IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 27, Issue 2, Ser. 3 (Mar. – Apr. 2025), PP 14-25
www.iosrjournals.org

Computation Offloading In Mobile Edge Computing Via Hybrid GA And Improved Inertia Weight Of PSO Algorithm

Nwogbaga, Nweso Emmanuel[1,*]; Latip, Rohaya[2,]; Emewu, Benedict Mbanefo[3]; Okpara, Chukwuemeka[3]

Department Of Computer Science, Faculty Of Computer Science, Universiti Putra11Malaysia, Seri Kembangan, Malaysia
Department Of Communication Technology And Networks, Faculty Of Computer Science, Universiti Putra Malaysia, Seri Kembangan, Malaysia
Department Of Computer Science, David Umahi Federal University Of Health Sciences, Nigeria

Abstract
Cloud computing helps mobile devices to process tasks generated from the environment due to the mobile device's limited processing capability, storage memory, and battery life. Due to the distance between cloud infrastructures and mobile devices together with the problems of network fluctuations and bandwidth issues, cloud computing introduces delay, increased energy consumption, and response time of mobile device's requests. Mobile edge computing was introduced to bring cloud computing closer to mobile devices through computation offloading to minimize delay, energy consumption, and response time of mobile devices. This improved quality of service can only be achieved with an efficient resource allocation algorithm to appropriately schedule tasks in MEC during computation offloading. In this paper, A hybrid Genetic Algorithm and Improved Inertia Weight Particle Swarm Optimization (GAIIWPSO) algorithm is proposed for resource allocation in the MEC computation offloading system. The proposed GAIIWPSO algorithm enhanced the PSO inertia weight recalculation process which leads to the minimization of delay, response time, and energy consumption of mobile devices. The proposed algorithm is tested in a MEC environment through simulation and the results proved to be superior to the existing algorithms.

Keywords: *Mobile Edge Computing, Computation Offloading, Genetic Algorithm, Inertia Weight, Particle Swarm Optimization*

Date of Submission: 11-03-2025 Date of Acceptance: 21-03-2025

I. Introduction

Internet of things (IoT) applications are revolutionizing modern society. The improvement in user Quality of Experience (QoE) and Quality of Service (QoS) is progressing from both hardware and software perspectives. This research focused on the aspect of improving QoS from a software perspective. IoT applications take different dimensions ranging from Mobile Computing (MC), Mobile Edge Computing (MEC), Cloud Computing (CC) [1–3], or combinations of these computing paradigms. All this different computing architecture was introduced because of the limitations of mobile devices [4,5]. Mobile devices are limited in processing capability, battery life, and storage capacity. Because of these limitations, cloud computing was introduced to solve the problems of mobile devices. The introduction of cloud computing introduced increased delays in IoT task processing. MEC was introduced to reduce the delay involved in sending tasks from local mobile devices to distant cloud infrastructures. When multiple processing devices are involved, the decision on which tasks should be moved to MEC or the cloud becomes another problem. In addition to the decision on which tasks to be moved, another problem of which processor at MEC or cloud is the task going to be allocated also emerged. This paper studied task scheduling and resource allocation in MEC and proffers a solution to resource allocation through a meta-heuristic approach. Meta-heuristic algorithms are applied in different aspects of life to proffer a solution to problems that do not have a definite solution [6–8]. Computation offloading in MEC is one of the examples of such NP-hard problems. Particle swarm optimization algorithm is generally accepted for this type of problem because of its exploitation and exploration capabilities [9]. We chronologically studied the PSO algorithm right from inception in 1995 [10] till date and identified one of the major parameters that determine the accuracy and speed of convergence of the PSO algorithm. The most important parameter in PSO is the inertia weight which is updated at the end of PSO iterations. This inertia

weight update balances between exploration and exploitation processes. Different researchers have contributed to improving the inertia weight calculation [11–13].

We, therefore, experimented with different approaches to inertia weight modification processes and proposed inertia weight updates based on the current, average, and maximum fitness functions of the system within the problem area. We applied the proposed inertia weight approach to our proposed hybrid genetic algorithm and improved the inertia weight particle swarm optimization (GAIIWPSO) algorithm. We tested the GAIIWPSO algorithm in MEC offloading and resource allocation and the proposed algorithm improved the delay, response time, and energy consumption of the mobile devices. The contributions of this paper are as follows;

- Proposed genetic algorithm for tasks and resource initialization in MEC system.
- Proposed a novel inertia weight calculation formula for improving the exploration and exploitation of the PSO algorithm
- Proposed GAIIWPSO for optimal resource allocation in MEC
- Compared the results of GAIIWPSO and the existing IWPSO and CODM algorithms.

