



Available online at www.qu.edu.iq/journalcm

JOURNAL OF AL-QADISIYAH FOR COMPUTER SCIENCE AND MATHEMATICS

ISSN:2521-3504(online) ISSN:2074-0204(print)



A Novel Approach for Optimal Service Placement in Fog-Based Internet of Things Using the Cappuccino Optimization

Zahraa Sabah Ali

Department of Computer Engineering, College of Engineering Islamic Azad university, Isfahan ,Iran. zahra26sep1989@gmail.com

ARTICLE INFO

Article history:

Received: 22 /04/2026

Revised form: 14 /05/2026

Accepted : 17 /05/2026

Available online: 30 /09/2026

Keywords:

Fog Computing, Internet of Things,
Task Scheduling, Metaheuristic
Algorithm, Cappuccino
Optimization, Resource Allocation.

ABSTRACT

With the increasing expansion of the Internet of Things (IoT) and the increasing volume of data generated by devices, the need for fast and efficient processing is increasingly felt. In this regard, fog computing has been proposed as an intermediary approach between cloud computing and edge devices, which aims to reduce latency, improve efficiency, and better resource management. Despite these advantages, challenges such as suboptimal resource allocation, high energy consumption, increased response time, and processing load imbalance are still considered major obstacles in this field. In order to overcome these problems, this research presents a novel approach based on the Cappuccino optimization algorithm that focuses on improving task scheduling and location of IoT services in the fog environment. The proposed algorithm is designed as a meta-heuristic method and is able to improve the quality of service (QoS) by optimizing resource allocation, reducing energy consumption, and balancing the load between nodes. To evaluate the performance, the algorithm was simulated in three test scenarios with different amounts of physical machines and tasks. The results showed that the proposed method has made improvements compared to reference algorithms such as CAWPQ and CAWTPQ; in one scenario, the scheduling delay was reduced by 12.5%, in another test, the energy consumption was improved by about 18%, and in the third case, the load balancing was increased by 23%. The findings indicate that the use of the Cappuccino algorithm in fog computing can be an effective step in promoting distributed processing, reducing task execution time, and optimizing the location of IoT services. However, the evaluation is based on simulations, and future work should test the approach in real fog computing environments. These achievements draw a positive outlook for future research in the field of fog computing and the Internet of Things.

MSC..

<https://doi.org/10.29304/jqcm.2026.18.32834>

1. Introduction

With the rapid growth of new technologies and the increasing use of the Internet of Things (IoT), a huge amount of data is being generated by smart devices and sensors. This data is considered a valuable resource for analysis, decision-making, and provision of smart services in areas such as smart cities, precision agriculture, healthcare, smart transportation, and manufacturing industries. However, processing and storing this vast amount of data using traditional computing models faces serious limitations [1, 2]. Transferring all data to cloud data centers, in addition to imposing high economic costs, leads to increased response latency, network bandwidth saturation, and reduced data security. These challenges are especially important in time-sensitive applications such as real-time traffic control, telesurgery, or emergency medical systems, where even a few milliseconds of delay can have irreversible consequences [3].

To overcome these limitations, the concept of fog computing was proposed as an intermediary layer between cloud computing and edge devices. By processing data as close to the source of production as possible, fog computing not only reduces response time and improves quality of service (QoS), but also minimizes bandwidth consumption and communication costs. Despite these advantages, effective exploitation of fog computing requires solving complex problems in task scheduling and resource allocation [4].

Scheduling in distributed systems is a critical and challenging process that determines which computing resource each IoT task or service will run on. This process directly affects indicators such as system efficiency, latency, energy consumption, and processing load balancing. Due to the heterogeneity of resources in distributed environments and the inherent limitations of fog nodes, task scheduling is known as an NP-Hard problem. This means that finding an exact and optimal solution for it on a large scale is practically impossible and requires the use of metaheuristic and metaheuristic methods. These methods provide the possibility of achieving near-optimal solutions in an acceptable time by searching the solution space approximately [5, 6]. Despite the development of various methods, most existing metaheuristic algorithms face limitations such as falling into local optima, slow convergence, and lack of dynamics in the face of environmental changes. For example, many of the proposed methods in previous research have focused solely on reducing response time or energy consumption, and have neglected the issue of processing load balancing and optimal resource allocation. Also, some algorithms lack sufficient flexibility to adapt to the large scalability of IoT environments. These limitations indicate that there are still significant gaps in the field of task scheduling and service localization in fog environments [7, 8].

In recent years, combining novel meta-heuristic approaches with scheduling problems has been considered as one of the attractive research areas. Among these approaches, we can mention nature-inspired algorithms such as genetic algorithm (GA), particle swarm optimization (PSO), ant colony algorithm (ACO), and algorithms based on bird or bat swarms. Each of these methods, despite their valuable achievements, still suffers from weaknesses in some key performance metrics [9]. Therefore, there is a strong need for more efficient algorithms that can simultaneously optimize multiple criteria, including energy consumption reduction, latency reduction, and load balancing improvement [10].

Although numerous metaheuristic approaches have been applied to fog task scheduling, most focus on single-objective optimization and overlook the integrated problem of service placement and load balancing. Additionally, limitations such as local optima stagnation and scalability issues remain insufficiently addressed, highlighting the need for a more adaptive multi-objective optimization framework. In this study, a novel approach based on the Cappuccino Optimization (CSA) is proposed to address this need. Inspired by the search behavior of Cappuccino monkeys, this algorithm has a high ability to avoid local optima and increase convergence speed. By considering the priority of tasks, their execution time, and the resource constraints in the fog environment, the proposed method is able to optimize the location of IoT services and allocate resources in a way that reduces energy consumption and maintains load balance. The distinctive feature of this algorithm is the use of directed random search, which enables the discovery of new and effective solutions. The main innovation of this research lies in the combination of service location and static task scheduling using the Cappuccino algorithm. Unlike many previous approaches that have focused on only one aspect of the problem, the present method attempts to simultaneously reduce latency, optimize energy consumption, and increase service quality. Initial results have also shown that this algorithm performs better than reference methods such as CAWPQ and CAWTPQ. The contribution of proposed method in brief are as follow

- A CSA-based method for service placement and task scheduling in fog computing environments.

- Simultaneous improvement of key QoS metrics, including latency, energy consumption, and load balancing.
- Better search performance using CSA, helping the system avoid local optima and allocate resources more efficiently.

In the rest of the article, the research structure is organized as follows: In the second section, the theoretical foundations and related studies are reviewed. The third section is dedicated to describing the proposed method and the details of the Cappuccino algorithm. In the fourth section, the results of simulation experiments and comparison with existing methods will be presented. Finally, in the fifth section, conclusions and suggestions for future research are presented.

2. Related works

Task Scheduling in Cloud Computing and Fog Computing environments is one of the fundamental and complex issues in Distributed Systems. In this process, user tasks are divided into sub-tasks and then allocated to computing resources such as physical machines and virtual machines based on scheduling policies. The main goal of scheduling is to reduce execution time, increase system throughput, and create load balancing among resources [11-13]. In general, the scheduling process includes three stages: Resource Discovery & Monitoring, Resource Selection, and Task Dispatching, in which the system first identifies the status of resources, then selects the best resource, and finally assigns tasks to it [36]. Due to the heterogeneity of resources and the limitations of fog computing, criteria such as Makespan, Cost, Deadline, Energy Efficiency, Load Balancing, and Quality of Service (QoS) are considered to evaluate the performance of scheduling algorithms.

Scheduling methods can be divided into three general groups: Traditional Scheduling Methods such as First Come First Served (FCFS), Shortest Job First (SJF), and Round Robin (RR), which are simple and fast but inefficient in complex cloud problems[14][15, 16]; Heuristic methods such as Min-Min and Max-Min, which provide approximate solutions but often stop at local optima; and meta-heuristic methods inspired by the nature and social behavior of living organisms, such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Cappuccino Optimization (CSA), which have a higher ability to escape local optima and search a wider space [17-19]. From a classification perspective, scheduling can be static or dynamic, centralized or distributed, and even preemptive or non-preemptive, each with its own advantages and limitations. Given that task scheduling is a Non-deterministic Polynomial-time hard (NP-Hard) problem [20], the development of novel meta-heuristic algorithms for Reduced Execution Time, Optimized Energy Consumption, Improved QoS, and Enhanced Load Balancing in fog and cloud computing has become a research necessity.

