An-Najah University Journal for Research – A

Natural Sciences

Internet of Things Based Application Placement Technique in Fog Environment

N Malathy[1,*], M Ruba[1] & S Vinothini[1]

(Type: Full Article). Received: 5th Dec. 2024, Accepted: 1st May. 2025, Published: 1st Feb. 2026,
DOI: https://doi.org/10.35552/anujr.a.40.1.2497

Abstract: Fog computing bridges the gap between IoT devices and cloud servers by providing low-latency computational resources closer to the network edge. Despite its potential, the rapid increase in IoT applications with diverse resource and quality-of-service (QoS) requirements presents significant challenges in application deployment and resource optimization. This paper addresses these challenges by introducing a comprehensive application placement framework designed to optimize execution time and energy consumption in a heterogeneous fog environment. The proposed framework consists of three phases. A pre-scheduling method is developed to efficiently allocate tasks by analyzing workflows to reduce computation delays and energy usage. Leveraging an Improved Memetic Algorithm (IMA), this strategy enables effective scheduling of parallel IoT workflows across fog and cloud servers, ensuring balanced resource utilization and enhanced scalability. A lightweight recovery method is incorporated to address runtime failures, ensuring the robustness and reliability of task execution. The performance of the proposed framework is evaluated using real and synthetic IoT workflows in the iFogSim environment. Experimental results demonstrate that the framework achieves a 65% reduction in the weighted cost and a 51% decrease in execution time compared to existing approaches. This makes it a promising solution for managing resource-intensive IoT applications in fog computing environments.

Keywords: Application Placement, Scheduling, Partitioning, Fog Computing, Improved Memetic Algorithm.

Introduction

Human life has been greatly enhanced by the extensive use of Internet of Things (IoT) devices in a variety of fields, including intelligent transportation, smart healthcare, and industrial automation [1]. Due to the massive volumes of data produced by these devices, sophisticated and latency-sensitive Internet of Things applications have emerged, such as online gaming, video streaming, augmented reality, and virtual reality. By 2030, there will be roughly 3.5 billion connected devices, according to Cisco reports and other IoT research studies.[19] These IoT devices usually transfer data processing tasks to more powerful computing layers because of their limited computational and energy resources. Using a pay-as-you-go model, cloud computing provides scalable solutions for networking, storage, computation, and management. However, maintaining stringent Quality of Service (QoS) requirements is still difficult because of the enormous volume of data produced by IoT devices and the considerable physical distance between users and cloud data centers. Traditional cloud-based models are inappropriate for IoT deployments because latency-sensitive applications require ultra-low response times, high availability, security, and guaranteed QoS.

To overcome these restrictions, the Open Fog Consortium unveiled Fog Computing, a framework intended to reduce the latency, storage inefficiencies, and bandwidth restrictions that come with cloud computing. To improve real-time capabilities, this paradigm emphasizes data processing and intelligence near the data production location. Fog Computing is an extension of cloud computing that was founded in 2015 by prominent technology companies like ARM, Cisco, Dell, Intel, Microsoft, and Princeton. It integrates various network layers while maintaining important advantages like virtualization, orchestration, and efficiency. However, putting a fog network into practice necessitates carefully weighing several variables, such as CPU, RAM, and network capacity, as well as computational, communication, and storage demands. [21,22] Before implementing a fog service model, other factors like anticipated IoT service requests, service types, execution environments, and mobility need to be determined. Smart gateways, routers, and base stations are important networking elements of the fog infrastructure that offer virtualized computer resources to satisfy real-time responsiveness demands [17].

All of the benefits of cloud computing are combined with extra features like mobility management and context-aware services in the three-layer Fog Computing (FC) architecture. For mobile-based IoT applications like the Internet of Vehicles and vehicular IoT systems, this architecture is especially helpful [19,20,24]. Fog computing dramatically lowers network propagation delays by positioning computing resources close to IoT devices. IoT applications are software as a service that, when they receive IoT data, perform a variety of tasks. For subsequent tasks like pre-processing decision analysis, these IoT data must be calculated in real-time. Every Internet of Things application consists of a collection of modules, tasks, or services that need different configurations of computing resources to operate. Finding the services' availability is a very difficult task for the service provider because of the diverse and dynamic behavior of these IoT applications that estimate precise computing resources. In addition to IoT applications' diversity and dynamic nature, different IoT applications have different needs for real-time responsiveness depending on the situation.

It is essential to map the set of services to the available computing resources to meet at least one of the goals, which will simplify the challenges of IoT applications and optimize the QoS index of different IoT application use cases [18,23,26]. Finding

1 Department of Information Technology, Mepco Schlenk Engineering College.
* Corresponding author email: malathy@mepcoeng.ac.in

the best computing resources for diverse IoT services is the goal of the IoT service placement problem (SPFC). Making the best choice for service placement addresses several problems, such as meeting deadlines, maximizing the deployment of IoT applications by optimizing resource usage, and effectively balancing loads to prevent overload and underload, among others. On the other hand, determining the best mapping choice for heterogeneous services is a well-known NP-complete problem [21]. As a result, the majority of authors made decisions that were almost ideal for their work [15-18] In addition to determining the best way to map different services in fog-cloud infrastructure, determining whether the objectives are single- or multi-objective increases SPFC's complexity. As a single objective optimization problem, the majority of authors developed SPFC using various models, such as linear programming (LP), integer linear programming (ILP), mixed integer linear programming (MILP), mixed integer non-linear programming (MINLP), etc. [21, 14-19,2]. A single objective optimization centered on either network usage, QoS maximization, cost minimization, or energy consumption minimization is the foundation of the majority of recent SPFC works [19, 20]. Nonetheless, there are numerous instances in which an IoT user aims to maximize multiple goals.

A single objective optimization appears unrealistic and unfeasible in these situations. In the literature, many authors have formulated SPFC as a biobjective or multiobjective optimization problem [16,25,27]. With multiple optimization objectives, the Multi-objective SPFC seeks to determine the best mapping between the collection of IoT application services and computing resources. These goals, which typically conflict and come from the perspective of the IoT user or service provider, include maximizing performance, maximizing reliability, minimizing energy consumption from the service provider's perspective, and minimizing costs from the user's perspective. Consequently, it appears feasible to formulate SPFC as a multiobjective, and it is crucial to optimize each objective function at an acceptance level without letting another solution dominate. Finding the best placement strategy for diverse heterogeneous IoT applications made up of separate services is the main focus of this paper. To solve the problem, a hybrid algorithm based on meta-heuristics is suggested. Finding a good trade-off between makespan, energy, and cost for a set of IoT tasks in the fog cloud system is the main goal of the hybrid meta-heuristic approach. Furthermore, the weighted sum multi-objective optimization can be readily converted to aggregated objective functions for multiple objectives, giving the user the freedom to select the objective functions' priority by selecting the appropriate weight.

Literature Survey

This section discusses relevant studies for application placement techniques in fog computing environments, where cloud and fog servers collaborate to meet the needs of IoT applications. Based on the dependency model of their IoT application's constituent pieces, they are categorized into independent and dependent categories (e.g., tasks). Each IoT application can be thought of as a collection of tasks that are either independent or dependent. The dependent one refers to programs that are made up of numerous dependent tasks, each of which runs only when its previous tasks have been completed. The tasks of the programs in the independent one, on the other hand, do not have such execution limits.

Independent Tasks

In Mobile Edge Computing networks, the computing tasks of many wireless devices are offloaded to multiple edge servers and one cloud server. Taking into account various real-time compute tasks at various wireless devices, each task is determined whether it should be performed locally at the wireless device and should be processed either in edge servers or the cloud server. Low-complexity computation offloading rules are used to ensure mobile edge computing network quality of service while reducing wireless device energy usage. For mobile edge computing networks, both a linear programming relaxation-based (LR-based) and a distributed deep learning-based offloading (DDLO) technique are found separately. In comparison to DDLO, heterogeneous DDLO can help achieve greater convergence performance. The DDLO methods offer greater performance than the LR-based algorithm, according to extensive numerical studies. Furthermore, the DDLO algorithm generates an offloading decision in less than 1 millisecond, which is multiple times faster than traditional algorithms. The LR-based algorithm is orders of magnitude faster [2].

Offloading with consideration for latency and power consumption is a promising subject in the realm of mobile cloud computing nowadays. The cloudlet concept has evolved to allow latency-aware offloading. Offloading an application to the most appropriate cloudlet, on the other hand, remains a significant difficulty. Cloudlets can handle a variety of applications. The type of application is checked when a request for task offloading arrives from a mobile device. The most appropriate cloudlet is chosen from a pool of cloudlets near the mobile device based on the type of application. The energy consumption of mobile terminals can be decreased using an application-aware cloudlet selection method. By dispersing the processes to be offloaded in various cloudlets, an application-aware cloudlet selection approach for multi-cloudlets can balance the load on the system. As a result, the chance of putting all loads on a single cloudlet for load balancing can be calculated [3]. Managing the transmission power of mobile devices and the assigned server computation while preserving their latency threshold reduces their energy consumption and computational cost in a multilayered Mobile Edge Computing system [4].

Fog Computing seeks to process data at the network's edge. Transmission delay, monetary cost, and application loss caused by Cloud Computing can all be decreased with Fog Computing. Because fog nodes have lower processing capacity than cloud platforms, running all apps on these nodes may cause some QoS requirements to be breached. As a result, crucial decisions must be made about where to execute each program to develop a cost-effective solution that meets all application requirements. The unit-slot optimization is a quantified near-optimal solution for balancing the three-way tradeoff between average response time, average cost, and an average number of application failures.