II. Review Of The Related Literature

The balance between the processes of exploration and exploitation of PSO is largely achieved by the inertia weight parameter. The contribution rate of a particle's prior velocity to its velocity at the present step is determined by the inertia weight. Eberhart and Kennedy first proposed the fundamental PSO in 1995 [10], and the initial proposed PSO does not have an inertia weight parameter. Shi and Eberhart [14] initially conceived the idea of Constant Inertia Weight in the year 1998. They expressed that high inertia Weight encourages global optimal whereas a low inertia Weight encourages local optimal. Furthermore, the idea of changing the inertia weight was presented by numerous researchers which improved the capabilities of PSO to some extent. PSO inertia weight strategies are sequentially reviewed in this work. A random inertia weight approach was proposed and experimentally proved that the approach improves the convergence of the PSO algorithm Eberhart and Shi [15]. The approach of linearly reducing the inertia weight of PSO was proposed by [16], which improved the efficiency and performance of the convergence of the PSO algorithm. Their experiment proved that 0.9 to 0.4 inertia weight value produces excellent results. Despite that it converges fast; it can easily be trapped by local optima. Global-local best inertia weight was proposed in [17], and the best value for the inertial weight was based on the global and local optima function of the particle iterations. The PSO local minimum premature convergence fault was addressed by proposing an adaptive inertia weight algorithm in [18] to enhance the PSO searching strategy. The population is controlled through inertia weight adaptive adjustment. Optimization of the particle swarm algorithm (PSOSA) through simulated annealing to optimize the inertia weight was proposed by [19]. The PSOSA algorithm performance was tested in an urban planning problem. PSOSA algorithm proves to be better than existing algorithms in terms of sustaining the increased number of buildings in the urban planning problem and also on the speed of convergence. Guimin et al. [20] proposed two exponent inertia weight algorithms that use decreasing inertia weight ideas during the search process. Their experimental results showed that the proposed algorithms converged faster during the search process than other existing algorithms. Feng et al. [21], proposed chaotic inertia weight based on chaotic optimization merits. RIW and CRIW PSO algorithms were compared and CRIW PSO algorithm performs better than CRIW algorithm. Malik et al. [22] proposed inertia weight based on a sigmoid increasing pattern. The paper discovered that increasing the inertia weight value linearly leads to speedy convergence while changing the inertia weight using sigmoid function lead to speedy fitness. They proposed the combination of linearly increasing the inertia weight value together with the sigmoid function which proved to enhance the convergence towards the optimal solution. Oscillating inertia weight was proposed in [23], and the proposed algorithm periodically alternates the local search and global search processes. The paper discovered that the proposed algorithm outperforms the existing algorithms in terms of speedy convergence. Gao et al. [24] presented an enhanced PSO algorithm that combined a chaos mutation operator with inertia weight based on a logarithm decreasing method. They proposed that chaos mutation enables the algorithm to avoid being trapped in local optima while logarithm-decreasing inertia weight enables the algorithm to converge fast. Gao et al. [25] proposed exponent decreasing inertia weight combined with stochastic mutation to avoid the early convergence of PSO which sometimes leads to periodical oscillating occurrences by PSO. The paper modeled a mixed integer nonlinear programming optimization problem and used the PSO weight improvement approach to solve the problem. Zhang et al. [26] proposed Uniform initialization and Cosine inertia weight Particle Swarm Optimization (UCPSO) algorithm to improve the global search capability of PSO. UCPSO utilizes three effective enhancements in the PSO algorithm. First, is the cosine inertia weight based on variable period cosine function, uniform initialization strategy, and ranked base strategy. The proposed UCPSO improved the PSO global search. Kiani et al. [27] proposed an improved variant of PSO to minimize the premature convergence of the PSO algorithm. The paper improved standard PSO based on two strategies. First, the proposed algorithm controls the inertia weight through sine chaotic inertia weight

calculation to balance between local and global search. Secondly, the proposed algorithm adopts the tangent chaotic approach to control the acceleration coefficients for searching for an optimal solution. Agrawal and Tripathi [28] proposed a Cumulative Binomial Probability Particle Swarm Optimization (CBPPSO) algorithm. The algorithm improves the exploitation and exploration during the optimal solution-searching process. CBPPSO, when compared with other existing algorithms, shows better performance on three real-world engineering problems they were tested on. Asif et al. [29] proposed self-inertia weight adaptive PSO (SIW_PSO) based on the feature selection method to improve the performance of classification algorithms. The proposed algorithm proves to perform better in text classification problems because of the algorithm's capability in finding the feature subset. Chen et al. [9] proposed a Sigmoid Increasing inertia weight Particle Swarm Optimization (SIPSO) algorithm for the identification of structural damage. The SIPSO algorithm is based on structural vibration response optimization constraints. SIPSO uses a sigmoid increasing weight approach to improve the local and global search process. SIPSO improved the performance of structural damage identification as compared with the existing algorithms. SIPSO speed of convergence improved and the accuracy of identification increased. The proposed algorithm suggests an improvement from the standard PSO. Deng et al. cooperative offloading decision method (CODM) algorithm is proposed in [30]. The paper proposes offloading decision with cooperative relay selection, power allocation to address the problem computation offloading in MEC. The proposed algorithm addressed the issue response time in mobile edge computing. [31] proposed user centric joint optimization loading algorithm based on improved dynamic inertia weight calculation approach. The proposed algorithm is a multi-objective approach to minimize energy consumption, delay, and price for mobile edge computing. From these literatures, it is obvious that the PSO is a very good meta-heuristic algorithm used in different field of live for searching for optimal solution. It is also discovered that Inertia weight value is an important parameter for PSO exploitation and exploration operations. We therefore, proposed enhanced PSO based on inertia weight calculation and applied it to resource allocation in mobile edge computing. There so many other works on improving the processes of MEC in terms of response time, energy consumption, and delay [32–35].