In [21], they presented a service placement optimization method to reduce power consumption in IoT environments as a minimum mixed-cost flow problem. In [22], they introduced the concept of fog colonies for hierarchical optimization. Each colony uses a GA to decide which services to place in the colony and which of these services are also transferred to neighboring colonies. In [23], they presented a tool called Fog Torch in two research papers. The first paper presents a model for using QoS systems in multi-sector IoT applications in fog architecture. The second paper presents the results of using Monte Carlo simulations in Fog Torch tool and categorizes the application of this tool in terms of QoS and resource consumption. In [24], they presented a new method for task scheduling in the IoT context in an energy-based fog environment. This paper investigates the migration of services from cloud to fog using mathematical and intuitive modeling, considering factors such as traffic, workload, and proximity of sensors. The results show that energy consumption is reduced by 56% and CPU utilization by 46%, but computational complexity hinders the improvement of migration speed. In [25], they present a comprehensive taxonomy for the scheduling problem in distributed systems, especially in cloud computing. This taxonomy is introduced by analyzing existing research as a tool to better understand scheduling approaches. They consider scheduling as a decision-making process for allocating resources among activities, which has faced new challenges with the emergence of distributed systems. In [26], the MAE3D method is a new automatic encoder for self-supervised learning on 3D point clouds, which first divides the point cloud into patches and masks some of them. Using patch-based transformers, local features and high-level relationships between patches are learned and the masked parts are reconstructed. The results show that MAE3D outperforms existing methods in downstream classification tasks (with accuracies of 93.4% and 86.2%). In [27], a novel multi-objective hybrid Cappuccino search with genetic algorithm (MHCSGA) was proposed to improve the scheduling parameters. The MHCSGA method optimizes multi-objective functions such as resource utilization, response time, build time, execution time, and throughput. In [28], a Proactive Hybrid Pod Autoscaling (ProHPA) method was proposed to improve the task

scheduling using a bidirectional long short-term memory (Bi-LSTM) model and an attention mechanism. This method predicts future CPU and memory usage and prevents over-allocation of resources using (ReVPA) resource usage with vertical pod autoscaling. Finally, ProHPA performs resource scaling instantly by utilizing predictions and pod information. In [29], they presented a hybrid algorithm based on simulated annealing and tabu search for task scheduling in cloud computing. This algorithm performed better than the whale and gray wolf algorithms in experiments, and its completion time decreased with the increase of virtual machines. Also, the proposed algorithm showed high efficiency in solving complex and large-scale problems.

As summarized in Table 1, most existing studies focus on either service placement or task scheduling individually, and many of them optimize only limited performance metrics such as energy consumption or response time. However, an integrated approach that simultaneously considers service placement, task scheduling, and load balancing in fog environments is still limited.

Table 1 - Comparative Summary of Related Studies

Ref	Method Type	Service Placement	Task Scheduling	Multi-Objective	Energy Optimization	Load Balancing	Limitation Identified
[21]	Optimization (Flow-based)	✓	✗	✗	✓	✗	Focus only on power consumption
[22]	GA-based	✓	✓	✓	✓	✗	Colony-based, scalability concerns
[23]	Simulation Tool (FogTorch)	✓	✗	✗	✓	✗	Mainly modeling & evaluation
[24]	Mathematical Modeling	✓	✗	✗	✓	✗	High computational complexity
[25]	Taxonomy Study	✗	✓	✗	✗	✗	Conceptual, no optimization model
[27]	Hybrid CSA + GA	✗	✓	✓	✓	✓	Focused on scheduling only
[28]	ML-based (Bi-LSTM)	✗	✓	✓	✓	✗	Resource prediction-focused
[29]	SA + Tabu Search	✗	✓	✗	✗	✗	Cloud-only, limited objectives

3. Proposed Method

The architecture of the proposed system is illustrated in Figure 1. In this framework, a large number of IoT devices continuously generate tasks that need to be processed with minimal delay. These tasks are transmitted to the fog layer, which acts as an intermediate computing layer between IoT devices and the cloud. The fog nodes are responsible for handling time sensitive tasks and providing faster responses compared to cloud processing. To efficiently allocate tasks among available fog resources, the proposed Cappuccino Optimization (CapSA) is employed as the scheduling and optimization mechanism. The optimizer evaluates different task to node assignments and selects the configuration that minimizes execution delay and resource consumption while maintaining load balance among fog nodes. If the available fog resources are insufficient, tasks can be forwarded to the cloud layer for further processing. This hierarchical architecture enables efficient service placement, reduces latency, and improves the overall performance of the IoT fog computing environment.

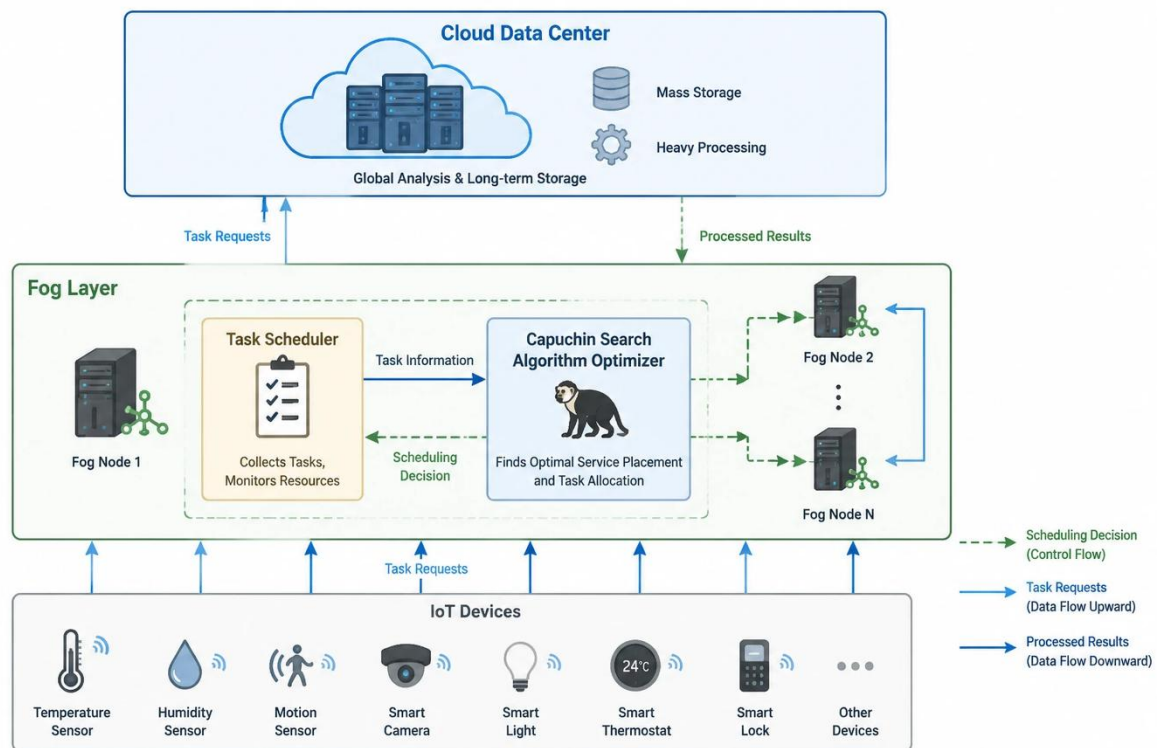


Fig. 1 - Proposed approach in applying Cappuccino optimization algorithm in service location

3.1. Data property

This research presents several innovations by applying Cappuccino metaheuristic algorithm for task scheduling in fog computing, including distributed resource optimization, flexibility in dynamic conditions, energy consumption reduction, improved scheduling considering latency and bandwidth, and adaptability to changing IoT needs. The assumptions of the method, including migration of heavily loaded machines in priority, load balancing, and energy consumption reduction, are given in Table 2. In this model, the limitations of IoT devices such as processing power, memory and energy, user mobility, equipment diversity, and different sensor network topologies, WiFi, Bluetooth, ZigBee, are considered. This information is used to optimize resource allocation and efficiently manage distributed and heterogeneous systems in fog environments, with the aim of improving efficiency, reducing latency, and increasing the overall system productivity. In this way, IoT users will be clustered into four priority user clusters: very urgent, urgent, almost urgent, and normal. Users with very urgent priority, such as military or healthcare users, will be allowed to utilize and use cloud resources with higher priority [30].