In a three-tier Cloud of Things system, the unit-slot optimization technique can provide cost-effective processing while ensuring average response time and average application loss [5]. Fog computing attempts to provide Cloud-like services at the network edge to enable the Internet of Things (IoT) applications that demand fast responses. Application deployment in Fog is difficult due to the hierarchical, dispersed, and heterogeneous nature of computing instances. The application placement challenge is exacerbated by differing user expectations and diverse functionalities of IoT devices. The placement of apps to compatible Fog instances based on user expectations can improve the system's Quality of Experience (QoE).

A QoE-aware application placement policy prioritizes distinct application placement requests based on user expectations and assesses Fog instances' capabilities based on their present status. It also makes it easier to arrange applications on appropriate Fog instances in the Fog computing environment,

ensuring that user QoE is maximized in terms of utility access, resource usage, and service delivery. The policy reduces data processing time, network congestion, resource affordability, and service quality dramatically [6]. Consider a multi-user mobile cloud computing system with a computing access point (CAP), in which each mobile user has numerous independent tasks that can be completed locally, at the CAP, or on a remote cloud server. For mobile users, the CAP serves as a network access gateway as well as a computing service provider.

To minimize the overall cost of energy, computation, the offloading decisions of all users' jobs as well as the allocation of computer and communication resources. Semi-definite relaxation (SDR), alternating optimization (AO), and sequential tuning (ST) are efficient three-step algorithms that always calculate a locally optimal solution and yield approximately optimal performance under a wide range of parameter values. Evaluating SDR-AO-ST's performance against a lower bound on the least cost, purely local processing, purely cloud processing, and hybrid local-cloud processing without the use of the CAP [7]. The goal of minimizing each task's computation time and energy consumption in the Industrial Internet of Things–edge–cloud computing architecture is to formulate the joint problem in which the Industrial Internet of Things (IIoT) devices select their computation-offloading methods. A finite improvement path to Nash equilibrium can be ensured using a free–bound method. The Nash equilibrium can be achieved with the help of a multi-hop cooperative messaging method and two QoS-aware distributed algorithms [8].

Dependent Tasks

A partitioning technique that transfers computation-intensive workloads from a single mobile device to a single edge or cloud server. The mobile device's placement engine is installed to discover a group of jobs to offload and lower the mobile application's execution time and energy consumption [9][10]. To reduce the time needed for IoT applications to run in an environment where multiple fog servers and a cloud server are readily accessible for application placement, which only evaluates one mobile device in their offloading system model and reduces mobile device power consumption by offloading some computation to the cloud server [11]. To execute multi-user jobs at the cloud server with low communication overhead and tasks at the edge layer with larger communication overhead [26].

The communication cost of transferring data from the IoT layer's sensors and devices to the fog layer during the scheduling process [12]. To ensure the quality of service (QoS) of application in a fog environment, which meets service delivery deadlines and maximizes resource utilization. A latency-aware application deployment policy was suggested in a system with numerous fog servers and a single cloud server [13]. To compute a task, we need both the user task data and the program that processes it as input. The use of caching at the Mobile Edge Computing (MEC) system to dynamically store program and/or task data has lately been acknowledged as a cost-effective approach to reducing compute time, energy usage, and bandwidth cost. It provides a strategy for joint optimization of service caching placement and computation offloading, even though the above-mentioned techniques focus on task placement as their primary goal [14].[15] suggested a batch job placement based on a Genetic Algorithm (GA), in which numerous users' mobile applications are forwarded to a single central edge server for placement decisions. The scheduling of various workflows using metaheuristics algorithms is discussed in [13]-[15]. The application placement in fog computing sussing opposition-based memetic algorithm was discussed in [16]-[18].

Several benefits have been found by current research on IoT service placement, such as effective offloading strategies, where low-complexity algorithms and deep learning-based frameworks have successfully decreased execution time and energy consumption in mobile edge computing environments. By allocating tasks according to application requirements, cloudlet selection and load-balancing techniques have improved system performance and energy efficiency. Additionally, by using strategies like latency-aware placement and QoE-aware policies, QoS and QoE optimization techniques have enhanced resource utilization and latency-sensitive task handling. Cooperative offloading and Nash equilibrium-based solutions are useful for optimizing shared resource usage in multi-user and multi-task scenarios. Additionally, flexible and dynamic application deployment has been made possible by hybrid approaches that integrate edge, fog, and cloud computing, guaranteeing performance gains under various circumstances.[25]

Existing work, however, also poses several difficulties. Inefficient task placement under high loads is a result of many approaches' limited scalability, which makes it difficult to manage large-scale IoT environments with a large number of devices and resource-intensive workflows.[27] High computational complexity is also an issue because, despite their effectiveness, optimization methods like genetic algorithms frequently result in excessive computational overhead and decision times, which makes them inappropriate for real-time applications. The reliability of IoT deployments may also be jeopardized by insufficient failure recovery, as only a small number of studies have addressed runtime failures during task execution. Furthermore, despite real-world restrictions like limited virtual machine capacities and variable bandwidth, some models assume infinite resources in edge or fog servers, ignoring resource constraints. Lastly, there are still issues with task dependencies because many studies overlook the intricate relationships between IoT application tasks, which results in inefficient task placement and execution order. These pros and cons highlight the contributions and gaps in prior research, forming the basis for the improvements proposed in the paper.

The main contributions of this paper are as follows:

- We propose a novel weighted cost model that simultaneously minimizes execution time and energy consumption for IoT applications in a hybrid fog-cloud environment. This model accounts for the limited computational resources of fog servers while ensuring efficient application deployment.
- A pre-scheduling approach is developed to efficiently organize and prioritize tasks based on their dependencies and workflows. This technique reduces computation delays and energy usage, enhancing the performance of resource-constrained IoT devices.
- A new batch application placement strategy is introduced, leveraging the Improved Memetic Algorithm (IMA) to optimize the placement of parallel IoT workflows. The algorithm incorporates local search techniques and diversity factors to avoid local optima and improve convergence speed.
- To enhance system reliability, we design a lightweight failure recovery mechanism that efficiently reallocates failed tasks to alternative servers, minimizing disruption and ensuring robust execution.
- The proposed framework is evaluated using real and synthetic workflows in the iFogSim simulator. Experimental results demonstrate up to a 65% reduction in weighted cost

and a 51% improvement in execution time compared to state-of-the-art approaches.

Materials and Methods

System Model

We consider a framework that includes various IoT devices, fog servers, cloud servers, and brokers, in which IoT devices execute their workflows (i.e., DAGs) locally or offload them entirely or partially to cloud and/or fog servers. Our system model is shown in Figure 1 at a high level. Each broker in this system framework has the means to support up to N IoT devices in its immediate area. In this work, Directed Acyclic Graphs (DAGs) are used to model IoT application workflows. Each node in the DAG represents a specific computational task, and the directed edges represent dependencies between tasks, indicating the execution order. This representation enables efficient scheduling, task placement, and optimization in a fog-cloud environment while respecting the constraints imposed by task dependencies and resource availability.

Problem Formulation

The task placement problem is formulated as an optimization problem to minimize the overall execution time of IoT applications as well as IoT energy consumption. The available servers are represented as N_1 with $|N_1|$=Q because there are multiple servers available to run the jobs $V_{n,\,i}$. The term an identifies the type of server, and the $N_1^{a,b}$ denotes one server. Then a=0 is an IoT device, a=1 is a fog server, a=2 is a cloud server, and b is the server index, T_n signifies the offloading configuration for each task, while $t_{n,i}$ denotes the workflow configuration for the nth IoT device. It is calculated using equation 1.

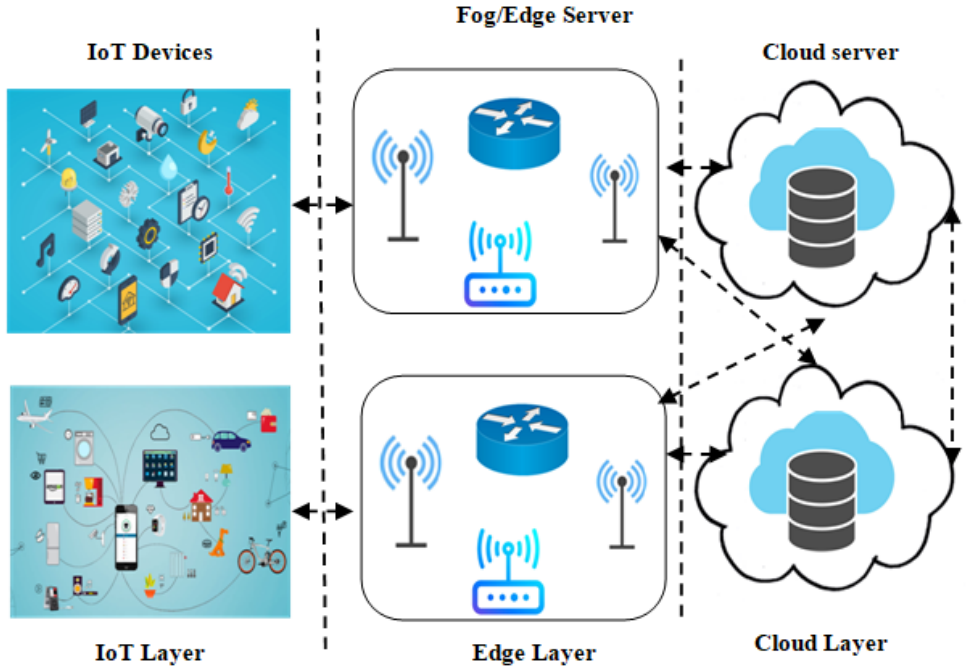


Figure (1): Overview of the Proposed System.

$$t_{n,i} = \begin{cases} 0, {N_1}^{a,b} = {N_1}^{0,n} \\ 1, {N_1}^{a,b} \epsilon \{{N_1}^{1,1}, {N_1}^{1,2} .. {N_1}^{1,F1}\}, |b| = f_1 \\ 2, {N_1}^{a,b} \epsilon \{{N_1}^{2,1}, {N_1}^{2,2} .. {N_1}^{1,c1}\}, |b| = C_1 \end{cases} \quad (1)$$

Here, $t_{n,i}$ =0 denotes that the task is assigned to the nth IoT device for local execution, $t_{n,i}$ =1 denotes that the task is assigned to either a cloud server or a fog server for remote execution. The f_1 and c_1 denote the number of fog servers and cloud servers.