III. Problem Formulation And System Model

The system model of mobile edge computing is presented in Figure 1. The MEC aims at reducing the delay in responding to the request of the mobile device, minimizing the mobile device's energy consumption, and minimizing the response time for every request sent from the mobile device. Assuming the MEC system consist of *J* mobile devices and *K* cloud devices, the mobile devices generate tasks that require processing. The tasks can either be processed within the local devices or offloaded to the cloud.

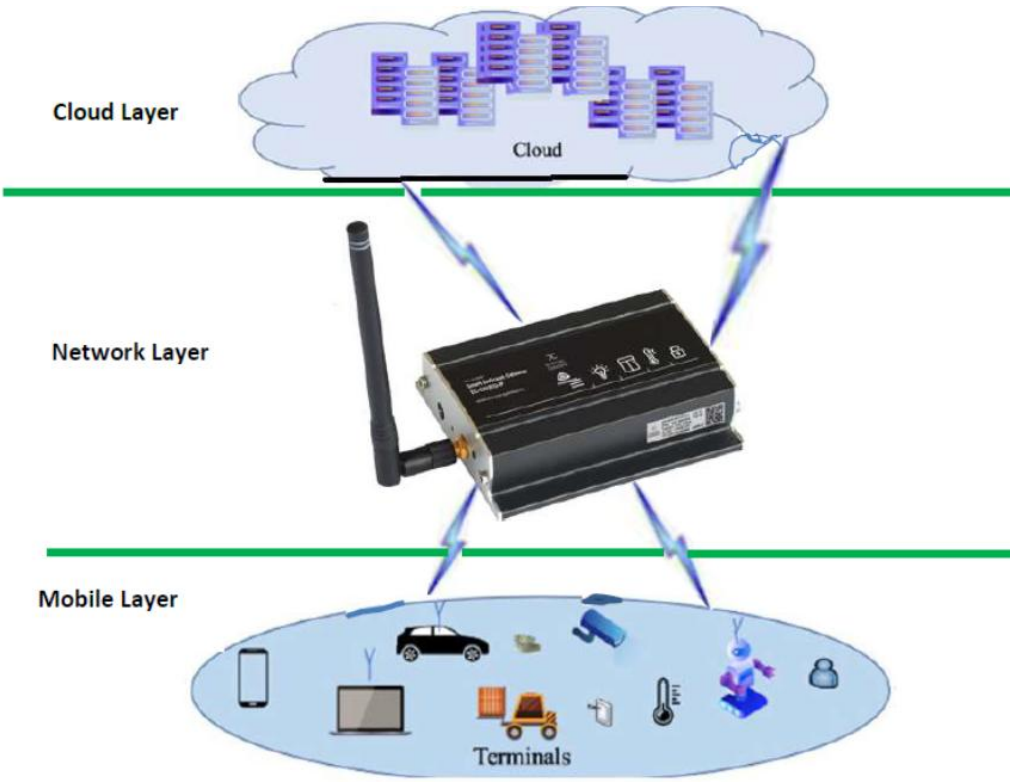

Fig. 1 System Model

The decisions on which tasks to be processed at the local device and those to be offloaded to the cloud are necessary to minimize delay, mobile device energy consumption, and response time in MEC. The problems of delay in task processing, energy consumption, and response time in the MEC system are modeled as follows:-

Time Delay Problem Modeling

In deciding whether to offload a task, the expected processing time for the generated task is estimated for all the devices at all the layers. The essence of estimating the task processing time at both the mobile layer and cloud layer is to determine whether to process the task at the mobile or to offload it to the cloud. The expected processing time at the layers is estimated at the mobile layer as follows.

Time to process the task on the mobile (P_m) $= \frac{worklaod\ (\emptyset)}{processing\ capability\ at\ mobile\ (PC_m)}$ (1)

where $\emptyset$ is the workload and PC_m is the available processing capability of the mobile device.

