, FN_{prev}^{te}, FN_{prev}^{me}$, and T_{prev}^{et}

Learning rate η

Number of iterations

Output:

Predicted coefficients: $\alpha, \beta, \gamma, \delta$

Start:

for each iteration $i = 1$ to iterations do

 Compute gradients for the objective function

 Update coefficients with directional constraints:

 Project coefficients to the range [0, 100]

end for

End

As mentioned in Algorithm 1, Gradient Ascent uses the following inputs to calculate the coefficient values such as previous coefficients $\alpha_{prev}, \beta_{prev}, \gamma_{prev}, \delta_{prev}$, previous results such as compute, transfer, memory energy, and execution time which aid in the prediction of new coefficients, Learning rate – a small positive number that controls the step size –, and Number of iterations. New coefficient values will be anticipated and then updated with the existing ones – using algorithm 1 in fitness function algorithm 2 – based on the previous results and coefficient values. Every time the heuristic algorithm iterates, this procedure is changed resulting in the optimum resource

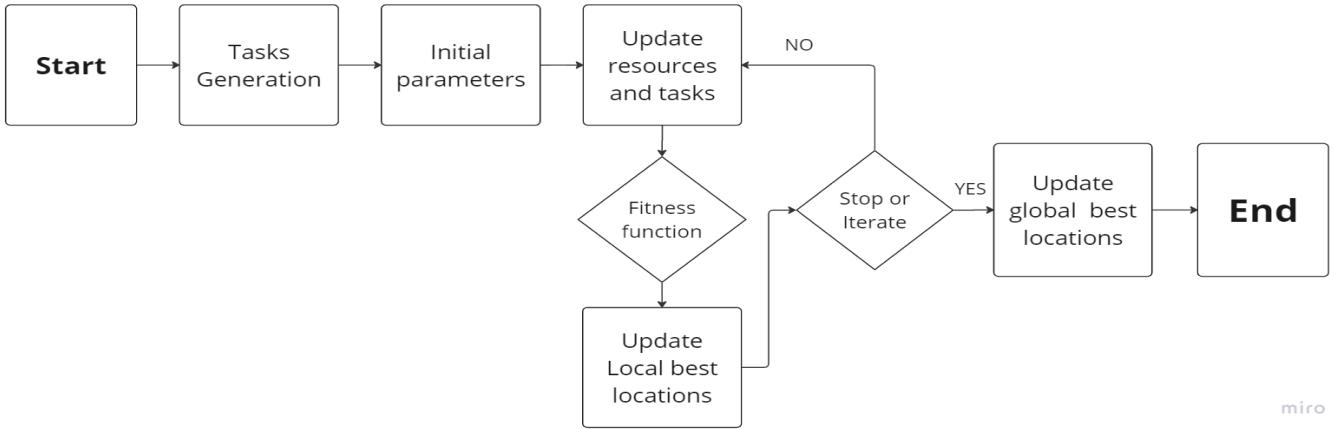


Fig. 2. Meta-heuristic optimisation algorithm flow.

allocation decision being made with fewer iterations.

As shown in Figure 1, our system consists of a set of edge devices and a set of F fog nodes, which are interconnected. Tasks, denoted as T , are randomly generated at each edge device, and their arrivals are assumed to be independent. We assume that fog computing provides computing resources, represented as $N = F \cup T$, where N is the total number of nodes.

The method evaluates the simulation by calculating several performance measures, such as energy consumption, makespan, and resource utilization, after updating the resources. The simulation results and coefficients are then updated, allowing for the task allocation approach to be improved in later iterations. These results are stored by the algorithm, which allows for ongoing enhancements to the work allocation procedure. Next, it checks whether the stopping criterion based on parameters such as computing energy, transfer energy, memory energy, or a defined makespan has been met. If the global best solution is achieved, the algorithm retrieves the optimal task allocation, indicating that the best possible solution has been found. This global best allocation represents the most efficient distribution of tasks to resources, considering the established performance objectives. **If, however, the stopping condition is not satisfied, the algorithm repeats the simulation process, enabling further refinement of the task allocation strategy.** The iterative nature of the algorithm enables it to adapt and improve the best task allocation based on updated information and simulation results. The proposed fitness function algorithm is as shown in Algorithm 2.