Weighted Cost Calculation

The purpose of the task placement technique is to select the optimal configuration of available servers for each IoT application to limit the weighted cost of execution for each IoT device, as illustrated below in equations 2 to 6:

$$\min_{\psi_\alpha, \psi_\beta \in [0,1]} \psi(T_n), \forall n \in \{1,2,3,\ldots N\} \quad (2)$$

$$\psi(T_n) = \psi_\alpha * \frac{\Gamma(T_n)}{\Gamma Local_n} + \psi_\beta * \frac{\Theta(T_n)}{\Theta Local_n} \quad (3)$$

Show that,

$$C_{11}: VM_{fog,i} \leq C_{fog,i}\ \forall i\ \in {N_1}^{1,1} \ldots {N_1}^{1,f}\} \quad (4)$$

$$C_{12}: |t_{n,i}| = 1, \forall_i \epsilon \{1,2,N\}\, t_{n,i} 1 \leq j \leq |v_n| \quad (5)$$

$$C_{13}: \Psi(W(V_n, i)) \leq \Psi(W(V_n, i) + V_n, i) \quad (6)$$

Where $\Gamma(T_n), \Theta(T_n), \Gamma Local_n, \Theta Local_n$ denote the IoT device's execution time, energy consumption, and local execution time and energy consumption. Furthermore, ψ_α and ψ_β are control parameters for execution time and energy consumption, respectively, that allow the weighted cost model to be customized to the user's requirements. We also assume that the task can be given to a single Virtual Machine (VM) on a fog or cloud server. The number of VMs on the ith fog server $VM_{fog,\,i}$ is fewer than or equal to the fog server's maximum capacity, $C_{fog,\,i}$, as indicated by C_{11}. C_{12} indicates that each time slot can only have one server allocated to job I from the workflow of the nth IoT device, C_{13} indicates that V_n, i predecessor ith tasks must be completed before the original task may be completed.

Execution Time Model

When ψ_α = 1 and ψ_β = 0 is used in Eq. (3), and the weighted cost optimization is equivalent to the execution time model. The purpose of the execution time optimization model is to identify the shortest possible execution time. The best setup for the application that is running on the computer nth IoT device, such that the application's execution time is reduced. The total latency

in task offloading ($\Gamma_{T_n}^{lat}$) can be used to calculate the entire execution time of each candidate configuration, the time it takes for tasks in a workflow to be computed based on the servers they've been assigned ($\Gamma_{T_n}^{exe}$) and the time it takes for data to be transmitted between each pair of dependent jobs ($\Gamma_{T_n}^{tra}$) workflow [is] given below in equation 7:

$$\Gamma(T_n) = \Gamma_{T_n}^{exe} + \Gamma_{T_n}^{lat} + \Gamma_{T_n}^{tra} \quad (7)$$

The computational execution time for the application operating on the nth IoT device is computed as follows in equation 8:

$$\Gamma_{T_n}^{exe} = \sum_{t_{n,i} \epsilon T_n} \gamma_{t_{n,i}}^{exe} \quad (8)$$

where $\gamma_{t_{n,i}}^{exe}$ denotes the computing time of task $V_{n,i}$ and it is calculated based on the server which is assigned from the following equation 9:

$$\gamma_{t_{n,i}}^{exe} = \begin{cases} \frac{h_{n,i}^{w}}{local^{cpu}}, & t_{n,i} = 0 \\ \frac{h_{n,i}^{w}}{Speed\_F^{f1} * local^{cpu}}, & t_{n,i} = 1 \\ \frac{h_{n,i}^{w}}{Speed\_F^{c11} * local^{cpu}}, & t_{n,i} = 2 \end{cases} \quad (9)$$

here $local^{cpu}$ denotes the IoT device computing power $Speed\_F^{f1}$ and $Speed_F^{c1}$denotes the speedup factor of fog and cloud servers. Based on the tasks assigned to servers, the offloading latency $\Gamma_{T_n}^{lat}$ of tasks corresponding to the nth IoT device is estimated using equation 10:

$$\Gamma_{T_n}^{lat} = \sum_{t_{n,i} \epsilon T_n} \gamma_{t_{n,i}}^{lat} \quad (10)$$

where $\gamma_{t_{n,i}}^{lat}$ demonstrates the offloading task $V_{n,i}$ is calculated based on the server which is assigned from the following equation 11:

$$\gamma_{t_{n,i}}^{lat} = \begin{cases} 0, & t_{n,i} = 0 \\ \mathrm{Lat_{LAN}}, & t_{n,i} = 1 \\ \mathrm{Lat_{WAN}}, & t_{n,i} = 1 \end{cases} \quad (11)$$

where $\mathrm{Lat_{LAN}}$ and $\mathrm{Lat_{WAN}}$, denotes the latency of LAN and WAN. The transmission time of the task's workflow to the nth IoT device is calculated using equation 12:

$$\Gamma_{T_n}^{tra} = \sum_{t_{n,i} \epsilon T_n} \gamma_{t_{n,i}}^{TRA} \quad (12)$$

where the transmission time of dependent tasks $V_{n,i}$ and $V_{n,j}$ is calculated using equation 13:

$$\gamma_{e_{n,i,j}}^{tra} = \begin{cases} \frac{e_{n,i,j}^{w}}{BW_{LAN}}, & CCT_i = CCT_1, CCT_3 \\ \frac{e_{n,i,j}^{w}}{BW_{wAN}}, & CCT_i = CCT_2, CCT_4 \\ 0, & CCT_i = CCT_3 \end{cases} \quad (13)$$

where $BW_{LAN}$and BW_{wAN} denotes the bandwidth of LAN and WAN.

Energy Consumption Model

When ψ_α = 1 and ψ_β = 0 are equal, and the weighted cost optimization equals the energy consumption model, according to Eq. (2). The goal of the energy consumption model is to discover the best feasible configuration of the application's tasks to reduce the energy consumption of the nth IoT device. The total energy consumed by any candidate configuration can be calculated as the sum of the energy consumed in each component($\Theta_{T_n}^{lat}$) job offloading and the energy used in task computation ($\Theta_{T_n}^{exe}$), as well as the energy used for the data exchange between each pair of dependent tasks($\Theta_{T_n}^{tra}$) of the application, as described in equation 14:

$$\Theta(T_n) = \Theta_{T_n}^{exe} + \Theta_{T_n}^{lat} + \Theta_{T_n}^{tra} \quad (14)$$

The amount of energy used to compute the nth IoT device's application is calculated as follows using equation 15:

$$\Theta_{T_n}^{exe} = \sum_{t_{n,i} \epsilon T_n} \theta_{t_{n,i}}^{exe} \quad (15)$$

where $\theta_{t_{n,i}}^{exe}$ denotes the amount of energy required to do the task $V_{n,i}$, as computed in the following formula mentioned in equation 16:

$$\theta_{t_{n,i}}^{exe} = \begin{cases} \gamma_{t_{n,i}}^{exe} * POW_{cpu}, & t_{n,i} = 0 \\ \gamma_{t_{n,i}}^{idle} * POW_{idle}, & t_{n,i} = 1,2 \end{cases} \quad (16)$$

where POW_{cpu} is the IoT device's CPU power, on which the task's $V_{n,i}$ runs. Because we only consider energy consumption from the perspective of IoT devices, whenever each task is offloaded to fog servers ($t_{n,i}$) or cloud servers ($t_{n,i}$), the respective energy consumption is equal to the IoT device's idle time $\gamma_{t_{n,i}}^{idle}$multiplied by the power consumption of that device in its idle mode POW_{idle}.The energy used to offload tasks corresponding to the nth IoT device $\Theta_{T_n}^{lat}$ is computed as follows using equation 17:

$$\Theta_{T_n}^{lat} = \sum_{t_{n,i} \epsilon T_n} \theta_{t_{n,i}}^{lat} \quad (17)$$

where $\Theta_{t_{n,i}}^{lat}$ denotes the offloading consumption of the task $v_{n,i}$ and is computed using equation 18:

$$\theta_{t_{n,i}}^{lat} = \begin{cases} 0, & t_{n,i} = 0 \\ \gamma_{t_{n,i}}^{idle} * POW_{idle}, & t_{n,i} = 1,2 \end{cases} \quad (18)$$

where $\theta_{t_{n,i}}^{lat}$ denotes the offloading consumption of the task $V_{n,i}$ and is demonstrated from:

The transmission energy consumption $\Theta_{T_n}^{tra}$for the nth IoT device is calculated as follows using equation 19:

$$\Theta_{T_n}^{tra} = \sum_{t_{n,i} \epsilon T_n} \theta_{t_{n,i}}^{tra} \quad (19)$$

where the transmission energy between the pair of dependent tasks $V_{n,i}$ and $V_{n,j}$ is calculated using equation 20:

$$\theta_{e_{n,i,j}}^{tra} = \begin{cases} \frac{e_{n,i,j}^{w}}{BW_{LAN}} * POW_{trans}, & CCE_i = CCE_1 \\ \frac{e_{n,i,j}^{w}}{BW_{wAN}} * POW_{trans}, & CCE_i = CCE_2 \\ 0, & CCE_i = CCE_3 \end{cases} \quad (20)$$

The CCE displays transmission configuration for each edge $e_{n,i,j}$ based on the assigned servers of its tasks to compute the transmission energy, which is determined from the transmission power of the IoT device, which is designated as POW_{trans} and it is calculated using equation 21

Table (1): Summary of Symbols and Notations.