Time to process the task in the cloud (P_{cloud}) $= \frac{\emptyset}{PC_c}$ (2)

where PC_c is the available processing capability of cloud devices.

In estimating the transmission time, the mobile device uplink bandwidth is considered.

Time to transmit the data from mobile to cloud (T_{cloud}) $= \frac{Task_{size}}{Bandwidth}$ (3)

$Task_{size}$ is the task data size to be offloaded and Bandwidth is the available mobile uplink bandwidth.

The time taken to receive the results from the cloud to the mobile is assumed to be negligible. This is because data have been processed and there is no need to resend the data back to mobile. The only signal sent back is the required action. In addition, the cloud down Bandwidth is very high compared to the amount of data size involved in the signals sent.

After estimating the processing time at both the mobile device and cloud together with the transmission time required, the next thing is to check whether to offload the task from the mobile or not using Equation (4). This equation compares the time it will take the mobile device or the cloud to respond to the mobile request.

$P_m >= T_{cloud} + P_{cloud}$ (4)

When the condition in Equation (4) is true, it means that the time required to process the workload on the mobile device is more than the sum of the time required to process the workload at the cloud layer and the transmission time. Delay is the time difference between the task's expected response time and the real (actual) response time.

Delays can either occur during the queuing process, transmission, processing, or both. In computational offloading, the major challenges are how to offload computationally intensive tasks to remote processing nodes without increasing the delay [36]. This metric is efficient in evaluating the performance of a system in terms of delay which also affects the response time. The system with minimal delay encourages IoT applications in latency-sensitive cases such as healthcare, connected cars, augmented reality, smart home, and smart cities. In latency-critical applications, if the latency target is exceeded, it will typically result in application failure. Latency and delay are used interchangeably in most literature [37,38]. In this paper, we proposed GAIIWPSO for efficient and dynamic resource allocation which minimizes the delay in the system. The delay (D) is calculated within the simulation environment according to equation (5);

$D = T_{real} - T_{cal}$ (5)

where T_{real} is the actual finish time of a task and T_{cal} is the expected (calculated) finish time of the task.

Energy Consumption Problem modeling

Energy consumption has been identified as an important metric in IoT applications [39]. Energy consumption is the amount of energy consumed by a resource to get a workload completed. The energy consumption rate is the ratio of the number of workloads the system has successfully executed to the total energy consumed to execute those tasks [39,40]. CPU chip architecture has a fixed energy consumption rate both for a busy time and idle time for the same architecture. The energy consumption of a device is proportional to the number of tasks processed. Let's assume that all mobile devices have equal bandwidth and equal energy consumption rates. The mobile device energy consumption (E_m) will be calculated according to Equation (6);

$E_m = E_b \times (T_{cloud} + P_m)$ (6)

where E_b is the mobile device energy consumption rate at a busy time, T_{cloud} is the time taken by a mobile device to transmit the offloaded tasks to the cloud (Equation 3), and P_m is mobile device processing time (Equation 1). The mobile device can be busy when it is transmitting tasks to the cloud and when it is processing the task. Therefore there are two parts of energy consumption, when it is processing tasks and when it is transmitting tasks to other processors [41].

Response Time Problem modeling

Response time measures the time between task arrival time at the mobile device and when the response is received [36,38,42,43]. It is also called the completion time required to complete a particular job. Put in another way, it is the difference between the workload finish time and the workload execution start time [44]. This metric is good for evaluating the performance of a system [45]. Response time increases with an increased number of operations and/or tasks [38]. We employed response time in this study to evaluate the performance of the proposed scheduling algorithm in terms of speedy response to IoT device requests compared to the current state of the art in computational offloading in a mobile edge computing environment. In this work, we measured the response time by taking the difference between task submission time and finish time within the simulation period. The response time ($T_{response}$) is calculated within the simulation environment according Equation (7);

$$T_{response} = T_{finish} - T_{arrival} \quad (7)$$

where T_{finish} is the task finished time and $T_{arrival}$ is the task arrival time.

IV. Proposed Genetic Algorithm And Improved Inertia Weight Particle Swarm Optimization (GAIIWPSO) Based Task Offloading Algorithm

In this research work, meta-heuristic algorithms were studied and enhanced. We followed discrete event simulation methodology as illustrated in Figure 2. We studied genetic algorithm and particle swarm optimization algorithms and from the inspirations gained from the study, proposed a Hybrid Genetic Algorithm and Improved Inertia Weight Particle Swarm Optimization (GAIIWPSO) algorithm to improve on the exploration and exploitation strategies of PSO. The proposed algorithm achieved the improved PSO search process in two important stages. First, the algorithm introduced a genetic algorithm for the initialization process to avoid premature convergence of standard PSO. Secondly, the proposed algorithm enhances the process of calculating the inertia weight of the standard PSO to improve the exploration and exploitation process of the standard PSO. These two enhancements gave rise to better results for the GAIIWPSO algorithm. The proposed algorithm is tested in mobile edge computing which improved the tasks-resource allocation in the system. We, therefore, present the two stages involved in the proposed GAIIWPSO algorithm.