Table 2. Information Collected from IoT Users

No.	Type of Information Collected
1	Computational, memory, and energy consumption limitations
2	Mobility of IoT users
3	Variety of IoT user devices and equipment
4	IoT user topology (sensor network, WiFi, etc.)

In the first step, the required data is collected from the edge computing environment so that the load balancer can check the status of resources and tasks for scheduling. This information includes the operating frequency of physical hosts, the volume of processed data, the energy consumption of hosts and IoT tasks, and the number of active and running tasks, which are detailed in Table 3. Then, based on this data and with the help of the Cappuccino meta-heuristic algorithm, the migration priority of virtual machines is determined and resource allocation is optimized to reduce energy consumption and increase efficiency.

Table 3. Information Collected from Virtual and Physical Machines in Edge Computing

No.	Type of Information Collected
1	Operating frequency of the physical host
2	Volume of data being processed by the physical host
3	Energy consumed by the physical machine
4	Energy consumed by the requested IoT task
5	Number of IoT requested tasks active on the physical host
6	Number of tasks running in the requested IoT task
7	Number of tasks running on the physical host

3.2. Optimization

Real-world optimization problems require efficient methods due to their high complexity. Deterministic algorithms such as gradient descent, although accurate, are trapped in local optima, while metaheuristic algorithms with a stochastic nature and black-box model are capable of achieving responses close to the global optimum. Among them, bioinspired algorithms have found a special place. One of these methods is the Cappuccino algorithm, which is used with a metaheuristic and flexible approach to solve complex problems such as scheduling and resource allocation in fog computing and IoT, and helps in load balancing and resource optimization by creating diversity in solutions and continuous improvement [31-33]. The CapSA algorithm was designed based on the social behavior of Cappuccino monkeys in search of food and is used to solve global and local optimization problems. Like other crowd-based algorithms, this method is a population-based algorithm that starts with a random initialization of a population of Cappuccinos, where each Cappuccino represents a candidate solution to the problem (here, task scheduling). This initial population is represented in the d -dimensional search space using a matrix x , as given in equation (1).

$$\begin{bmatrix} x_1^1 & \dots & x_d^1 \\ \vdots & \dots & \vdots \\ x_1^n & \dots & x_d^n \end{bmatrix} \quad (1)$$

where x represents the position of the Cappuccinos, d is the number of problem variables, n is the number of Cappuccinos, and x_d^i represents the d th dimension after the position of the i th Cappuccino. Equation 2 is used to assign the initial position of each Cappuccino in the population.

$$x^{ij} = ub_{ij} + r \times (ub_{ij} - lb_{ij}) \quad (2)$$

The above relation where ub_{ij} and lb_{ij} are the lower and upper bounds for the i -th Cappuccino in the j -th dimension, respectively, and r is a random number generated uniformly in the range of 0 to 1. x_i is the new position of the Cappuccino. The index determines the new position and the index J determines the current position. The generation of new populations (new solutions) in CapSA relies on the current position of the Cappuccino and the positions of the best Cappuccinos, as well as the food source, called F . The food source, F , is the target of the Cappuccinos in the d -dimensional search domain, which must be iteratively updated. The following relation is used to generate a new solution in the proposed algorithm:

$$x_j^i = F_j + \frac{P_{bf}(v_j^i)^2 \sin(2\theta)}{g} \quad (3)$$

The parameters used in Equation 3 are as follows: x_j^i specifies the position of the alpha Cappuccino and other follower Cappuccinos in dimension j , F_j is the position of the food in dimension j , P_{bf} is the equilibrium probability created by the Cappuccino tails during the leaping movement, g is the gravitational force, which is equal to 9.81, θ is the angle of the Cappuccino jump, τ is a lifetime parameter that systematically decreases in all iterations, v_j^i is the velocity of the i th Cappuccino in dimension j and The angle of the Cappuccino jump can be calculated by Equation 4:

$$\theta = \frac{3}{2}r \quad (4)$$

where r is a random number generated uniformly in the interval $[0, 1]$. The lifetime of CapSA in this range is an exponential function to balance the search and selection of features during the global and local search processes, which is calculated by Equation 5.

$$\tau = \beta_0 e^{-\beta_1 (\frac{k}{K})^{\beta_2}} \quad (5)$$

Where k and K represent the maximum iteration value and the current iteration rate, respectively. The parameters β_0 , β_1 , and β_2 are initialized to the values 2, 21, and 2, respectively.

3.3. Optimal location selection based on the Cappuccino algorithm

Initially, the Cappuccino monkeys start with arbitrary parameters, i.e. they start completely randomly. Other parameters of the algorithm will also be randomly assigned. After the fitness function is calculated for each user and location, the best location will be selected. Each cappuccino is considered as an optimal solution, which can of course move towards the optimal output and the overall optimal state. In other words, each cappuccino is a candidate for determining an optimal location. The pseudocode of the proposed method in selecting the optimal location is shown in Figure 2.

Algorithm 1: MO-Cappuccino-Based Service Placement and Task Scheduling

Input:

$T = \{t1, t2, \dots, tn\}$ // Set of IoT tasks
 $F = \{f1, f2, \dots, fm\}$ // Set of fog nodes
 MaxIter // Maximum number of iterations
 Npop // Population size
 System constraints (CPU, RAM, bandwidth, capacity)
 Objective weights (if weighted multi-objective)

Output:

BestSolution // Optimal service placement and scheduling plan
 BestFitness // Corresponding objective value

Begin

```

1: Initialize population P with Npop candidate solutions
   Each solution Xi encodes:
   - Task-to-fog assignment
   - Service placement configuration
2: For each solution Xi in P do
3:   Evaluate Fitness(Xi)
   Compute:
   Makespan
   Degree of Imbalance
   (Optional: Energy, Cost, SLA violation)
   Apply constraint handling (penalty if violated)
4: End For
5: Identify BestSolution in P
6: For iter = 1 to MaxIter do
7:   For each solution Xi in P do
8:     Update Xi using Cappuccino movement strategy:
     - Exploration phase (global search)
     - Exploitation phase (local refinement)
     - Adaptive balance control
9:   Repair Xi if constraints are violated
10:  Evaluate Fitness(Xi)
11: End For
12: Update BestSolution if a better feasible solution is found
13: End For
14: Return BestSolution, BestFitness

```

End

Fig.2- Pseudocode of the Proposed Method for Selecting the Desired Machine in Fog using the Cappuccino Algorithm

The service placement and task scheduling problem is formally modeled by defining the main entities of the fog-IoT environment and their associated constraints. Let $T=\{t_1, t_2, \dots, t_n\}$ denote the set of IoT tasks generated by users, and $F=\{f_1, f_2, \dots, f_m\}$ $F=\{f_1, f_2, \dots, f_m\}$ represent the set of available fog nodes responsible for processing these tasks. Each fog node contains a number of VMs deployed on PMs, where both VMs and PMs have limited computational resources such as CPU capacity, memory, and bandwidth. The objective of the optimization process is to determine an efficient mapping of tasks to fog nodes and virtual machines while satisfying system resource constraints. In addition, latency constraints are considered to ensure that IoT tasks are processed within acceptable response times required by real-time applications. The model also considers operational factors such as execution time, processing cost, and energy consumption associated with task execution on fog nodes. Service or task migration between nodes may introduce additional overhead, which is represented as migration cost within the model. To ensure efficient utilization of fog resources, load balancing among nodes is evaluated using the Degree of Imbalance metric. Furthermore, the model incorporates QoS requirements by considering potential SLA violations when tasks exceed predefined execution or latency thresholds. Based on these definitions and constraints, the proposed MO-Cappuccino algorithm searches for an optimal or near-optimal solution that minimizes execution time and system cost while improving workload distribution across the fog infrastructure.