Symbol	Description
N_l	Set of available servers (fog and cloud servers).
A	Type of server (a=0: IoT device, a=1: fog server, a=2: cloud server).
B	Index of a specific server within a type.
T_n	Offloading configuration for tasks of the nth IoT device.
$V_{n,i}$	i^{th} task in the workflow of the n^{th} IoT device.
f_i, c_i	Number of fog servers and cloud servers
τ	Execution time of a task.
E	Energy consumption of a task.
λ_{LAN},λ_{WAN}	Latency in Local Area Network (LAN) and Wide Area Network (WAN).
B_{LAN}, B_{WAN}	Bandwidth of LAN and WAN.
ξ	Weighted cost optimization parameter for execution time.
η	Weighted cost optimization parameter for energy consumption.
$SI(V_{n,i})$	Server index assigned to task Vn,i.
$\mu_{fog,i}$, $\mu_{cloud,i}$	Speedup factors of the fog and cloud servers.
P_{CPU}	CPU power of the IoT device.
P_{idle}	Idle power consumption of the IoT device.
P_{trans}	Transmission power of the IoT device.

$$CCE_i\left(e^{w}_{n,i,j}\right) = \begin{cases} t_{n,i} \oplus t_{n,j} = 1, & i = 1 \\ t_{n,i} \oplus t_{n,j} = 2, & i = 2 \\ otherwise, & i = 3 \end{cases} \quad (21)$$

where CCE_1 signifies data flow between two jobs $V_{n,i}$ and $V_{n,j}$, which are allocated to the IoT device and fog servers, respectively. CCE_2 is also used to depict the interaction between two jobs assigned to IoT devices and cloud servers. Because the transmission energy consumption is calculated from the perspective of the IoT device, it is equal to zero whenever one of the participating jobs in an edge $e^{w}_{n,i,j}$ is not assigned to the IoT device, as shown in CCE_3 .The symbols and the notations used are given in table 1.

Methodology

Application Placement Technique

Ordering, batch application placement, and failure recovery are the three aspects of our suggested application placement technique. Brokers can manage concurrent IoT device processes using an approach presented in the pre-scheduling phase. Then, to minimize the weighted cost of each IoT device, we offer an optimized version of the Improved Memetic Algorithm (IMA) for batch application placement. In addition, we provide a lightweight failure recovery method in our technique to deal with any potential runtime issues.

Ordering Phase

The broker receives workflows from IoT devices. Furthermore, depending on their particular workflows, it evaluates the local execution time and energy consumption of IoT devices. In terms of the quantity and weight of tasks, dependencies, and the amount of data flow between each pair of dependent tasks, IoT device workflows are diverse. Furthermore, each workflow's task execution order should be ordered so that a new task $V_{n,i}$ cannot be run until all tasks in its $W(V_{n,i})$ have completed their execution.

Process of Ordering Phase

Algorithm 1 explains how the pre-scheduling phase organizes tasks of each process and consequently builds schedules and concurrent workflows. In Algorithm 1, the local execution time and energy consumption for each workflow are estimated and saved in Local Time and Local Energy respectively, (lines 3 and 4). DAGs are useful because they can have some root vertices (source nodes), The Find_ Root method locates all of the DAG's root vertices, SRCn (line 5). This method is used to check whether the P(Vn,i) is equal to null or not. The Single Root_ Transformer method creates a new DAG called nDAG which has only one single root (line 6). To attain this, we should create a dummy vertex called n Dummy Root and connect this vertex to all source vertices of SRCn obtained from the original DAG. We can specify the schedule number for each vertex starting from the Dummy Root by using the Breadth-First-search (BFS) algorithm (line 7). The outcome of the overall loop is to provide a scheduled number for every task. This algorithm iterates over concurrent workflows so that the tasks with the same schedule number are arranged in a row of 2D ArrayList called Final2DArrayList.The method gets (x) and adds (Vn,i) are used to access a 2D ArrayList and add a new entry to a list(line12).

This pre-scheduling process is depicted a with an example scenario. Consider there exists two workflows with four and three vertices. The first workflow has one source vertex, while the second workflow contains two source vertices. Following the discovery of the source vertices, the Single Root_ Transformer method forms a new DAG called n DAG with only one source root. Then, the BFS algorithm is applied to schedule each task. While the schedule number for all the tasks is identified, the tasks with the same schedule number should be placed in 2D ArrayList called Final2DArrayList

Batch Application Placement Phase

We propose a batch application placement technique in which an Improved Memetic Algorithm (IMA) is used to decide where tasks in each schedule should be placed. Tasks in each schedule can be conducted in parallel since they are either independent tasks in one processor or tasks from various workflows (with no dependency) that are executed in parallel.

Algorithm 1: Ordering Phase

Input: WOF: List of all workflows

Output: Final2D Array L ist, Local Time,

Local Energy

N = |WOF| (number of workflows)

Iterate through each workflow (n = 1 to N):

- Local Time. Add (Cal Local Exe Time (WOFn))
- Local Energy. Add (Cal Local Exe Energy (WOFn))
- SRCn = Finder_Root (WOFn)
- nDAG = Single Root_Transformer (WOFn, SRCn)
- BFS (nDAG, Duplicate Root)

Iterate through each workflow (n = 1 to N):

For each node in WOFn (i = 1 to |WOFn|):

- integer x = Checking Order Number ($V_{n,i}$)
- Final2DArrayList.get(x). add($V_{n,i}$)

Algorithm 2: Improved Memetic Algorithm

Input: List of Workflows, Final2DArrayList: The 2D ArrayList that contains all schedules.

Output: Con_final, Cost_final

Set Result List MA = null

Iterate through each schedule (i = 1 to S):

- MAResult.get(i) =
  - IMA(Final Ordered List. get(i))
- Con_final =
  - Result Processor (Result L is tMA. get(i))

Iterate through each workflow (n = 1 to N):

- Compute cost: Cost_final [n] =
  - Cost Calculator (Con_final)

Algorithm 2 gives an overview of the suggested batch application placement step. This phase takes as input a list of all workflows WOF and schedules Final2DArrayList and outputs the workflow configuration as Con_ final and the overall execution cost as Cost_ final. Due to a large number of schedules, the Improved Memetic Algorithm (IMA) is used to decide on tasks for the current schedule while taking prior schedule server allocations into account (line 3). Because tasks in each schedule come from one or more workflows, the Result Processor Result L ist MA method collects task assignments from all schedules Result L is tMA, organizes them by workflow, and stores them in a 2D Arraylist called Con_ final, where each row represents one workflow (line 4). The Cost Calculator Con_ final method calculates the execution cost of each workflow based on the relevant obtained configuration once the task assignment for all schedules is completed. The IMA is the key function of this phase.

Improved Memetic Algorithm (IMA)

The Improved Memetic Algorithm (MA) is an algorithm that combines evolutionary-based search methods like GA with one or more refinement methods (e.g., local search, individual

learning) to solve various optimization issues like routing and scheduling. Each candidate solution is represented by an individual in the IMA, and the solution is retrieved from a population of candidate individuals. By using GA functions, we proposed an Improved Memetic Algorithm (IMA), in which a local search method is applied to the selected individuals of each iteration. Each possible configuration of servers assigned to tasks in a single schedule is encoded individually in the IMA. The atomic part of each individual is a gene that describes the task in a schedule and carries a tuple (p,q) here, p defines the type of assigned server and q defines the index of the server. The values of each tuple are derived in which values of the type of assigned server and index of the server are defined.

The Genetic Algorithm is made up of four steps initialization, selection, crossover, and mutation. The IMA is made up of the first four steps of a genetic algorithm and local search, which is a refinement method. The utility of each candidate is evaluated by using a fitness function that enables the IMA to choose the best individuals in every iteration. In the context of the genetic algorithm, a population consists of multiple candidate solutions, referred to as individuals. Each individual is represented as a chromosome, which is further divided into genes that encode specific components of the solution. Fitness values are calculated for each individual to evaluate their quality.

Algorithm 3: IMA Algorithm

Input**:** schedule_ tasks: Set of tasks

Output: Org P List. Get (0)

Set Org P List = null, DivP List = null

Call Initialization (schedule_tasks)

Assign Org PList = Selection (OrgP)

Assign Div PList = Selection (DivP)

Iterate for i = 1 to I:

- Crossover (Org PList, Div P List)
- Mutation (Org P List, Div PList)
- Local Search (OrgPList, Div PList)
- Update Org PList = Selection (OrgP)
- Update Div PList = Selection (DivP)

Initialization

In this step, the maximum number of iterations I, population size P_Size, and other IMA parameters. Individuals in the population are created, and the population is initialized. Moreover, a new population has been added to the Original Population (OrgP). Diversity Population is a term used to describe a strategy for increasing the diversity of solutions (DivP). Because the IMA's major purpose is to determine the best possible server configuration. The cost of local execution downs and a pre-defined individual is assigned and created for the OrgP, in which all gene tuple values are tuples set to the local servers of their choice. Because those whose fitness values are lesser than the pre-defined individual are not selected in the following iterations, the number of low-utility individuals is reduced. In the initialization step, the rest of the individuals in the OrgP and the individuals in the DivP are generated at random.

The solution is represented as an individual in the Improved Memetic Algorithm (IMA), and each individual is associated with a particular task assignment configuration within a specified timeframe. Each gene in the person is a task and is encoded as a tuple (p, q), where q is the server index and p is the type of server (e.g., 0 for an IoT device, 1 for a fog server, and 2 for a cloud server). For example, a gene (1,2) indicates that the second fog server is given the task. A person is made up of several of these genes, which together determine where each task should be placed in a schedule. The number of tasks (n) and the number of available servers (m) determine the size of the search space, which leads to

The number of available servers (m) and the number of tasks (n) determine the size of the search space, which comes out to be m^n. The effective search space is decreased by practical limitations like task dependencies and resource limitations, even though this size increases exponentially with the number of tasks. When working with large search spaces, heuristic algorithms like IMA are especially helpful because deterministic approaches become computationally costly due to exponential growth. Additionally, heuristic methods efficiently provide near-optimal solutions when it is not possible to obtain an optimal solution in a practical timeframe. The intricacy of constraints, such as resource limitations, dependency models, and latency considerations, emphasizes even more how flexible heuristic methods are in comparison to deterministic algorithms.