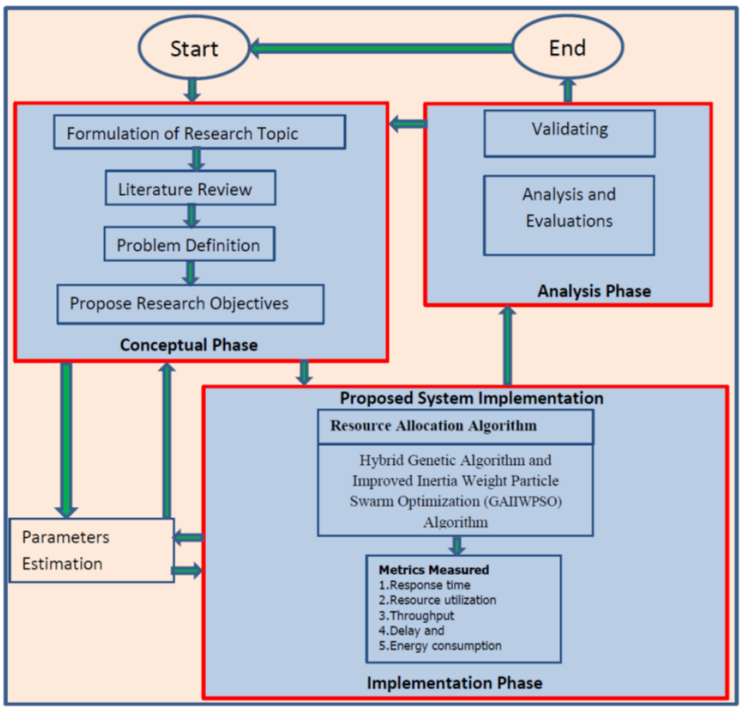

Fig. 2 Research Methodology

Initialization Using Genetic Algorithm

Genetic Algorithm (GA) is one the good heuristics algorithms that are applied widely in different areas of optimization problem-solving. GA algorithm is flexible and easy to implement. GA makes use of evolutions which makes it acceptable widely. The chromosomes mating and forming new offspring that evolve is one of its main characteristics that makes it applicable in solving an optimization problem. The basic GA algorithm is based on a set of possible solutions which represent optimization problem solutions. A given solution is a prospective candidate for an optimal solution to the optimization problem. Candidate representation is very

important because it is used to decide on the choice of a genetic operator to start with. In MEC, this candidate is a representation of a pair of tasks and the processing device that the task is assigned for processing. These three genetic operators include the selection operator, crossover operator, and mutation operator. In a genetic algorithm, the first step is the initialization process. In the MEC environment, this initialization means assigning all available tasks to all available processing devices randomly without minding whether each pair are the best pair or not. After the random pairing of tasks and devices, GA uses its three operators to re-assign those tasks to their best processors based on the fitness function.

Selection operator

To achieve convergence towards an optimum solution and avoid premature convergence, GA selects the best offspring to represent the parents in a new population. A sequence of offspring solutions is generated at each stage and the best offspring is selected to improve the pairing toward the optimal solution. The selection operator gives rise to a new generation. The selection operator is also called survival selection because it is not all solutions that survive. Some solutions will survive while others will die.

Crossover operator

The Crossover operator in GA is used to combine two solutions from the selection operator. The crossover operator combines the genetic materials usually called genes to form new offspring. The operator can combine more than two parent's solutions components wise to form new offspring. Some GA algorithms can simplify the crossover steps by randomly choosing the parents with uniform distribution.

Mutation operator.

The last GA operator makes changes from the solution by disturbing the present population from the crossover operator. It is based on a random modification of the candidate's genes to get a better fitness solution. The mutation operator has a mutation rate. It generates candidates which are less or equal to the mutation rate. It selects two genes randomly from the same chromosomes and checks whether they are the same. If they are not the same, they will retain their present positions, else, they will be swapped to generate a new population. This new population represent a new distribution of tasks and available resource in MEC. The GA algorithm illustrates how GA is used to initialize tasks for the improved inertia weight PSO as presented in Algorithm 1. For this experiment, GA runs for 10 iterations which means that the value of *i* in the GA algorithm is 10.