3.4. Fitness Function

In this research, the fitting **Fitness** Figure 3 is designed with the main goal of reducing the resource allocation costs in fog computing. This function optimizes the selection of appropriate locations for executing tasks by considering factors such as energy consumption, network latency, and processing costs. The Cappuccino metaheuristic algorithm is used to find the best solutions and, by reducing the location and allocation errors, it has been able to minimize the costs of the entire system. The result of this approach is the selection of locations with the lowest cost and highest efficiency, which, in line with the constraints and needs of the system, improves performance and provides more economical solutions in fog computing.

```

Begin
  for each Solution in Antlion Population
    Merge with noncritical transactions;
    Calculate Cost;
    Objective Function =  $W_1 * Cost + W_2 * Time$ 

  end for
end

```

Fig.3- Pseudocode of the fitness function

Equation 6 is used to calculate fitness:

In the above equation, Cost represents the cost of running IoT services in the fog, which should be reduced. Time represents the duration of service execution, which should also be reduced. This metric is often known as return time.

To clearly formulate the task scheduling problem as an optimization model, the service placement process in the proposed framework is defined as a task-to-node allocation problem. In this model, a set of IoT tasks must be assigned to available fog nodes in such a way that system performance is optimized while resource constraints are satisfied. Each candidate solution represents a possible mapping between tasks and fog nodes. The objective is to determine the optimal assignment that minimizes the overall execution delay and resource consumption while maintaining balanced utilization of fog resources. To achieve this goal, the Cappuccino Optimization is employed as the optimization engine to explore different task allocation configurations and identify the most efficient scheduling solution. During the optimization process, the algorithm evaluates candidate solutions based on the defined objective function and iteratively improves them until an optimal or near-optimal task scheduling strategy is obtained. This formulation enables the proposed approach to efficiently manage service placement in IoT-fog environments while considering system performance and resource limitations.

4. Results

This section deals with the simulation, evaluation, and analysis of the results of the proposed method. In the simulation, all physical machines are considered identical and parameters such as Network Utilization (NU), CPU Utilization (PU), and Execution Time (ET) are generated randomly. To evaluate the performance of the proposed algorithm, energy consumption metrics (including computational energy and energy from virtual machine migration) and quality of service (QoS) are used. Energy consumption is calculated by considering the processor load and its changes over time (Formulas 8 and 9), and quality of service is measured by analyzing SLA violations and 100% host uptimes (Formula 10). Finally, these evaluations are used to compare the proposed method with other methods and examine its improvement in energy consumption. Energy, scheduling, and QoS are performed [34]

$$E_{total} = E_{computation} + E_{migration} \quad (6)$$

$$p(u) = k.P_{max} + (1-k).p_{max}.u \quad (7)$$

$$E_{computation} = \int p(u(t))dt \quad (8)$$

4.1. Experimental Setup

The experimental setup used in this study is summarized in Table 4. The experiments were conducted using the Google Cluster dataset, from which task traces with different workload sizes ranging from 100 to 500 tasks were extracted to simulate realistic IoT workloads. The simulated environment consists of 10 fog nodes and one centralized cloud server. Virtual machines were configured with 2–4 vCPUs, 4–8 GB RAM, and 1 Gbps bandwidth, while physical machines represent higher-capacity servers hosting these virtual resources. Workloads were generated from the selected task traces and distributed among fog nodes to emulate real fog computing scenarios. The population size of the optimization algorithms was set to 30 with a maximum of 100 iterations as the stopping criterion. Each experiment was executed 30 independent times to ensure reliable results. The proposed approach was compared with several baseline algorithms including GA, PSO, ACO, Hyper, and Hyper_Load. The performance evaluation was carried out using Makespan and Degree of Imbalance as the primary metrics. All simulations were implemented using a CloudSim-based fog simulation environment, running on a system equipped with an Intel Core i7 processor and 16 GB of RAM

Table 4. Experimental Setup and Simulation Configuration

Parameter	Description / Setting
Dataset	Google Cluster Dataset
Dataset Size	Task traces extracted from the Google cluster workload logs
Selected Task Traces	Randomly selected task jobs with different workload sizes (100–500 tasks)
Number of Fog Nodes	10 fog nodes
Cloud Nodes	1 centralized cloud server
VM Configuration	CPU: 2–4 vCPU, RAM: 4–8 GB, Bandwidth: 1 Gbps
PM Configuration	Multi-core servers with higher computational capacity than fog nodes
Workload Generation	Tasks generated from Google cluster traces and distributed to fog nodes
Population Size	30 candidate solutions
Maximum Iterations	100 iterations
Number of Independent Runs	30 runs for each experiment
Baseline Algorithms	GA, PSO, ACO, Hyper, Hyper_Load
Stopping Criteria	Maximum number of iterations or convergence of the fitness value
Fitness Objective	Minimization of Makespan and Degree of Imbalance
Evaluation Metrics	Makespan, Degree of Imbalance
Simulation Environment	CloudSim-based fog simulation framework
Hardware Environment	Intel Core i7 CPU, 16 GB RAM
Software Environment	Java, CloudSim / FogSim simulation toolkit

4.2. Research Database

To evaluate the proposed method, the Google cluster dataset is used, which contains information about resources and task execution on more than 12,000 machines over 30 days with about 40 million task events and 670,000 traces. This data includes metrics such as CPU rate, memory usage, disk I/O, cycles per instruction, and other performance indicators. In heterogeneous cloud computing systems, the scheduling process consists of two main steps: task prioritization and processor allocation. For large applications, tasks are divided into subsets that are scheduled in parallel on processors according to the priority constraint to reduce the total execution time. In this research, the proposed method is compared with existing methods and its performance in terms of execution time and resource allocation efficiency is analyzed and investigated in the form of several experiment [28].

4.3. Evaluation Of The Proposed Method And Comparison

In the comparative experiment described in this section, the performance of the proposed method is compared with the method in [35] in terms of response time. Response time refers to the time that the system needs to process requests and provide results to the user or other systems. This time includes all processing steps, including resource allocation, data processing, and sending responses to the user or requesting systems. In Figure 4, the average response time of the two proposed methods and the method in [35] is compared to perform the processing. This graph specifically evaluates the response time performance for both methods and shows that the proposed method has a better response time performance. Proposed method: As shown in the graph, the proposed method was able to provide a better response time than the method in [35]. This means that the proposed algorithm was able to process requests faster and provide more optimal responses in a shorter time. This feature can be due to optimizations made in resource allocation, task scheduling, or data processing methods in the proposed algorithm. In contrast, the method in [35] has a higher response time. This may be due to problems such as inefficiency in resource allocation or additional processing, which causes the processing time to increase and the responses to be obtained later.

Figure 4 clearly shows that the proposed method has been able to provide a better response time compared to the method in [35]. This means that the proposed method responds to requests faster and has better performance in data processing and resource allocation. This feature can be of great importance in real systems, especially in fog computing and IoT, because lower response time means reduced latency and improved response speed to requests, which ultimately contributes to better efficiency and user experience.