Fitness Function

For Orginal Population (OrgP), the IMA employs two global and local fitness functions, which are used to assess the utility of each M_p^{orgp} (indiv) indicating the utility of a server's configuration for each task of one workflow on that schedule M_q^{orgp} (Vn,i). The M_q^{orgp} (Vn,i) receives the task Vn,i and local fitness value is calculated [from Equ. (2)] with the execution cost of the unassigned workflow are equal to zero. Algorithm 4 describes how the fitness value of every individual M_p^{orgp} (Vn,i) is calculated. The M_p^{orgp} (Vn,i) is the total of local fitness M_q^{orgp} (Vn,i) of tasks in one schedule. Due to the concurrent execution of tasks of one workflow in every schedule, the local fitness M_q^{orgp} (Vn,i) values of tasks that belong to the same workflow Maxi Loc are calculated (lines 1-11). The Concurrent Task Checking is the method that stores the parallel tasks of workflow in the Concurrent Set (line 3). Moreover, the local fitness of every task in the Concurrent Set is calculated and the maximum local fitness is stored in Maxi Loc (lines 4-10). Thus, the global fitness value is calculated by the sum of all values of MaxiLoc, which stores the maximum local fitness value of each workflow.

Input: indiv: An individual showing tasks of Schedule

Output: pBest

Iterate through each workflow (n = 1 to K):

- Reset Concurrent Set = null
- Concurrent Set = Concurrent Checking (indiv, WO Fn)
- Maxi Loc[k] = Mqorgp (Concurrent Set. get(1))

Iterate through Concurrent Set (i = 1 to |ConcurrentS et|):

- temp = Fqorgp (Concurrent Set[i])
- If temp > Maxi Loc[k], update: Maxi Loc[k] = temp
- Iterate through Maxi Loc (i = 1 to |MaxiLoc|):

P Best = p Best + Maxi Loc. Get (i)

In IMA, the main goal of Diversity Population ($divP$) is to diversify the individuals so that the probability of local optimum decreases. Moreover, the fitness function of $divP$, The $M_p^{divP}(indiv_n^{divP})$ is completely different from $orgP$ and it is calculated as follows using equation 22 and 23:

$$M_p^{divP}(indiv_n^{divP}) = \sum_{i=1}^{Psize} HD\left(indiv_i^{orgP}, indiv_n^{divP}\right) \quad (22)$$

where $Psize$ describes the population size of $orgP$ and $divP$. The individual of $orgP$ and $divP$ is described as $indiv_i^{orgP}$ and $indiv_n^{divP}$.The $HD\left(indiv_i^{orgP}, indiv_n^{divP}\right)$ is the Hamming distance function that calculates the difference between individuals of the assigned server to the tasks and it is described as follows:

$$HD\left(indiv_i^{orgP}, indiv_n^{divP}\right) = \sum_{k=1}^{S} div\_f \quad (23)$$

where f describes the size of each individual. To compute the fitness of one individual of $divP$, we calculate its difference by the number of individuals in the OrgP in Eqs. (22) and (23), and the individual with the higher difference receives a higher fitness value. This aids in the retention of individuals with a greater $divP$ difference who are more diverse than the individuals in the IMA. Since the various type of servers (i.e., Internet of Things, fog, and cloud) with varying numbers of servers in each type (for example, server index) is taken into account by the system. A diversity factor div_f is defined in the model, which explains each task's fitness based on the type and index of the task server that has been assigned as follows using equation 24:

$$df = \begin{cases} 2, sym\left(\left|ST\left(indiv_{i,k^{orgp}}\right) - ST\left(indiv_{r,k^{divp}}\right)\right|\right) = 1 \\ \quad sym\left(\left|ST\left(indiv_{i,k^{orgp}}\right) - ST\left(indiv_{r,k^{divp}}\right)\right|\right) = 0 \\ \qquad\qquad \delta \\ 1, sym\left(\left|ST\left(indiv_{i,k^{orgp}}\right) - ST\left(indiv_{r,k^{divp}}\right)\right|\right) = 1 \\ \quad sym\left(\left|ST\left(indiv_{i,k^{orgp}}\right) - ST\left(indiv_{r,k^{divp}}\right)\right|\right) = 0 \\ \qquad\qquad \delta \\ 0, sym\left(\left|ST\left(indiv_{i,k^{orgp}}\right) - ST\left(indiv_{r,k^{divp}}\right)\right|\right) = 0 \end{cases} \quad (24)$$

Here, sym is the Symbolic Function which is described in equation 25:

$$Sym(|p-q|) = \begin{cases} 0, m = n \\ 1. m \neq n \end{cases} \quad (25)$$

From Eqn (25), each task in the DivP (i.e., ST ($indiv_{r,k}^{divp}$)) receives a greater fitness value if the server type of the corresponding task is an individual of OrgP (ST ($indiv_{i,k}^{orgp}$)). The div_f is set as 1 if the server types of these tasks are equal. Furthermore, if the two jobs are assigned to the same server (i.e., the same server type and server index), the DivP fitness value for that work is zero.

Selection

The purpose of selection is to choose high-utility individuals from both OrgP and DivP for future iterations based on their respective fitness functions. The individuals of OrgP and DivP are sorted according to their fitness functions, and the top three candidate solutions from each population, as well as one random individual, are selected and saved in the ^{orgp}List and ^{divp}List, respectively.

Crossover and Mutation

The purpose of the crossover stage is to create new individuals (known as offspring) from a group of individuals chosen in the selection step (called parents). The IMA develops two offspring from each pair of selected parents using a two-point crossover operation. The total number of new offspring for each population is computed in each iteration using the equation 26:

$$\text{Offspring Number} = \frac{w!}{(w-p)!}, p = 2 \quad (26)$$

Two crossing points are chosen at random from the parents in the two-point crossover. The genes between the two crossover locations are then exchanged between the parent individuals, leaving the rest unaffected. Because the IMA generally uses two populations, OrgP and DivP, inbreeding occurs when individuals from the same population crossbreed, whereas crossbreeding occurs when individuals from separate populations crossbreed. Individual variety is provided by crossbreeding, which helps to avoid local optimal values with a higher probability. In addition, crossbreeding results are recorded in a selected list of both populations (OrgPList, DivPList), whereas inbreeding results are stored in the selected list of corresponding populations. The mutation function in the APMA affects many genes of progeny based on a predefined probability in the hopes of developing individuals with better utility. In the mutation step, a random resetting mutation is applied, where selected genes are reassigned to a different server type and/or server index with a predefined mutation probability. This ensures diversity in the population and enhances the exploration of the search space to avoid premature convergence."

Local Search Step

Optimization approaches that utilize a population of candidate solutions are known as genetic algorithms (GAs). Parent selection, crossover, mutation, and replacement are the four processes that the population goes through. GAs is commonly thought of as search techniques for locating high-performance areas in large, complex search spaces, but they are not well suited for fine-tuning solutions. The components of GAs, on the other hand, might be custom-designed and their characteristics fine-tuned to enable effective local search behavior. Several models have lately been given with this goal. These algorithms are referred to as Local Genetic Algorithms in this chapter (LGAs). Considering that the crossover locations and mutation genes are chosen at random, a new function called local search is constructed, which is based on the OrgP(F_q^{orgp}(Vn,i))'s local fitness function. The crossover function and mutation provide randomness, which is important since it allows for a higher possibility of jumping out of local optimal locations. Together with those random functions, the local search function accelerates convergence to the global optimal solution.

Algorithm 5: Local Search Step

Input: OrgP List: Selected list of the original population, DivP List: Selected list of diverse population

Set length = |Org P List|

Buff List = set List (MAXINT)

Iterate through each individual (p = 1 to |indiv|):

Iterate through the original population list (q = 1 to length):

If Flop (indivq, porgp) < buff List. Get (p),

update:

- Buff List [p] = Flop (indivq, porgp)
- Org P List. Add (GenerateIndiv (buff List. Get (0)))
- Update Popul (Org P, Org P List)
- Update Popul (Div P, Div P List)

Although the local search function enhances the likelihood of faster convergence to the global optimal solutions, it may cause two issues. First, relying entirely on local search functions increases the likelihood of becoming stuck in local optimal spots. Second, the local search mechanism takes a long time to traverse the search space for an issue with a big solution space. As a result, when implementing a local search function in the IMA, these two considerations should be taken into initializing the buff List with an infinite number of values. Genes with the same index number are evaluated in terms of their local fitness values for individuals in the Org P List. Flop (indivq, porgp) and best genes are chosen and stored in buff List's respective index numbers (lines 3-9). Because the fitness function is based on the execution cost, a lower fitness value indicates a better assignment (line 5). Following that, a new individual is generated and saved in the Org P List (line 10). Finally, the Org P and Div P are combined with the updated Org P List and Div P List from the local search stage and the top candidate solutions from each population are chosen for the populations of the next iteration (lines 11-12). The best individual of the Org P stored in Org P List. Get (0) is returned as the outcome of the IMA whenever it reaches its halting criterion.