Algorithm 1: Genetic Algorithm

```
1  Initialize the population randomly
2  for i = 1 to number of iterations do
3      selection operator
       Input: Random Generated Chromosomes
       Output: Allocation Solution "Chromosomes best fitness"
4      Set: Number of Tournaments

5      for i = 0 To numberOfTournament do
6          id = Math.Random() * chromosomes-size
7          tournament[i] = get-chromosome(id)
8      Fitnessvalue ← tounament[i].Fitness
9      Transfer to Crossover operator
10     crossover operator
       Input: Two Chromosomes
       Output: Offspring Chromosome
11     r =(Math.Random()*chromosome.lenght)
12     for i = 0, j = 0 to r do
13         offspring-chromosome [j] = chromosome[i]
14     for i = r to chromosome.lenght do
15         offspring-chromosome [j] = chromosome2[r]
16     mutation operator
       Input: offspring-Chromosomes
       Output: New-Chromosome
17     Set: MutationRate = 0.5
        if Math.random() <MutationRate) then
18         t1 = Rand[0,1] * offspring-chromosome.lenght
19         t2 = Rand[0,1] * offspring-chromosome.lenght
20         if offspring-chromosome[t1] = offspring-chromosome[t2] then
21             Swap (offspring-chromosome[t1], offspring-chromosome[t2])
22 if Fitness is attained then
23     Exit
24 else
25     i = i + 1
26 end
```

Algorithm 1 Genetic Algorithm for Initializing Tasks in MEC

Improved Inertia Weight PSO (IIWPSO)

PSO is considered suitable for this study because IoT devices behave in the same way as a swarm. In mobile edge computing, connectivity is always through a wireless connection, in that case, the network resources are dynamically changing. The variations are as a result of mobility of mobile devices, bandwidth instability within the network [2,3]. For these reasons, the network workload at the cloud or mobile node varies frequently. In order to ensure the reliability of mobile devices receiving responses on time, the need to consider an proficient resource allocation algorithm arises. Particle swarm optimization algorithm is a heuristic algorithm which copies the intellect of a swarm. PSO practices communication and learning as their basic ideologies. Different particles generate their way to the best solution and communicate with other members. Every other members of the swarm get the information and learn the optimal solution to take. The particles create a common global best solution for the swarm. This idea of cooperation and intellect shown by the swarm makes them achieve better results quicker. The aim of particle swarm optimization algorithm is to discover the best solution through separate particles' cooperation and communication between the entire swarm to find the global best solution. The swarm particles are potential solutions. Every particle remembers its current velocity *(V(t))* and current position *(X(t))*. The particles also remember their best position called particle best position (X_{pbest}) and the swarm's best position called the global best position (X_{gbest}). The particles search for optimal solutions iteratively. The velocity and position of the particles are updated using Equations (8 & 9).

$$X(t+1) = X(t) + V(t+1) \quad (8)$$

where $X(t+1)$ is the position for the particle at t +1 time (new position), $X(t)$ is the initial position the particle, and $V(t+1)$ is the new velocity of the particle.

$$V(t+1) = \omega V(t) + C_1 R_1(0,1) * (X_{pbest} - X(t)) + C_2 R_2(0,1) * (X_{gbest} - X(t)) \quad (9)$$

where ω is the inertia weight, C_1 and C_2 are the acceleration coefficients for the X_{pbest} and gX_{gbest} respectively. X_{pbest} is the particle's best position. X_{gbest} is the global best position for swarm, and R_1 and R_2 is a numbers between 0 and 1 choosed randomily.

The inertia weight of PSO represents the influence that the previous velocity has on the new velocity. When the value of the inertia weight is high, the particles are prevented from falling into the region of interest immediately. A large inertia weight value makes the particles keep on searching outside the region of interest for some time. Small inertia weight is capable of causing the particles to move to the region of interest immediately. This means that a large inertia weight encourages global search capability known as exploration whereas a small inertia weight encourages local capabilities to search referred to as exploitation. Exploration helps PSO to avoid being trapped in the local optimum, but exploration can minimize the PSO convergence accuracy and lowers the convergence speed. Exploitation increases convergence speed and increases the convergence accuracy, but can cause PSO to fall into premature convergence and can easily be trapped into local optimal. The exploration and exploitation function influences the performance of PSO greatly. Therefore, choosing the appropriate inertia weight is necessary for PSO to perform optimally. The inertial weight of PSO was introduced in 1995 and has undergone a series of modifications as chronologically discussed in the literature review of this paper. In this paper, we experimented with different methods and proposed a novel inertia weight calculation method which is used to update inertia weight for each of the PSO iterations using the minimum, maximum, and average inertia weight values. The novel inertia weight proposed in this work improves the exploration and exploitation that leads to better accuracy and speed of convergence in the PSO algorithm. The proposed enhanced inertia-weight formula for PSO is calculated according to Equation (10);

$$\omega = \begin{cases} \omega_{min}, & f < f_a \\ \omega_{Ave}, & f = f_a \\ \omega_{max}, & f > f_a \end{cases} \quad (10)$$

where ω_{min}, ω_{max}, and ω_{Ave} are the inertia weight minimum, maximum, and average values respectively. f_a and f are the value of the average fitness function and the value of the present iteration fitness function. With this proposed inertia weight formula, PSO checks the fitness of the particles at every iteration, if the fitness function is not yet met, it will compare the current fitness value with the average fitness value based on Equation (10) and use the result to update the inertia weight value accordingly. The improved inertia weight PSO algorithm is presented in Algorithm (2).