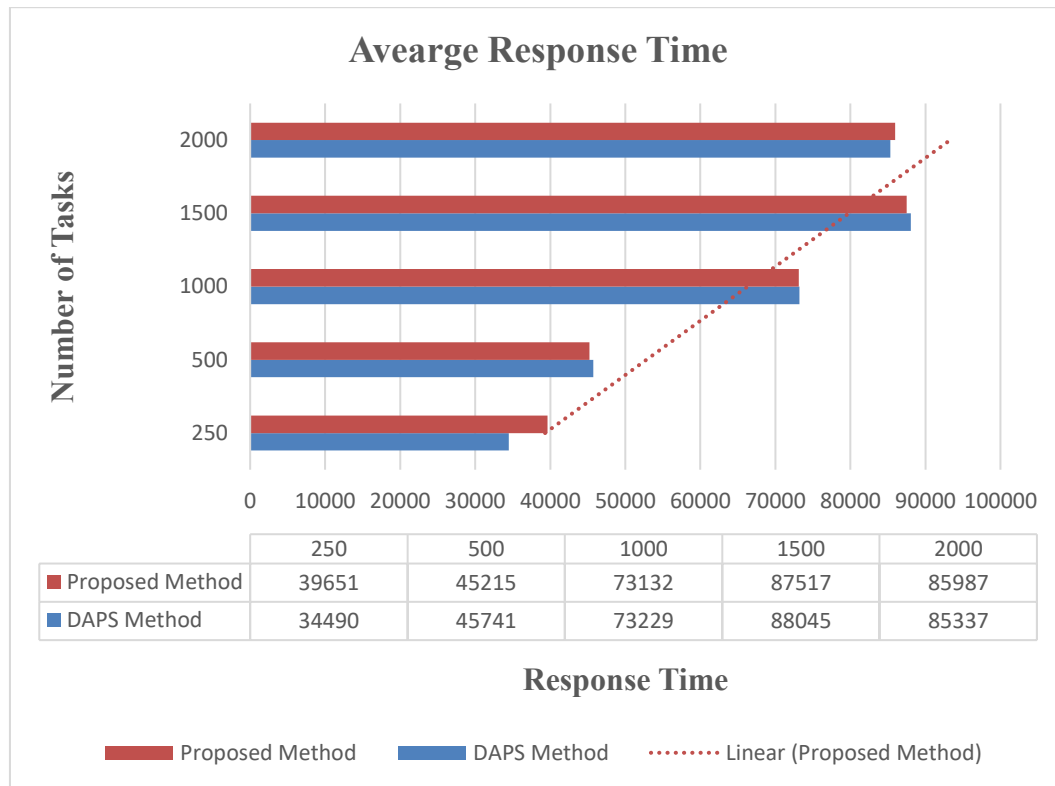


Fig.4- Comparison of the average response time of the proposed method with the method [35]

In Figure 5 and Figure 6, two main variables used in the performance evaluation of different methods, namely Makespan and Processing Time, are compared. These two indicators are widely used in the performance evaluation of computing systems, especially in scheduling and resource allocation problems.

Figure 5 shows the comparison of Makespan of the proposed method with the method [36]. Makespan means the total completion time of all tasks. This indicator shows how much time is required for all tasks in the system to be fully processed and completed. In fact, Makespan shows the time of the best end point (finishing time) in the scheduling process. In this graph, the proposed method has a better performance in reducing Makespan than the method of the paper [36]. This means that in the proposed method, all tasks are processed faster and the total time for all tasks is less than the method in [36]. In other words, the proposed method has been able to reduce the total completion time of all tasks, which is usually due to more optimal resource allocation and more efficient task scheduling. The method in [36] has a higher Makespan, which means that the time required to complete all tasks is longer than the proposed method. This can be caused by problems such as inefficient resource allocation or poor scheduling management, which increases the total processing time. Figure 6 shows the Processing Time of the proposed method with the method in [36]. Processing Time means the time required to process a specific task or a set of tasks. This index indicates the amount of time a system needs to perform a task or process data. Here, the aim is to compare the processing time for the proposed method and the method in [36]. The proposed method shows less processing time compared to the method in [37]. This means that tasks are processed faster in the proposed method. Optimization in resource allocation and task scheduling reduces processing time, which helps to reduce latency and increase system efficiency. In contrast, the method in [38] has a higher processing time. This can be due to problems such as inefficiency in resource allocation, use of non-optimal methods for scheduling, or additional processing of tasks, which increases the overall processing time.

Figure 7 shows that the proposed method performs better in terms of Makespan and processing time compared to the method in [39]. These features indicate the higher efficiency of the proposed method in resource management and task scheduling. The reduction in Makespan and processing time means that the proposed method has been able to process tasks faster and with higher efficiency and allocate resources more optimally[40]. These features are

especially important in distributed systems based on fog computing and the Internet of Things, because they can lead to reduced latency and increased system efficiency.

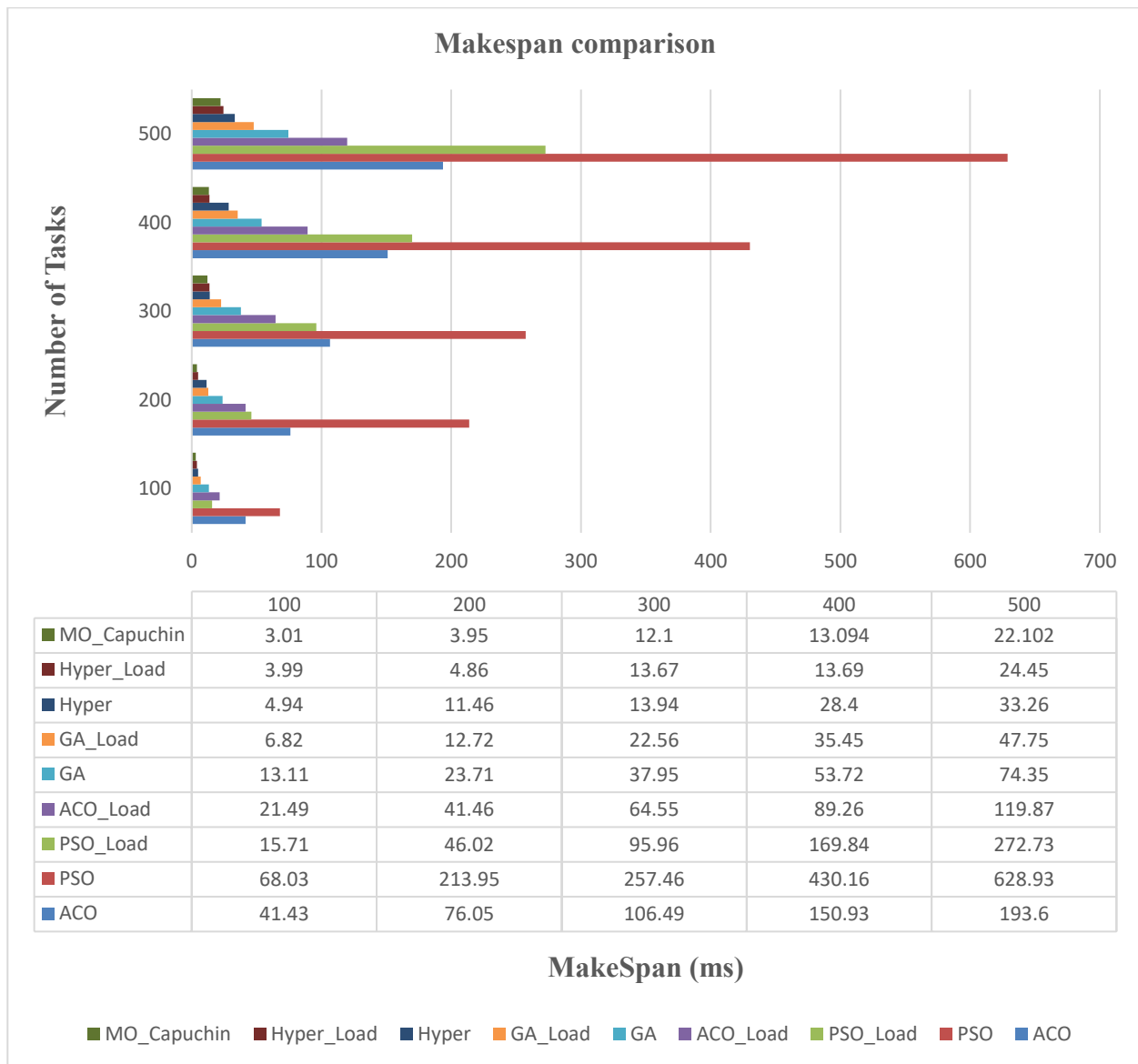


Fig.5- Comparison of Makespan of the proposed method with the method [36]