As shown in the swarm intelligence algorithm 3, the meta-heuristic algorithm like PSO is initiated by random initialisation and solutions. In each iteration, a new solution is found by making local movements over the current solution. In each iteration, particles are updated with two values, namely personal best (pBest) and global best (gBest). pBest is the best solution achieved so far, whereas the gBest is the second-best value obtained by any particle in the population. If the stopping condition is met, the algorithm retrieves the optimal task

Algorithm 2 Fitness Function

Input:

Set of Fog Nodes $N = \{N_1, N_2, \dots, N_i\}$

Set of Tasks $T = \{T_1, T_2, \dots, T_i\}$

Output

Fitness Value V

Start:

for each Task $T_i \in T$ **do**

 Calculate Minimum Computing Energy using Eq. (4)

 Calculate Minimum Transfer Energy using Eq. (5)

 Calculate Minimum Memory Energy using Eq. (6)

 Calculate Minimum Makespan using Eq. (7)

 Calculate Fitness Value using Eq. (2)

end for

 Calculate new coefficients from algorithm 1

End

allocation, indicating that the optimal solution has been reached. This optimal task allocation represents the most efficient distribution of tasks to resources, considering the defined performance objectives. However, if the stopping condition is not satisfied, the algorithm repeats the simulation process, refining the task allocation strategy further. The iterative nature of the algorithm enables it to adapt and improve task allocation based on updated information and simulation results. The algorithm continues to iterate until the optimal solution is obtained or the stopping condition is met.

IV. PERFORMANCE EVALUATION

The proposed method has been developed in both the proposed particle swarm optimisation (P-PSO) and proposed Crow Search Algorithm (P-CSA) algorithm, which is examined below [8]. To evaluate their performances, we compared the proposed multi-objectives PSO and CSA algorithms and single objectives (baseline) B-PSO and B-CSA that the original meta-heuristic algorithms are based on. **TableI** collects the values of

Algorithm 3 Energy-Efficient Task offloading algorithm

Input:

 Set of Fog Nodes $N = \{N_1, N_2, \dots, N_i\}$,

 Set of Tasks $T = \{T_1, T_2, \dots, T_i\}$
Output

 Tasks allocated to fog nodes = $\{N_{1i}, N_{2i}, \dots, N_{ji}\}$
Start:
for each Particle $p \in N, T$ **do**
for each iteration I **do**

Calculate Fitness from Algorithm 2

Calculate personal best value of the particle

if (Fitness value > pBest) **then**

pBest = Fitness value

end if

Calculate global best value of the particle

if (Fitness value > gBest) **then**

gBest = Fitness value

end if

Update particle velocity using

Update particle position using

end for
end for

Store Optimal solution

End

the parameters of the energy consumption for compute, transfer, and memory energy adopted in the experiments.

For experimentation, we take into account makespan and energy like compute, transfer, and memory energy on the fog environment made up of a variety of heterogeneous fog nodes with interconnect network topology. The number of FNs fluctuates from 50 to 200 for numerous trials, and the number of IoT tasks varies from 200 to 1000. The infrastructure graph comprises three types of compute nodes: cloud, fog, and edge devices. Edge IoT devices – task generators – connect to nearby fog nodes using Wi-Fi, such as IEEE 802.11p. Data flow mappings to network edges are updated periodically and fog nodes also use Wi-Fi to communicate with nearby base stations, forming a mesh network. Furthermore, they have the capability to connect to the cloud via WAN. WAN links consume more energy per bit; however, direct Wi-Fi communication between fog nodes is more energy-efficient than communication with edge devices due to the constant direct line of sight and the use of energy-efficient, high-throughput access points [15].

To calculate MIPS (Millions of Instructions Per Second) for each CPU, we can use the Tensilica Xtensa 32-bit LX6 microprocessor that is used in the ESP32. Except for one module, the ESP32-S0WD, which employs a single-core system, usually uses a dual-core design. The CPU processing capacity of each heterogeneous fog node is between 600 to 1000 MIPS at a clock frequency of up to 240 MHz. The amount of compute energy consumption for active mode (dynamic), idle mode (static), transfer energy – WAN and Wi-Fi, and memory energy is shown at tableI based on the monitoring

application described in [16].

TABLE I
INFRASTRUCTURE POWER AND COMMUNICATION SPECIFICATIONS

Parameter	Value
Power (static)	0.1 W/MIPS
Power (dynamic)	0.36 W/MIPS
Power (memory)	212.866 μ Wh
Edge to Fog (Wi-Fi)	1.3 Gbit/s - 300 nJ/bit
Fog to Fog (Wi-Fi)	1.3 Gbit/s - 100 nJ/bit
Fog to Cloud (WAN)	50 Mbit/s - 6658 nJ/bit
Cloud to Fog (WAN)	20 Mbit/s - 572 nJ/bit