Algorithm 2: IIWPSO Algorithm

```
Initialization: n, D, C1, C2, R1, R2, ω, Xpbest, Xgbest, X, V, and t
for t < tmax or Fitness achieved do
    for i = 1 to n do
        for d = 1 to D do
            Update velocity Vid according to Equation (9);
            Update velocity Xid according to Equation (8);
            Update Inertia Weight (ω) according to Equation (10);
        end
        if Fitness (Xi) < Fitness (Xpbest) then
            Fitness (Xpbest) = Fitness (Xi)
        if Fitness (Xpbest) < Fitness (Xgbest) then
            Fitness (Xgbest) = Fitness (Xpbest)
        end
        end
    end
    t = t + 1
end
```

Algorithm 2 IIWPSO Algorithm

Proposed GAIIWPSO Offloading Algorithm

This section presents the proposed hybrid GA and IIWPSO algorithm for resource allocation in the tasks offloading system. The process of offloading tasks from mobile devices to the cloud infrastructures is illustrated in Algorithm 3. Tasks are generated on mobile devices. The algorithm uses the generated tasks and the information about the mobile devices and the processing devices of the cloud to estimate the processing time required by the tasks at both the mobile layer and the cloud layer. It also estimates the transmission time required if the tasks are to be offloaded. The processing time on mobile is calculated according to Equation 1, while Equation 2 calculates the time required for processing the task in the cloud, and Equation 3 calculates the transmission time from mobile to the cloud. The algorithm compares the time to process the task on mobile with the sum of the time to transmit the task to the cloud and the time to process the task in the cloud. If the processing time at the mobile layer is higher, then there is a need to offload the task to the cloud. If task offloading exists according to Equation 4, then the algorithm uses Algorithms 1 and 2 to allocate the task to the available processing resources. If Equation 4 is not true, the task will be processed at the mobile layer. Finally, the proposed algorithm will calculate the delay experienced by the mobile device according to Equation 5, calculate the energy consumed by the mobile device according to Equation 6, and the response time according to Equation 7. The proposed GAIIWPSO algorithm is presented in Algorithm 3.

Algorithm 3: GAIIWPSO Tasks Offloading Algorithm

```
Generate tasks randomly at IoT layer
Calculate the processing time at IoT (Tm) and at Edge server (Tedge), and Cloud (TCP) required by
    the task based on processing capabilities at IoT and edge server as in steps 3 and 4
Pm = ω/PCm (According to Equation (1))
Pedge = w/PCE (According to Equation (2))
if Pm >= Tedge + Pedge (According to Equation (4)) then
    Task is offloadable
    if Task is offloadable then
        Initialize the tasks with the available devices according to Algorithm (1)
        Apply Algorithm (2) to finalize the resource allocation
    Process the task and return the results
else
if Task is not offloadable then
    Process the task at the IoT layer
    Calculate the Delay according to Equation (5)
    Calculate the energy consumption according to Equation (6)
    Calculate the response time according to Equation (7)
```

Algorithm 3 GAIIWPSO Tasks Offloading Algorithm

Experimental settings

Following the base paper system specifications and parameter settings, the experiment simulated the proposed GAIIWPSO algorithm and compared the result with the benchmark results. The proposed algorithm is implemented using Python 2.7 together with Networkx. The system comprises smart mobile devices, a smart gateway, and the cloud. The computational capacity of the cloud device is set at F = 10GHz while the CPU frequency of the smart devices is randomly distributed between 0.5GHz to 1GHz. The number of mobile devices in the network is set to be between 10 devices. The bandwidth of the individual edge devices is uniformly distributed within the range of 10 Mb/s to 1000 Mb/s. The tasks' latency is 0.5s each. The mobile device energy consumption rate at the busy time is 100mJ given as $E_b = 100mWatts$ in the base paper (1Watts = 1J/s). The device idle time power consumption rate is 10mJ. Table 1 presents the summary of the simulation parameters.

Table 1 Parameters Settings for GAEIWPSO Algorithm

Parameter	Value
Mobile devices	10
Mobile device APC	0.5 – 1GHz
Bandwidth	10mb/s – 1Gb/s
Mobile device power consumption rate at a busy time (E_b)	100mJ/s
Mobile device power consumption rate at idle time	10mJ/s

V. Results Discussions

The results for the proposed GAIIWPSO algorithm are discussed in this section. The delay experienced by the tasks and energy consumption recorded by the mobile devices in the simulations is compared with the IWPSO algorithm of [31] while the response time of the proposed algorithm is compared with the response time of the CODM algorithm according to [30].