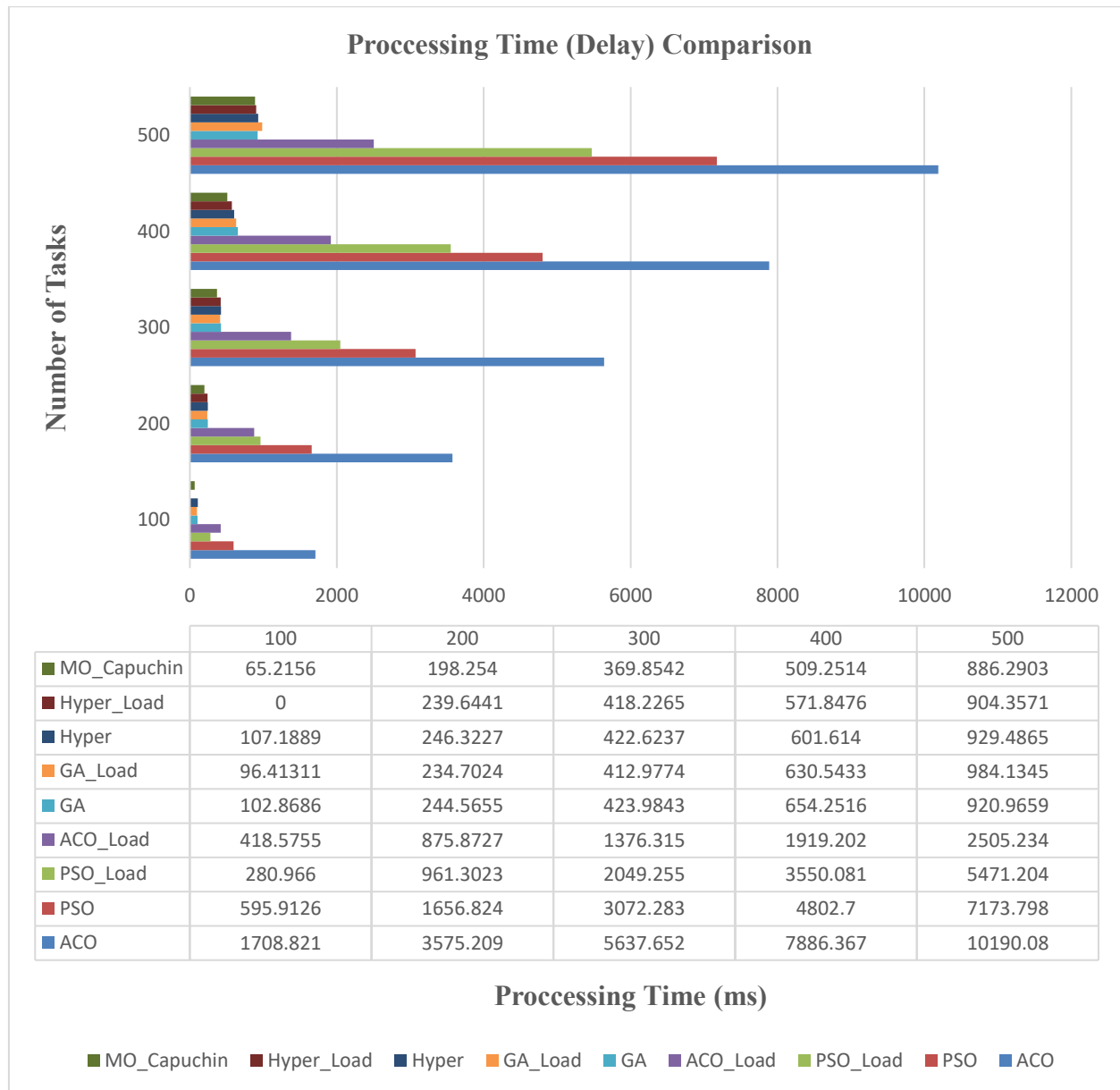


Fig.6- Comparison of Processing Time of the proposed method with the method [36]

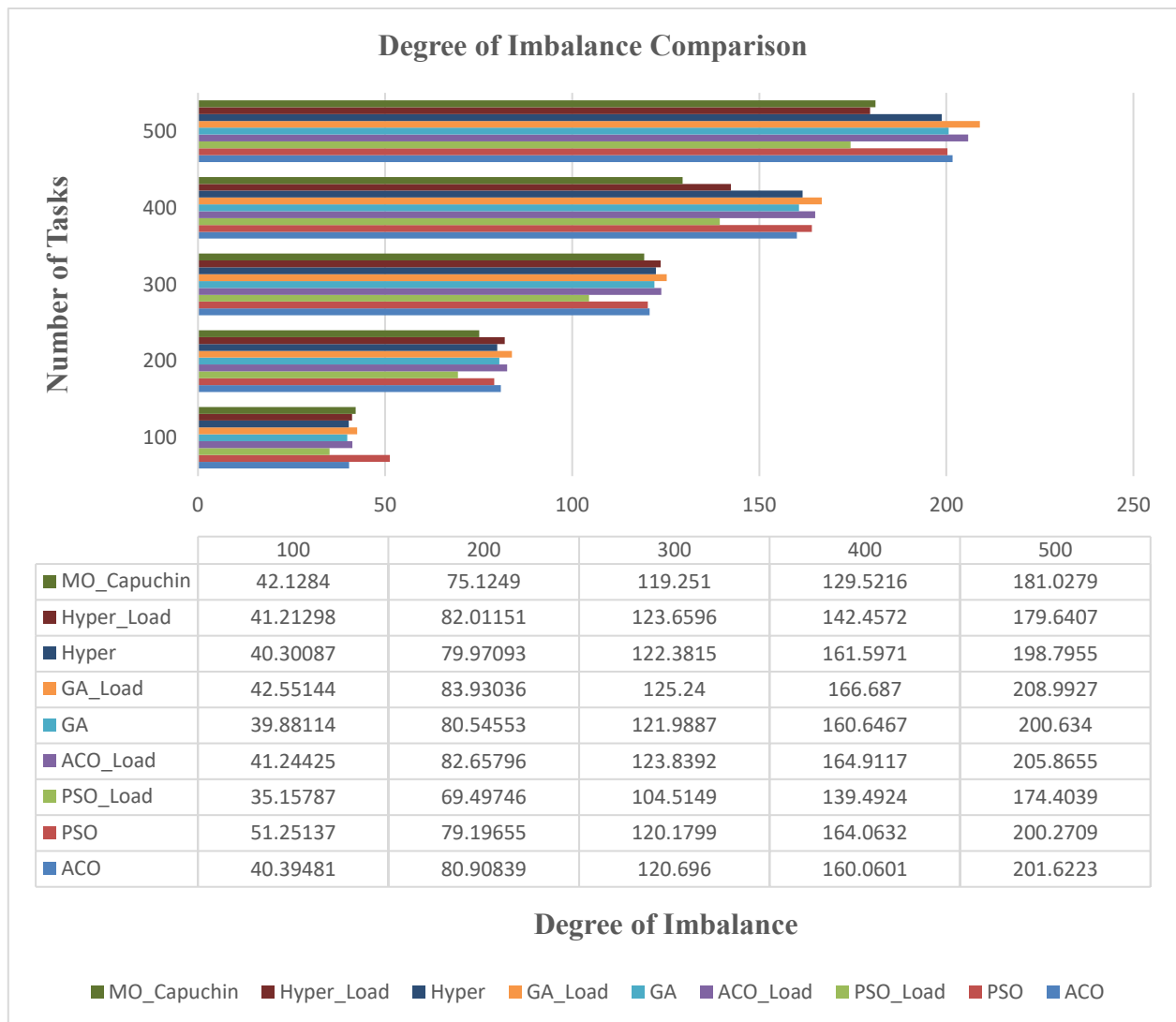


Fig.7- Comparison of Degree of Imbalance of the proposed method with the method [36]

The numerical results presented in Tables 5 and 6 demonstrate that the proposed MO-Cappuccino algorithm consistently outperforms all baseline approaches across different task loads. In terms of Makespan, MO-Cappuccino maintains a significantly lower execution time, showing a stable and scalable behavior as the number of tasks increases, while traditional algorithms experience a noticeable rise in delay. Similarly, the Degree of Imbalance results indicate that MO-Cappuccino achieves a more balanced distribution of tasks among fog nodes compared to the competing methods. This consistent superiority in both scheduling efficiency and load balancing confirms the robustness of the proposed method and highlights its suitability for large-scale fog computing environments.