A. Energy and makespan diagrams

Table II shows how the number of tasks affects algorithm energy consumption. As can be observed in table II, the simulation results demonstrate that an increase in the number of tasks imposes a heavier load on the system in general leading to increasing energy consumption. Here, we assume the number of FNs to be constantly 100 and the number of tasks were computed from a sparse deployment with 250 tasks to a dense workload consisting of 1000 tasks. As table II shows, with an increment of the number of tasks, the proposed scheme results in more energy consumption. For instance, in P-PSO with 250 to 1000 tasks, energy consumption is increase from 48.45 to 178.01 watt because, with more tasks, it needs more energy to compute all tasks. As the table indicates, the proposed CSA and PSO method is considerably energy efficient, and helps fog nodes to conserve their energy and thus prolongs network lifetime particularly for dense task deployment in which the problem of lifespan optimisation becomes more complicated than B-PSO and B-CSA deployments.

TABLE II
TOTAL ENERGY CONSUMPTION FOR DIFFERENT NUMBER OF TASKS

Tasks	P-PSO	B-PSO	P-CSA	B-CSA
250	48.45	52.76	46.95	49.66
500	91.49	97.27	90.06	95.68
750	135.36	141.25	135.27	139.52
1000	178.01	185.01	176.01	183.32

TABLE III
TOTAL ENERGY CONSUMPTION FOR DIFFERENT NUMBER OF NODES

Nodes	P-PSO	B-PSO	P-CSA	B-CSA
50	177.09	195.69	178.11	188.41
100	179.2	187.05	181.36	190.71
150	181.06	189.55	183.12	193.05
200	183.66	191.08	187.96	197.96
250	186.88	194.06	189.8	201.58

As table III indicates, how the number of nodes affects algorithm energy consumption. Here, we assume the number of nodes was computed from a sparse deployment with 50 to 250 nodes to a dense workload consisting of 1000 tasks. The proposed scheme significantly reduces energy consumption for resource allocation optimisation and thus, prolongs lifetime of

fog nodes. The amount of energy consumption of the proposed PSO and CSA algorithm comparing with baseline PSO and CSA with 250 nodes is 186.88, 189.8, 194.06, and 201.58 watt, respectively. Since each node uses energy in the idle mode, increasing the number of nodes has a negative impact on energy consumption. Consequently, the amount of energy consumed for the scenario with more nodes is higher, as you can see in table III with different numbers of nodes and compare to table II with different numbers of tasks.

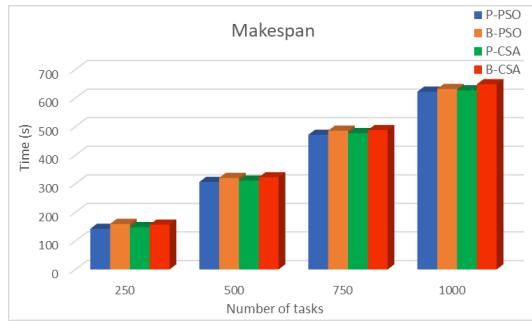


Fig. 3. Makespan for different number of tasks

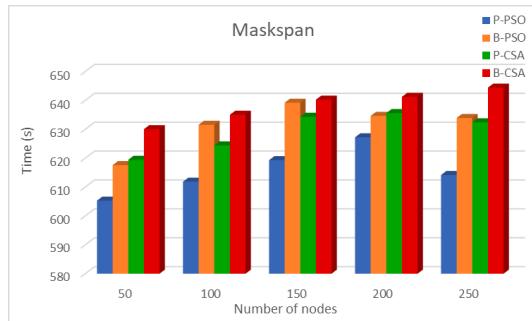


Fig. 4. Makespan for different number of nodes

Figure 3 shows the makespan resulting from the proposed method in terms of task density form 250 to 1000 tasks with 100 fog nodes. As Fig. 3 shows, with an increment of tasks in the workload, the makespan increases to compute all tasks. For instance, with 1000 tasks, the makespan in B-PSO and P-PSO is raised from 622.47 to 632.41 because, with more tasks, it needs more time to compute 1000 tasks on 100 nodes. In other word, makespan in baseline algorithms like B-PSO (from 159.55 to 632.17) and B-CSA (from 156.96 to 648.68) is more than proposed algorithms including P-PSO (from 142.39 to 622.47) and P-CSA (from 148.27 to 627.48). Moreover, figure 4 illustrates the makespan for the number of nodes were computed from a sparse deployment with 250 nodes to a dense workload consisting of 1000 tasks. Because additional tasks require more time to calculate 1000 tasks on 100 nodes, the makespan in B-PSO and P-PSO is increased from 611.85 to 631.85. Stated differently, makespan in baseline algorithms such as B-PSO (ranging from 617.58 to 633.92) and B-CSA (spanning from 630.03 to 644.42) exceeds that of proposed

algorithms such as P-PSO (ranging from 605.29 to 614.16) and P-CSA (spanning from 619.44 to 632.41).