Failure Recovery Phase

Failures can occur in any system, so providing an effective failure recovery strategy is critical. Brokers in our system keep track of all the servers and see if they have any plans to undertake a whether the task will be completed soon or not. Furthermore, they estimate the complete cost of each work based on its local fitness value, $M_q$orgp (Vn, i) taking into account the assigned server for each task. If any job fails to execute, choose a surrogate server for that task. As inputs, the failure recovery technique takes a list of currently available free servers (including IoT devices) and a failed job. The local fitness value $M_q$orgp (Vn, i) of that task is then calculated for free servers. Finally, jobs with the least $M_q$orgp (Vn, i) for execution will be forwarded to the server.

Overhead Analysis

The computational steps of the Improved Memetic Algorithm (IMA) and the lightweight pre-scheduling phase determine the time complexity of the suggested application placement technique. The task ordering procedure in the pre-scheduling phase creates the Directed Acyclic Graph (DAG) by a traversal of the Breadth-First Search (BFS). This results in a time complexity of O(V + E), where V is the number of tasks (vertices) and E is the number of dependencies (edges) in the workflows. There are several computational steps in the Improved Memetic Algorithm (IMA). The complexity of the initialization phase, which entails creating a population of size, Psize is O(Psize*T) where T is the number of tasks in a schedule. A complexity of O(I*Psize*T) results from evaluating the fitness function for I iterations. O(I*Psize*T) is the complexity of the crossover and mutation operations, which also depend on Psize and T. Furthermore, a subset of Psize is subjected to the local search refinement step, yielding a complexity of O(L*T) where L is the number of individuals undergoing local refinement. Thus, the overall time complexity of the proposed solution is O(V+E)+O(I*Psize*T). This analysis highlights that the algorithm is scalable for moderate-sized workflows but may incur higher computational costs for larger systems, which should be considered in real-time applications.

Results and Discussion

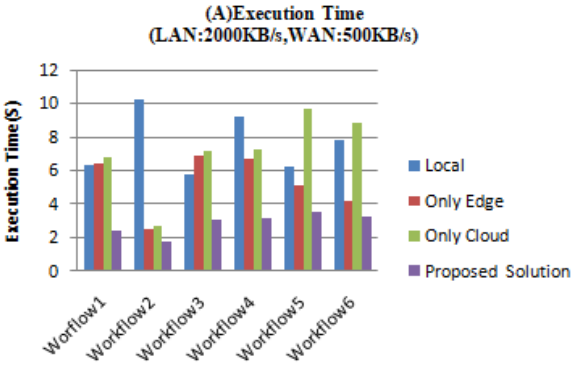
The system setup and parameters, as well as a full performance analysis of our technique in comparison to its competitors, are discussed in this section.

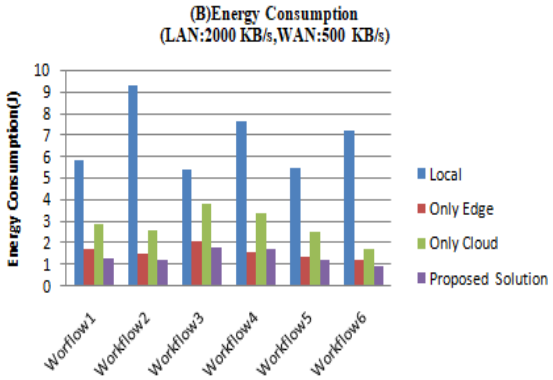
System Setup and Parameters

In this experiment, we used the iFogsim simulator to assess all approaches. Real workflows and synthetic workflows were used here. The DAGs that were obtained from the facial recognition app (Workflow1) and the QR code recognition application were used to create the real workflows (Workflow2). In contrast, several synthetic workflows are designed to examine various possible kinds of workflows (Workflow3 to Workflow6). Consider a case in which there are six IoT devices, each with its own workflow (Workflow1 through Workflow6). One fog broker connects each group of six IoT devices, and fog brokers have a connection to six fog servers and three cloud servers. Every fog server has three virtual machines (VMs) in this design, while each cloud server has 16 virtual machines (VMs). IoT devices also have the computing power of 500 MIPS [15], with a power consumption of 0.9W and 0.3W in processing and idle levels, respectively. Additionally, IoT devices need 1.3 watts of transmission power. We also assume that each VM of fog servers has to have a computer power of 6 to 8 times that of IoT devices [15], whereas each VM of cloud servers has a computing power of 10 to 12 times that of IoT devices. Table 2 displays the parameters of our evaluation and their corresponding values.

Table (2): System Parameters.

Parameter	Value
IoT Devices	6
Fog Servers	6
Cloud Servers	3
LAN Bandwidth	2000–4000 KB/s
WAN Bandwidth	500–1000 KB/s
LAN Delay	0.5 ms
WAN Delay	30 ms
IoT Device Computing Power	500 MIPS
Fog Server Speedup Factor	6–8
Cloud Server Speedup Factor	10–12
IoT Device Idle Power	0.3 W
IoT Device CPU Power	0.9 W
IoT Device Transmission Power	1.3 W





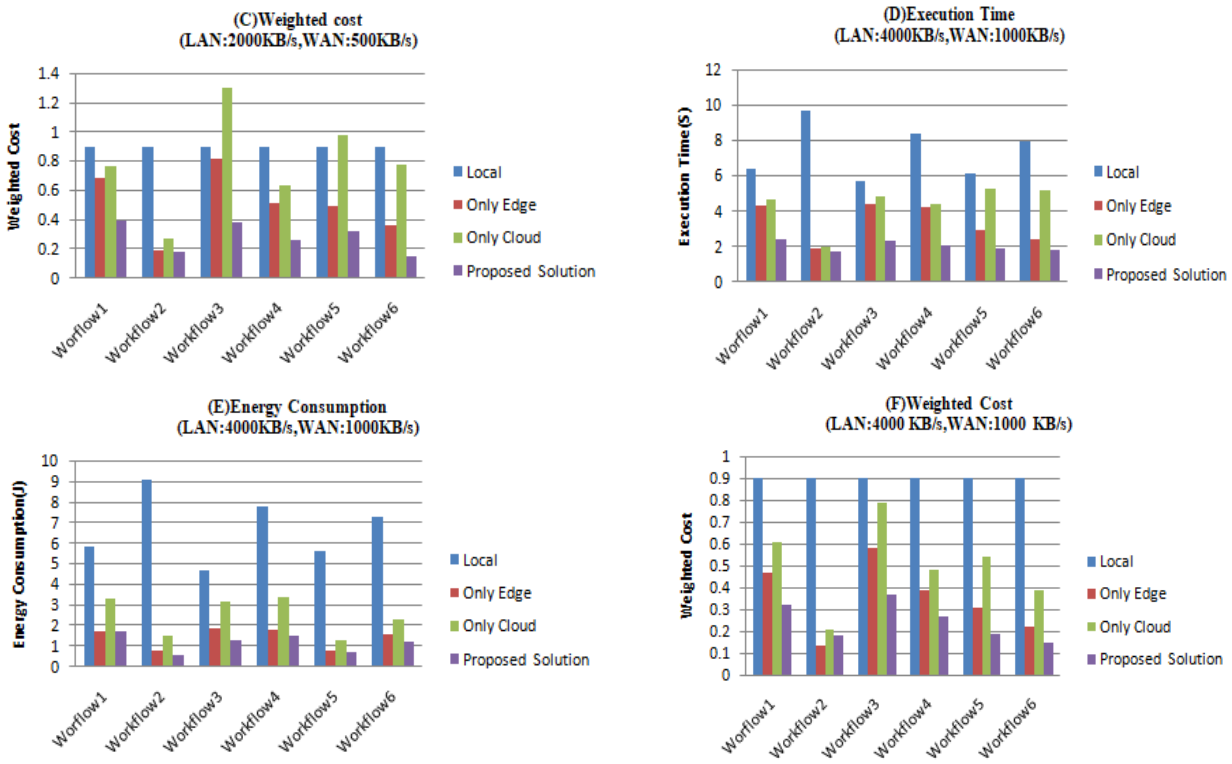


Figure (2): Execution cost, Energy Consumption, Weighted Cost with various workflows on varying bandwidth values.

Performance Study

To analyze the performance of our technique in various experiments, we evaluated three quantitative parameters: execution time, energy consumption, and weighted cost. The effectiveness of techniques with varying bandwidths, iteration sizes, decision durations, failure recovery, and system size analyses is evaluated in 5 trials. Both $\Psi\gamma$ and Ψ_θ are set to 0.5, indicating the importance of processing time and energy use in the conclusions. These values, however, can be changed depending on the needs of the users and network conditions. For evaluations, the methods are also used to evaluate the ability of our technique:

LOCAL: Because all workflow tasks in this technique are completed locally on their respective IoT devices, workflow tasks cannot be executed in tandem. This method's outcomes can be used to evaluate the technique's gain.

NLY EDGE: All workflow tasks are outsourced to fog servers at the edge layer for implementation in this approach. If all of the virtual servers on a server are full, no more virtual servers are available; the remaining jobs will have to wait until more computing resources become available.

ONLY CLOUD: All workflow tasks are to be carried out on cloud servers by using the method.

Bandwidth Analysis

In this analysis, we look at how approaches perform at different bandwidths, as shown in Figure 4 and Pop Size are set to 100 and 20, respectively, for maximum iteration size and population size. In comparison to the local execution of workflows, Figure 3 demonstrates that when bandwidth grows, process execution time, energy usage, and weighted cost decrease, implying improved application placement. Furthermore, because fog servers are located near IoT devices and can be accessed with faster bandwidth and low bandwidth, the edge technique surpasses the only cloud option in most circumstances. However, as fog servers have fewer resources than cloud servers, they are unable to provide the best results. As can be demonstrated, our proposed technique outperforms all others due to two main aspects: it involves simultaneous usage of fog and cloud servers. Second, due to its local fitness function, local search, and the variation provided by the Div P, it keeps a higher chance of staying away from local optimal values, grows faster to the optimal solution, and has a higher probability of remaining away from local optimal values. It's important to note that while the weighted cost of the only cloud method in some scenarios, such as Workflow5 in Fig. 2c, is lower than the local execution, its execution time in Fig. 2a is much greater. Because the $\Psi\gamma$ and Ψ_θ are both set to 0.5, execution time and energy usage are given equal weight. As a result, the weighted cost indicates a low gain for task placement due to the lower value for energy consumption in this workflow when compared to the obtained execution time.