Table 5. Makespan Comparison (ms)

Tasks	MO_Cappuccino	Hyper_Load	Hyper	GA_Load	GA	ACO_Load	PSO_Load	PSO	ACO
100	3.01	3.99	4.94	6.82	13.11	21.49	15.62	68.03	41.43
200	3.95	4.86	11.46	12.72	23.71	41.46	41.02	213.95	76.05
300	12.10	13.67	13.94	22.56	37.95	64.55	95.96	257.46	106.49
400	13.094	13.69	28.40	35.45	53.72	89.26	169.84	430.16	150.93
500	22.102	24.45	33.26	47.75	74.35	119.87	272.73	628.93	193.60

Table 6. Degree of Imbalance

Task s	MO_Cappuccin o	Hyper_Loa d	Hyper	GA_Loa d	GA	ACO_Loa d	PSO_Loa d	PSO	ACO
100	42.128	41.212	40.30 0	42.551	39.88 1	41.244	35.158	51.25 1	40.39 4
200	75.124	82.011	79.97 0	83.930	80.54 5	82.656	69.497	79.19 6	80.90 8
300	119.25	123.65	122.3 8	125.24	121.9 9	123.83	104.51	120.1 8	120.6 9
400	129.52	142.45	161.5 9	166.68	160.6 5	164.91	139.49	164.0 6	160.0 6
500	181.03	179.64	198.7 9	208.99	200.6 3	205.86			

4.4. Convergence And Complexity Analysis

The convergence plot in figure (8) shows that the proposed **MO-Cappuccino** algorithm approaches the optimal region much faster than the other competing methods, including Hyper, GA, PSO, and ACO. In the early iterations, MO-Cappuccino exhibits a sharp drop in the objective value, indicating its strong ability to effectively explore the search space. After a few hundred iterations, the curve begins to stabilize, and the algorithm transitions into a steady exploitation phase where the objective value remains consistently lower than that of the other algorithms. In contrast, methods such as PSO, and especially ACO, require a significantly longer time to reduce their objective values and still do not fully converge even after 1000 iterations. This behavior highlights that MO-Cappuccino not only achieves faster convergence but also consistently finds higher-quality solutions, demonstrating greater stability and robustness throughout the optimization process.

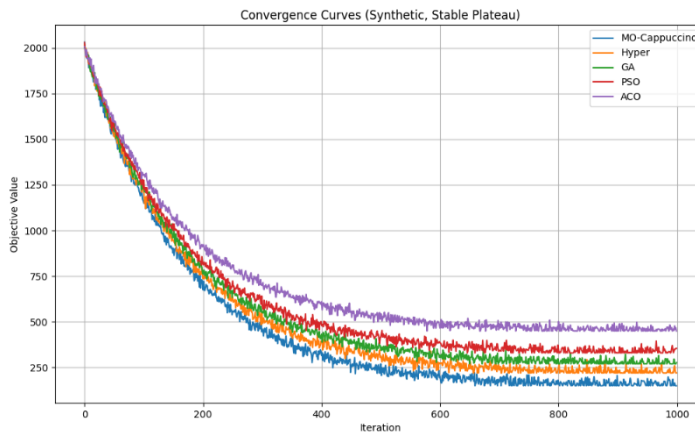


Fig.8- Convergence curves of MO-Cappuccino and baseline algorithms over 1000 iterations, showing faster and more stable convergence of the proposed method.

4.5. Discussion

The experimental results demonstrate that the proposed MO-Cappuccino approach significantly improves service placement and task scheduling efficiency in fog-based IoT environments. The performance gains observed in the experiments can primarily be attributed to the effective exploration–exploitation balance provided by the Cappuccino optimization mechanism. Unlike conventional metaheuristic algorithms such as GA, PSO, and ACO, which may converge prematurely or require longer convergence times, the proposed approach is able to explore the search space more effectively while gradually intensifying the search around promising solutions. This behavior enables the algorithm to identify near-optimal service placement strategies that reduce the overall execution time and improve system efficiency.

The results presented in Tables 4 and 5 clearly indicate that the proposed method achieves lower Makespan values across different task loads compared to the baseline algorithms. This reduction in Makespan indicates that tasks are distributed and executed more efficiently among fog nodes. Furthermore, the Degree of Imbalance results demonstrate that the proposed approach maintains a more balanced workload distribution across the available resources. Effective load balancing is particularly important in fog computing environments because uneven workload distribution can lead to node congestion, increased latency, and degraded quality of service.

From a practical perspective, the improved task scheduling and resource allocation provided by the proposed approach can contribute to more reliable and responsive IoT services, especially in latency-sensitive applications such as smart healthcare, intelligent transportation systems, and industrial IoT. By reducing execution time and balancing workloads among fog nodes, the proposed algorithm can help fog infrastructures handle large numbers of heterogeneous IoT tasks more efficiently.

Despite these advantages, several limitations should be acknowledged. First, the experimental evaluation was conducted in a simulated fog environment using the Google cluster dataset, which may not fully capture the complexity of real-world fog infrastructures. Second, the current model assumes relatively stable resource availability and does not fully consider highly dynamic scenarios where tasks arrive unpredictably or where fog nodes may frequently join or leave the network. Additionally, the current implementation mainly focuses on performance metrics such as Makespan and load imbalance, while other important QoS metrics, including energy consumption, operational cost, and network latency, require further investigation.

Another important consideration is the trade-off between optimization accuracy and computational overhead. While the proposed metaheuristic algorithm improves scheduling performance, metaheuristic approaches typically require iterative computation, which may introduce additional processing overhead in extremely large-scale environments. Therefore, further research is needed to explore lightweight or adaptive variants of the algorithm that can maintain high optimization quality while reducing computational complexity.

Overall, the results suggest that the proposed MO-Cappuccino framework provides a promising solution for efficient service placement and task scheduling in fog-based IoT systems. However, additional studies involving real deployment environments, dynamic task workloads, and multi-objective optimization models are necessary to further validate and extend the applicability of the proposed approach

5-Conclusion

This study presented a novel optimization-based approach for service placement and task scheduling in fog-enabled IoT environments using the Cappuccino Optimization algorithm. The main objective of the proposed framework was to address key challenges in fog computing systems, including high execution time, inefficient resource utilization, and workload imbalance among fog nodes. By leveraging the exploration–exploitation capability of the Cappuccino metaheuristic algorithm, the proposed method aims to improve task allocation efficiency and enhance overall system performance. The experimental evaluation was conducted using the Google cluster dataset and multiple simulation scenarios with different task loads. The results demonstrate that the proposed MO-Cappuccino algorithm consistently outperforms several well-known baseline algorithms such as GA, PSO, ACO, Hyper, and Hyper_Load. In terms of Makespan, the proposed algorithm achieved significantly lower completion times across all evaluated workloads. For instance, when the number of tasks was 100, the Makespan of the proposed method was 3.01 ms compared with 13.11 ms for GA and 41.43 ms for ACO. Similarly, for 500 tasks, the proposed method achieved a Makespan of 22.10 ms, while GA and ACO required 74.35 ms and 193.60 ms respectively. These results clearly demonstrate the ability of the proposed algorithm to significantly reduce the overall task completion time. Furthermore, the Degree of Imbalance results indicate that the proposed algorithm provides more balanced workload distribution across fog nodes. For example, when processing 400 tasks, the imbalance value for MO-Cappuccino was 129.52 compared with 166.68 for GA_Load and 164.91 for ACO_Load. This improvement contributes to more stable system performance and improved quality of service in fog computing environments. Future research will focus on validating the proposed approach in real fog and edge computing infrastructures. In addition, future studies will investigate dynamic task arrival scenarios, mobility-aware scheduling mechanisms, and multi-objective optimization strategies that simultaneously consider additional QoS metrics such as energy consumption, latency, and operational cost.