B. Impact of coefficient

For each value of the mentioned parameters, the algorithm was independently run 30 times for each stated parameter. Each run of the proposed algorithm is implemented on a randomly generated simulated workload and node deployment. Figure 5 shows the average rate of iterations for the best candidate solution — global best — and then, results from the proposed method in terms of task density. The number of fog nodes was 100 and the number of tasks were computed from a sparse deployment with 250 tasks to a dense workload consisting of 1000 tasks. As Fig. 5 shows, with an increment of tasks in the workload, the swarm intelligence algorithms result in more iterations to find the best solution. For instance, with 1000 tasks, the iteration rate is raised from 18 to 66 [P-PSO] because, with more tasks, it needs more iteration to find the best solution. It is noticeable that the impact of the proposed scheme on decreasing the iteration rate of highly dense workload is less than that of middle-dense and sparse task deployment. For example, in the condition consisting of 250 tasks in P-PSO (please observe Fig. 5), the algorithm decreases the iteration rate from 24 to 18 (6 decrements), while in the condition with 1000 tasks which is a high workload, the algorithm decreases the iteration rate from 80 to 66 (26 decrements). The number of iteration for 1000 tasks in B-PSO and P-PSO is decreased from 80 to 66 (26 decrements), while in B-CSA and P-CSA is increased from 82 to 79 (only 3 decrements).

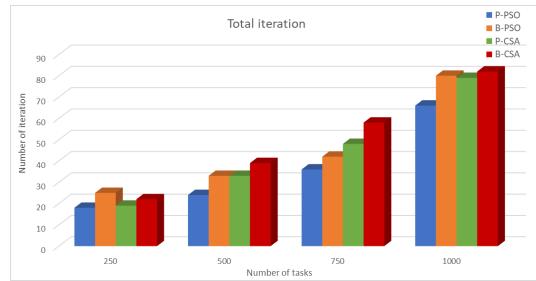


Fig. 5. Total iteration rate for different number of tasks

Figure 6 illustrates the load distribution rate for 1000 tasks that are split equally among 50 nodes, guaranteeing a balanced load distribution across the whole nodes on the network. As Fig. 6 shows, each node computed a number of tasks with energy and time consumption. It is noticeable that the impact of the proposed scheme on distributing the number of tasks across a number of nodes is highly valuable. For example, the proposed PSO algorithm developed tasks across 50 nodes with a minimum and maximum number of tasks 9 and 30 (19 average), respectively, while in the Baseline PSO algorithm, the minimum and maximum number of tasks allocated to one node is 11 and 33 (22 average). Additionally, a minimum and maximum number of tasks for P-CSA 0 and 33 (16 average), respectively, while in the Baseline CSA algorithm, the minimum and maximum number of tasks allocated to one

node is 0 and 39 (22 average). As you can see, some nodes in the B-CSA, like 47, 48, 49, and 50, have no tasks assigned to them, while other nodes, like 17, 18, 19, 32, 34, 35, and 36, must compute more than 30 tasks, which puts a lot of pressure on them and causes their energy consumption to increase significantly. The reason behind the high average in a B-PSO and B-CSA is that the fitness function did not consider whole objectives like transfer or memory energy for task allocation. In other words, coefficients in the proposed scheme – P-PSO and P-CSA – can help to have a balance in task allocation with stopping to allocate tasks to low capacities – compute, transfer, memory capacity – fog nodes. Therefore, the average rate in the task allocation is not considerably increased while the lifespan is enhanced since energy consumption increases at a significantly higher rate than the rise in CPU usage, demonstrating a non-linear relationship between resource use and energy demand.