Figure 2 illustrates the impact of varying bandwidth values on the execution cost of IoT workflows using different application placement strategies. It compares the proposed method with baseline approaches (Local, Only Edge, and Only Cloud), demonstrating how increased bandwidth improves application placement efficiency. The figure highlights that the proposed method consistently outperforms others due to its effective use of both fog and cloud resources, as well as its capability to avoid local optima through the Improved Memetic Algorithm (IMA).

Maximum Iteration Number Analysis

The maximum iteration number, which may be used to assess the speed with which evolutionary application placement approaches reach the ideal answer, is one of the most essential factors to compare. The solutions of the local implementation, edge, and only cloud approaches do not change over time; the acquired outcomes of these methods are shown to help understand the efficacy of other strategies. For this investigation, the Pop Size, LAN, and WAN bandwidths are set to 20, 2000, and 500 KB/s, respectively. As shown in Fig. 3, increasing the maximum number of iterations improves our technique's solutions for all workflows when compared to the local, only edge, and only cloud techniques.

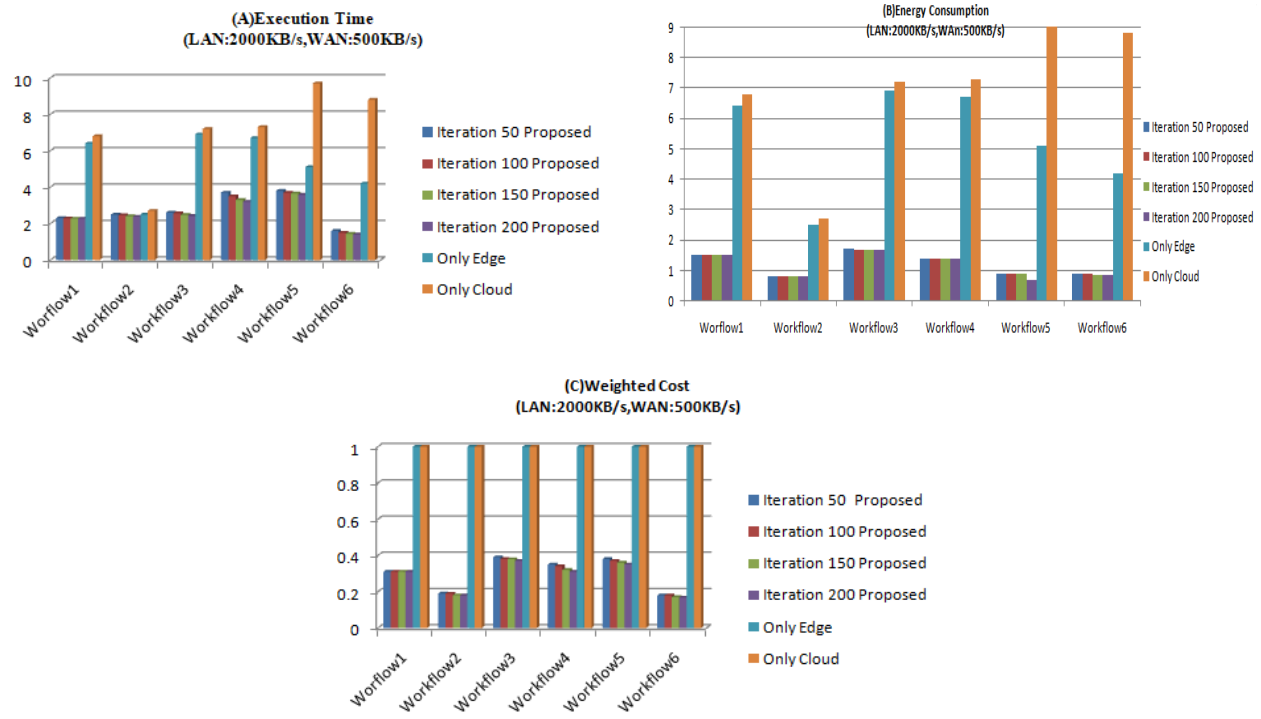


Figure (3): Execution cost of workflows with different maximum iteration numbers of values.

Our method converges to a superior solution in a less number of iterations. The achieved results of our approach in I= 50 for all processes surpass the results obtained even at I = 200, as shown in Fig. 3a. This pattern is shown in Fig. 3b for the weighted cost of execution. It is crucial to remember that while extending the maximum number of iterations can lead to better solutions, it also increases the decision time of algorithms, this is important for some workflows, especially those that are latency-sensitive.

While increasing bandwidth and the number of iterations demonstrably improve solution quality, these findings have practical limitations. Higher bandwidth may lead to diminishing returns, and achieving such conditions may be cost-prohibitive or infeasible in certain environments. Similarly, increasing the number of iterations comes with trade-offs in terms of decision time and resource consumption, particularly for latency-sensitive or energy-constrained applications. Future work should explore adaptive strategies to optimize bandwidth usage and dynamically determine the ideal number of iterations based on system constraints.

The effectiveness of each strategy is evaluated in this experiment based on the amount of time it takes to get a suitable solution. Whereas application placement algorithms provide server configurations that minimize IoT application execution time and energy usage, the time spent getting to that solution is also crucial.

Establishing excellent server configurations for IoT applications over a long period of time might have a detrimental impact on the IoT applications' execution time needs. Another significant reason for the importance of decision time analysis, particularly for optimization algorithms, is that iteration size analysis alone cannot determine the efficiency of a single application placement approach. This is because, while one method may produce greater outcomes in fewer iterations than its rivals, another strategy may need more iterations. The duration of each loop may be significantly greater, resulting in a longer decision time. As an outcome, whereas the maximum iteration size analysis is critical, the decision time analysis is used as a backup to ensure that one approach is effective. The LAN and WAN bandwidth consumption in this experiment is 2000 KB/s and 500 KB/s, correspondingly. For four distinct decision times,

Table 3 provides the COM2019 execution times for the recommended solution. As the decision time of approaches climbs from 100 milliseconds to 400 milliseconds, the execution time of approaches decreases, meaning that higher utility outcomes are attained. Our solution's obtained results gradually decrease from 100 to 400 milliseconds, whereas COM2019's study results show a significant decreasing trend in the variations of 100-200 milliseconds and 200-300 milliseconds, as well as a steady decrease between 300 and 400 milliseconds, intimating that COM2019's results modeled convergence at 400 milliseconds. It is obvious that our approach not only produces superior values in the same decision time as the COM2019 but that its results at 100 ms also beat the COM2019's findings at 400 ms. This shows that irrespective of the number of iterations, our method converges to the best solutions faster.

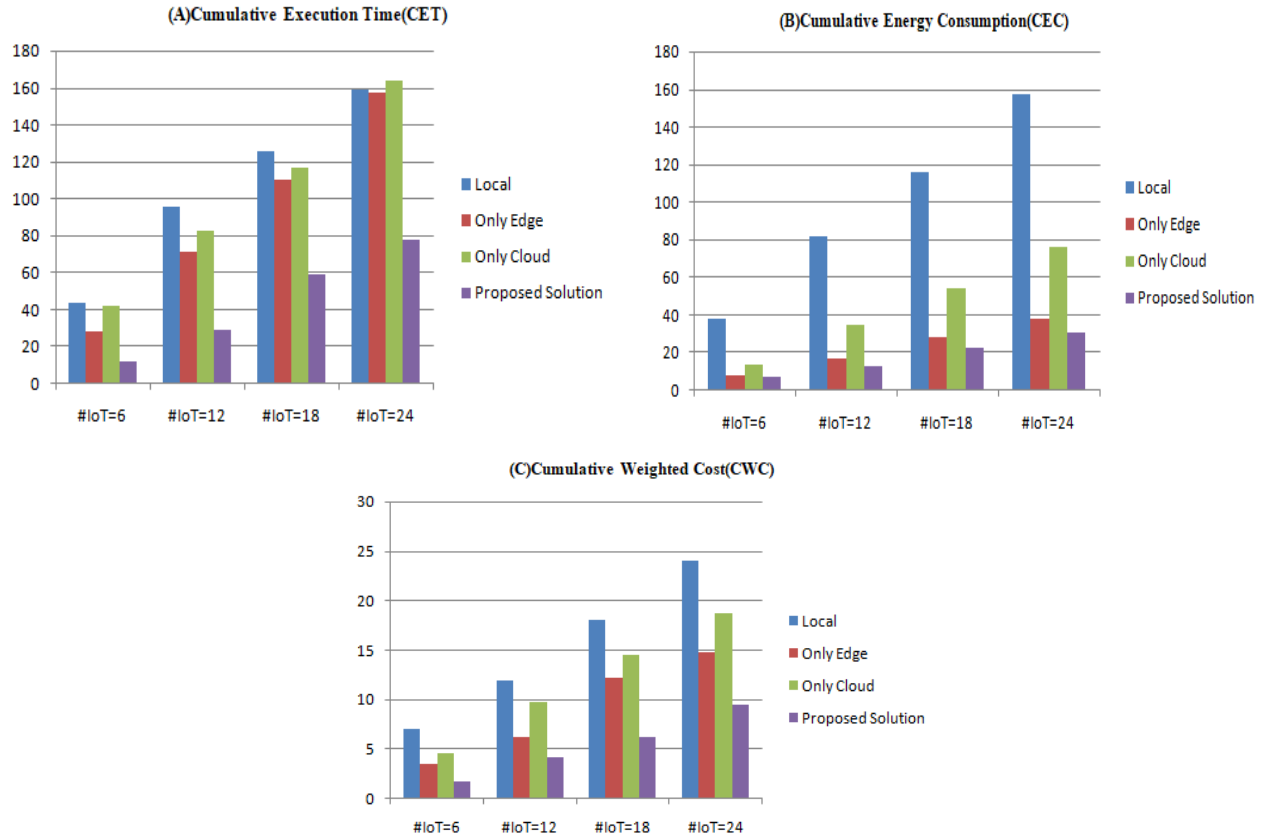


Figure (4): System size analysis with different number of IoT devices.