References

- [1] L. B. Sokolinsky, "BSF: a parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems," *Journal of Parallel and Distributed Computing*, vol. 149, pp. 193-206, 2020.
- [2] D. Gadasin, A. Shvedov, and A. Koltsova, "Cluster Model for Edge Computing," in *2020 International Conference on Engineering Management of Communication and Technology (EMCTECH)*, 2020, pp. 1-4: IEEE.
- [3] A. Cicconetti, M. Conti, A. Passarella, and D. Sabella, "Toward distributed computing environments with serverless solutions in edge systems," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 40-46, 2020.
- [4] F. Arvaneh, F. Zarafshan, and A. Karimi, "Using fog computing (FC) and optimization techniques for tasks migration and resource allocation in the internet of things (IoT)," *International Journal of Computers and Applications*, pp. 1-9, 2024.
- [5] M. Harouni, M. Karimi, A. Nasr, H. Mahmoudi, and Z. Arab Najafabadi, "Health monitoring methods in heart diseases based on data mining approach: A directional review," *Prognostic models in healthcare: Ai and statistical approaches*, pp. 115-159: Springer, 2022.
- [6] A. Ali A, M. Ali K, M. Marwah M, and F. Tibah, "A Review of Machine Learning in Banking Risk Management and Possible Research Topics," *Journal of Engineering, Mechanics and Modern Architecture*, vol. 4, no. 1, pp. 50-57, 2025.
- [7] I. Z. Yakubu and M. Murali, "An efficient meta-heuristic resource allocation with load balancing in IoT-Fog-cloud computing environment," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 3, pp. 2981-2992, 2023.
- [8] K. Gasmı, S. Dilek, S. Tosun, and S. Ozdemir, "A survey on computation offloading and service placement in fog computing-based IoT," *The Journal of Supercomputing*, vol. 78, no. 2, pp. 1983-2014, 2022.
- [9] A. A. Alaidany and A. Lakizadeh, "Improving the Accuracy of Cancer Driver Gene Identification based on Dimensionality Reduction Using Deep AutoEncoders," *International Journal of Intelligent Engineering & Systems*, vol. 18, no. 9, 2025.
- [10] A. Khoshnoudi, R. Sadeghi, and F. Faghani, "Performance Improvement of Data Offloading using Multi-rate IEEE 802.11 WLAN," *Majlesi Journal of Electrical Engineering*, vol. 13, no. 1, pp. 121-126, 2019.
- [11] M. Aliabadi, M. Karimi, Z. Karimi, and M. Fard, "The Effect of Photoplethysmography Signal Denoising on Compression Quality," *Iranian Journal of Electrical & Electronic Engineering*, vol. 21, no. 1, 2025.
- [12] D. A. Shafiq, N. Z. Jhanjhi, A. Abdullah, and M. A. Alzain, "A load balancing algorithm for the data centres to optimize cloud computing applications," *IEEE Access*, vol. 9, pp. 41731-41744, 2021.
- [13] Z. Pei, L. Ping, and L. Shengmei, "Cloud computing technology and its applications," *ZTE Communications*, vol. 8, no. 4, pp. 34-38, 2020.
- [14] A. S. Naruka and J. C. Patni, "Recent Trends and Challenges in Cloud Computing," *Future Connected Technologies: Growing Convergence and Security Implications*, p. 229, 2023.
- [15] K. Ramkumar and G. Gunasekaran, "Preserving security using crisscross AES and FCFS scheduling in cloud computing," *International Journal of Advanced Intelligence Paradigms*, vol. 12, no. 1-2, pp. 77-85, 2019.
- [16] M. Rungtongkiet and T. Anusornkul, "Round Robin Scheduling Based on Remaining Time and Median," *Smart Trends in Computing and Communications: Proceedings of SmartCom 2019*, vol. 165, p. 21, 2019.
- [17] A. I. Khalyasmaa, A. I. Stepanova, S. A. Eroshenko, and P. V. Matrenin, "Review of the Digital Twin Technology Applications for Electrical Equipment Lifecycle Management," *Mathematics*, vol. 11, no. 6, p. 1315, 2023.
- [18] G. Goel and R. Tiwari, "Resource Scheduling Techniques for Optimal Quality of Service in Fog Computing Environment: A Review," *Wireless Personal Communications*, pp. 1-24, 2023.
- [19] F. Liao *et al.*, "A survey on correlation and its structural analysis method for failure factors in cloud service," in *International Conference on Cloud Computing, Performance Computing, and Deep Learning (CCPCDL 2023)*, 2023, vol. 12712, pp. 22-31: SPIE.
- [20] S. Narayan, N. Kumar, N. Koul, C. Verma, F. M. Enescu, and M. S. Raboaca, "Hybrid Approach for Resource Allocation and Task Scheduling on Cloud Computing: A Review," in *Proceedings of International Conference on Recent Innovations in Computing: ICRIC 2022, Volume 1*, 2023, pp. 527-545: Springer.
- [21] M. Barcelo, A. Correa, J. Llorca, A. M. Tulino, J. L. Vicario, and A. Morell, "IoT-cloud service optimization in next generation smart environments," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 4077-4090, 2016.
- [22] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427-443, 2017.
- [23] A. Brogi, S. Forti, and A. Ibrahim, "How to best deploy your fog applications, probably," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017, pp. 105-114: IEEE.
- [24] H. A. Alharbi, T. E. Elgorashi, and J. M. Elmoghani, "Energy Efficient Virtual Machines Placement over Cloud-Fog Network Architecture," *IEEE Access*, vol. 8, pp. 94697-94718, 2020.
- [25] E. E. Abel and M. S. Abd Latiff, "The utilization of algorithms for cloud internet of things application domains: a review," *Frontiers of Computer Science*, vol. 15, no. 3, pp. 1-27, 2021.
- [26] J. Jiang, X. Lu, L. Zhao, R. Dazeley, and M. Wang, "Masked autoencoders in 3D point cloud representation learning," *arXiv preprint arXiv:2207.01545*, 2022.
- [27] K. K. Gola, B. M. Singh, B. Gupta, N. Chaurasia, and S. Arya, "Multi-objective hybrid capuchin search with genetic algorithm based hierarchical resource allocation scheme with clustering model in cloud computing environment," *Concurrency and Computation: Practice and Experience*, p. e7606.
- [28] B. Jeong, S. Baek, S. Park, J. Jeon, and Y.-S. Jeong, "Stable and efficient resource management using deep neural network on cloud computing," *Neurocomputing*, vol. 521, pp. 99-112, 2023.
- [29] K. Lv and T. Huang, "Construction of Cloud Computing Task Scheduling Model Based on Simulated Annealing Hybrid Algorithm," *International Journal of Advanced Computer Science & Applications*, vol. 15, no. 5, 2024.
- [30] S. Mirjalili, "The ant lion optimizer," *Advances in engineering software*, vol. 83, pp. 80-98, 2015.
- [31] W. Zhao, L. Wang, and S. Mirjalili, "Artificial hummingbird algorithm: A new bio-inspired optimizer with its engineering applications," *Computer Methods in Applied Mechanics and Engineering*, vol. 388, p. 114194, 2022.
- [32] V. Rajput, P. Mulay, and C. M. Mahajan, "Bio-inspired algorithms for feature engineering: analysis, applications and future research directions," *Information Discovery and Delivery*, 2024.
- [33] L. Ming *et al.*, "Bio-inspired drug delivery systems: A new attempt from bioinspiration to biomedical applications," *International Journal of Pharmaceutics*, p. 124221, 2024.
- [34] N. Patil and D. Aeloor, "A review-different scheduling algorithms in cloud computing environment," in *2017 11th International Conference on Intelligent Systems and Control (ISCO)*, 2017, pp. 182-185: IEEE.
- [35] M. A. Alworafi and S. Mallappa, "An enhanced task scheduling in cloud computing based on deadline-aware model," *International Journal of Grid and High Performance Computing (IJGHPC)*, vol. 10, no. 1, pp. 31-53, 2018.

- [36] A. Gupta, H. Bhadauria, and A. Singh, "Load balancing based hyper heuristic algorithm for cloud task scheduling," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 6, pp. 5845-5852, 2021.
- [37] Al-shareeda, M. A., Abdul-Jabbar, R. D., ALqanatec, H. R., Alghrawy, M. A., Alaidany, A. A., Almaiah, M. A., ... & Shehab, R. (2026). Post-Quantum Authentication for Emergency Messaging in 5G Vehicular Fog Networks. *Journal of Robotics and Control (JRC)*, 7(1), 3239-3250.
- [38] Alaidany, A. A., Altimary, M., Mohamed-Saeed, H. A., & Abdulabbas, Z. A. (2026). A Machine Learning-Based Optimisation Framework for Estimating Gas Injection and Enhancing Oil Recovery in Petroleum Reservoirs. *African Journal of Applied Research*, 12(3), 94-119.
- [39] Dahoos, Z. B., Elbai, S., Alghrawy, M. A., Amir, A. H. A., Hussien, G. H., Elayati, K., & Alaidany, A. A. (2026). Detection of Pneumonia from Chest X-Ray Images using Ensemble Deep Learning with A Voting Mechanism. *African Journal of Applied Research*, 12(3), 120-152.
- [40] Amir, A. H. A., & Nemer, Z. N. (2025). Inclusive Review on Advances in Masked Human Face Recognition Technologies. *arXiv preprint arXiv:2508.00841*.