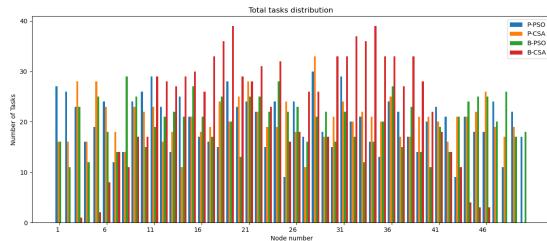


Fig. 6. Total applications distribution

V. CONCLUSIONS

In conclusion, this paper addresses the critical challenge of task offloading optimisation in heterogeneous fog computing environments, which is essential to meet the demanding requirements of modern Internet of Things (IoT) applications. By introducing a novel energy-centric task offloading algorithm based on a meta-heuristic approach, we effectively distribute computational loads across fog nodes, reducing transfer time and energy consumption. Our algorithm incorporates multiple objectives, including computing, transfer, memory energy, and makespan, to optimise resource utilisation and achieve energy efficiency. The use of dynamic coefficients in the fitness function ensures flexible and adaptive load distribution while keeping CPU usage within an optimal range, thus further minimising energy consumption. In other words, the proposed scheme decreased the number of iterations to obtain optimise solution for task allocation with using dynamic coefficients. The proposed multi-objectives approach demonstrates superior performance in both reducing the makespan and improving energy efficiency compared to single-objective algorithms. Future work will focus on integrating additional methods to further optimise task scheduling solutions, as well as evaluating a broader range of existing algorithms to achieve even more efficient outcomes. We will also explore distributed communication models such as P2P and gossip protocols to address task scheduling in decentralized fog environments.

Finally, we aim to develop distributed multi-objective task scheduling frameworks, prioritising availability, security, and reliability for fog computing systems.

REFERENCES

- [1] Abbasi, Shirin, Navid Khaledian, and Amir Masoud Rahmani. "Trust management in the internet of vehicles: a systematic literature review of blockchain integration." International Journal of Information Security (2024): 1-24.
- [2] Samani, Zahra Najafabadi, Nishant Saurabh, and Radu Prodan. "Multilayer resource-aware partitioning for fog application placement." In 2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC), pp. 9-18. IEEE, 2021.
- [3] Souri, Alireza, Amir Masoud Rahmani, Nima Jafari Navimipour, and Reza Rezaei. "A hybrid formal verification approach for QoS-aware multi-cloud service composition." Cluster Computing 23 (2020): 2453-2470.
- [4] Samani, Zahra Najafabadi, Nishant Saurabh, and Radu Prodan. "Multilayer resource-aware partitioning for fog application placement." In 2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC), pp. 9-18. IEEE, 2021.
- [5] Khaledian, Navid, Keyhan Khamforoosh, Reza Akraminejad, Laith Abualigah, and Danial Javaheri. "An energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment." computing 106, no. 1 (2024): 109-137.
- [6] Arri, Harwant Singh, and Ramandeep Singh. "Energy optimization-based optimal trade-off scheme for job scheduling in fog computing." In 2021 8th international conference on computing for sustainable global development (INDIACOM), pp. 551-558. IEEE, 2021.
- [7] You, Qian, and Bing Tang. "Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things." Journal of Cloud Computing 10 (2021): 1-11.
- [8] Akraminejad, Reza, Navid Khaledian, Amin Nazari, and Marcus Voelp. "A multi-objective crow search algorithm for optimizing makespan and costs in scientific cloud workflows (CSAMOMC)." Computing (2024): 1-17.
- [9] Delavar, Arash Ghorbannia, Reza Akraminejad, and Sahar Mozafari. "HDECO: A method for Decreasing energy and cost by using virtual machine migration by considering hybrid parameters." Computer Communications 195 (2022): 49-60.
- [10] Chafi, Saad-Eddine, Younes Balboul, Mohammed Fattah, Said Mazer, and Moulhime El Bekkali. "Novel PSO-Based Algorithm for Workflow Time and Energy Optimization in a Heterogeneous Fog Computing Environment." IEEE Access (2024).
- [11] Varshney, Saiyam, and Gur Maaj Saran Srivastava. "Efficient Workflow Scheduling in Fog-Cloud Environments using Ant Colony Optimization." In 2024 Second International Conference on Data Science and Information System (ICDSIS), pp. 1-5. IEEE, 2024.
- [12] Vogelsang, Thomas. "Understanding the energy consumption of dynamic random access memories." In 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 363-374. IEEE, 2010.
- [13] Márkus, András, and Attila Kertész. "Modelling Energy Consumption of IoT Devices in DISSECT-CF-Fog." In CLOSER, pp. 320-327. 2021.
- [14] Ahvar, E., Orgerie, A., and Lébre, A. (2019). Estimating energy consumption of cloud, fog and edge computing infrastructures. IEEE Transactions on Sustainable Computing, EA:1-12, DOI: 10.1109/TSUSC.2019.2905900.
- [15] Wiesner, Philipp, and Lauritz Thamsen. "Leaf: Simulating large energy-aware fog computing environments." In 2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC), pp. 29-36. IEEE, 2021.
- [16] Fanariotis, Anastasios, Theofanis Orphanoudakis, and Vassilis Fotopoulos. "Reducing the Power Consumption of Edge Devices Supporting Ambient Intelligence Applications." Information 15, no. 3 (2024): 161.