Table (3): Decision time Analysis.

Time	Technique	Workflow Execution Time Result (s)					
		WF1	WF2	WF3	WF4	WF5	WF6
100 ms	Proposed	2.4	1.8	3.1	3.2	3.5	3.3
	COM2019	4.333	2.917	3.422	6.276	6.526	3.09
200 ms	Proposed	2.37	1.76	2.92	3.17	3.42	3.27
	COM2019	4.073	2.707	2.984	5.344	5.109	2.529
300 ms	Proposed	2.31	1.72	2.85	3.12	3.38	3.18
	COM2019	3.656	2.494	2.868	4.388	4.709	2.746
400 ms	Proposed	2.27	1.65	2.79	2.73	3.27	3.12
	COM2019	3.623	2.445	2.753	3.663	4.295	2.523

This is mostly due to the fact that time is measured in milliseconds (ms), representing the time allocated for the optimization algorithm. Workflow execution time results are measured in seconds (s) and represent the total time taken to execute the workflows under the determined placement strategy.

Failure Recovery Analysis

The influence of failure recovery methods on application placement approaches is investigated in this experiment. In comparison to local execution, Table 3 shows the outcomes of our method in failure recovery mode (FR Mode), where the probability of failure is 5%. In this experiment, the maximum number of iterations size I is set to 100, while the rest of the parameters are left at their default settings from decision time analysis. Table 3 shows that the results of our method in FR mode continue to beat local execution for all processes and generate offloading gains. Failed tasks result in inadequate workflow execution due to interdependence among tasks in one workflow in strategies that ignore failure recovery. However, by tolerating a tiny overhead of the failure recovery phase, our technique can yield a reasonable gain over local execution.

Table (4): Failure Recovery Analysis.

Technique	**Workflow Execution Time Results**					
	WF1	**WF2**	**WF3**	**WF4**	**WF5**	**WF6**
Proposed	2.871	2.732	2.885	3.511	3.66	1.51
Local	6.3	10.2	5.8	9.2	6.2	7.8

System Size Analysis

The impact of system size on various application placement approaches is investigated in this system experiment. Each fog broker in our system decides where each IoT device's application should be installed. As a result, we increase the number of IoT devices and fog servers for each fog broker by a factor of six, from six to twenty-four for each fog broker. This allows us to evaluate the performance of our suggested technique. In addition, we use identical procedures in this experiment as we did in the prior ones. The LAN and WAN bandwidth utilization are also set to 2000 and 500 KB/s, correspondingly, with the remainder of the parameters remaining unchanged from Table 4. When varying numbers of IoT devices are linked to a single fog broker, the result of Cumulative Execution Time (CET), Cumulative Energy Consumption (CEC), and Cumulative Weighted Cost (CWC) is shown in Fig. 6. The overall execution cost of all IoT devices is referred to as cumulative. As the number of IoT devices rises, the CET, CEC, and CWC rise as well. All approaches CET, CEC, and CWC are cheaper than of the local execution cost in all cases; nonetheless, our suggested strategy beats other ways and saves money in all scenarios.

Conclusion

In this study, we proposed a weighted cost model and a novel batch application placement technique using the Improved Memetic Algorithm (IMA) to minimize the execution time and energy consumption of IoT applications in fog-cloud environments. The proposed technique also incorporates a lightweight pre-scheduling approach and an efficient failure recovery mechanism to handle runtime issues effectively. Experimental evaluations demonstrate that our method outperforms state-of-the-art approaches in various scenarios: In Bandwidth Analysis the proposed system achieved a 65% reduction in weighted cost compared to baseline techniques

such as Local Execution, Only Edge, and Only Cloud methods. In Decision Time Analysis the system attains improved execution time by 51% compared to COM2019 at equal decision time, with results converging faster even at lower decision times (e.g., 100 ms). In failure recovery mode, the proposed method consistently delivered execution time gains of up to 53% compared to local execution, highlighting its robustness. System Size Analysis demonstrated superior scalability, with Cumulative Execution Time (CET) and Cumulative Weighted Cost (CWC) reductions across all scenarios, even as the number of IoT devices increased. These results confirm the effectiveness and scalability of the proposed technique, offering significant performance improvements over existing state-of-the-art methods in terms of execution cost, energy consumption, and weighted cost metrics. Future work will explore the integration of monetary cost considerations into the weighted cost model and address challenges associated with mobility models in dynamic fog environments.

Disclosure Statements

- **Ethics approval and consent to participate:** This article doesn't need any ethical approval.
- **Data Availability:** The dataset used is publicly available
- **Authors Contribution:** N. Malathy -Methodology, Draft revision and editing, Verification, Validation and supervision. Ruba, Vinothini - Methodology, Implementation, Draft preparation, Verification, Validation
- **Funding:** This work doesn' t receives any fund.
- **Conflict of Interest:** The authors declare that they have no conflict of interest.
- **Acknowledgements:** The authors would like to thank Mepco schlenk engineering college for providing environment to carry out this research work.

Open Access

References

1] Goudarzi, M., & Wu, H. (2021). An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments. *IEEE Transactions on Mobile Computing*, 20(4), 1298–1311. https://doi.org/10.1109/TMC.2020.2967041

2] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, "Multiserver multiuser multitask computation offloading for mobile edge computing networks," Sensors, vol. 19, no. 6, 2020, Art. no. 1446.

3] D. G. Roy, D. De, A. Mukherjee, and R. Buyya, "Application-aware cloudlet selection for computation offloading in the multi-cloudlet environment," J. Supercomputing, vol. 73, no. 4, pp. 1672–1690, 2020.

4] E. El Haber, T. M. Nguyen, D. Ebrahimi, and C. Assi, "Computational cost and energy efficient task offloading in hierarchical edge-clouds," in Proc. 29th IEEE Int. Symp. Personal Indoor Mobile Radio Commun., 2022, pp. 1–6.

5] Y. Nan, W. Li, W. Bao, F. C. Delicato, P. F. Pires, and A. Y. Zomaya, "A dynamic tradeoff data processing framework for delaysensitive applications in cloud of things systems," J. Parallel Distrib. Comput., vol. 112, pp. 53–66, 2020.

6] Malathy N, Grace Sophia J, Swathi S, Vijaya-Subasri K. Privacy-preserving medical diagnosis system using harris hawk optimization in edge computing. Int Res J Multidisciplin Scope. 2024;5(1):157–70

7] Malathy Navaneetha Krishnan, Revathi Thiyagarajan. Multi-objective task scheduling in fog computing using improved gaining sharing knowledge-based algorithm. Concurrency and Computation: Practice and Experience.2022;34:1- 22.

8] Malathy Navaneetha Krishnan, Revathi Thiyagarajan. Entropy-based complex proportional assessment for efficient task scheduling in fog computing. Transactions on Emerging Telecommunications Technologies.2023;23:2.

9] Malathy Navaneetha Krishnan, Revathi Thiyagarajan. Opposition-based Improved Memetic Algorithm for Placement of Concurrent IoT Applications in Fog Computing. Transactions on Emerging Telecommunications Technologies.2024

10] H. K. Apat, P. Sattarapu, R. N. Dash, V. Goswami, S. Mohanty and R. K. Barik, "Leveraging towards Multi-objective IoT Application Placement in Fog Computing Environment," *2023 IEEE 3rd International Conference on Smart Technologies for Power, Energy and Control (STPEC)*, Bhubaneswar, India, 2023, pp. 1-6, doi: 10.1109/STPEC59253.2023.10430628.

11] Hemant Kumar Apat, Bibhudutta Sahoo, Veena Goswami, Rabindra K. Barik,A hybrid meta-heuristic algorithm for multi-objective IoT service placement in fog computing environments,Decision Analytics Journal,Volume 10,2024,100379,ISSN 2772-6622,https://doi.org/10.1016/j.dajour.2023.100379.

12] Hemant Kumar Apat, Bibhudatta Sahoo,A Blockchain assisted fog computing for secure distributed storage system for IoT Applications, Journal of Industrial Information Integration,Volume 42,2024,100739, ISSN 2452-414X,https://doi.org/10.1016/j.jii.2024.100739.

13] Mohammad Goudarzi , Huaming Wu , Marimuthu Palaniswami , and Rajkumar Buyya , "An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments", IEEE Transactions on Mobile Computing, Vol. 20, No. 4, April 2021.

14] Maria Diamanti, Panagiotis Charatsaris,Eirini Eleni Tsiropoulou and Symeon Papavassiliou," Incentive Mechanism and Resource Allocation for Edge-Fog Networks Driven by Multi-Dimensional Contract and Game Theories",vol no: 3, 10.1109/OJCOMS.2022.3154536,2022.

15] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," IEEE Trans. Evol. Comput., vol. 15, no. 5, pp. 591–607, Oct. 2011

16] Isaac Lera, Carlos Guerrero, and Carlos Juiz," Analysing the Applicability of a Multi-Criteria Decision Method in Fog Computing Placement Problem", IEEE International Conference on Fog and Mobile Edge Computing (FMEC 2019).

17] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," ACM SIGMETRICS Perform. Eval. Rev., vol. 40, no. 4, pp. 23–32, 2013.

18] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" Computer, vol. 43, no. 4, pp. 51–56, 201