Delay

Figure 3 shows the graph of the delay recorded in the simulations while Table 2 presents the values for the delay recorded for different sets of tasks simulated. The proposed GAIIWPSO algorithm achieved a better delay compared to the existing IIWPSO algorithm because the proposed algorithm was able to minimize the delay resulting from the resource allocation. The delay recorded in the simulations increases with an increase in the number of mobile device requests. In particular, when 200 tasks are simulated, the delay recorded by IWPSO is 25ms while the proposed GAIIWPSO recorded 15ms, which is 40% improvement in the delay of IoT application requests. Similarly, when 300 tasks are simulated, IWPSO recorded a delay of 44ms while GAIIWPSO recorded 35ms, which represents a 20% improvement. The result shows that as the number of input tasks increases, the percentage improvement reduces.

Table 2 Delay

Input Tasks (number)	Delay	
	IWPSO (ms)	GAIIWPSO (ms)
200	25	15
300	44	35
500	60	51
650	80	72

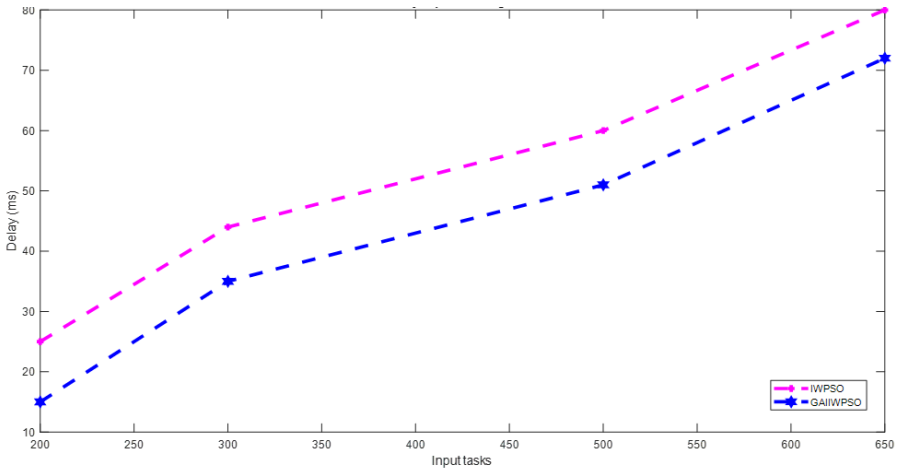

Fig. 3 Delay

Energy Consumption for GAEIWPSO

Figure 4 shows the energy consumption of the proposed GAIIWPSO algorithm and the base paper IWPSO algorithm. Table 3 presents the values recorded during the simulations. The total energy consumption increases as the input tasks increase. When the input tasks are 350 in number, the proposed GAIIWPSO algorithm's total energy consumption is 143J while the benchmark result energy consumption by the IWPSO algorithm is 230J representing a 37.8% improvement in the energy consumption. When the total input tasks increased to 650, the GAIIWPSO algorithm energy consumption is 254J as against 410J for the IWPSO algorithm representing 38.04% energy enhancement. The improvement is achieved because the proposed GAIIWPSO algorithm reduced the mobile device duration of usage by making the resource allocation in the network faster. The speedy resource allocation is achieved through the proposed enhancement of the PSO inertia weight calculation method and initializing the resource allocation with the GA algorithm.

Table 3 Energy Consumption for GAIIWPSO

Input Tasks	Energy Consumption (J)	
	IWPSO Algorithm	GAIIWPSO Algorithm
200	130	80
350	230	143
500	310	192
650	410	254

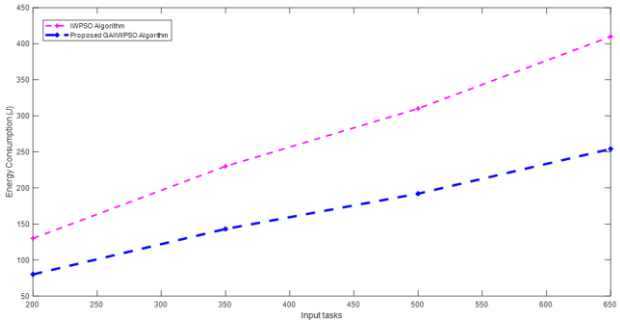

Fig. 4 Energy Consumption for GAIIWPSO

Response Time

The response time of the proposed GAIIWPSO algorithm is compared with the response time of the existing CODM algorithm. For example, when 30 numbers of tasks are simulated, the CODM algorithm response time is 12.5ms while the GAIIWPSO algorithm response time is 11ms. Likewise, when 40 numbers of input tasks are simulated, the CODM algorithm recorded 18.4ms response time while the proposed GAIIWPSO algorithm recorded 17ms response time. These results show that the proposed GAIIWPSO algorithm responds to mobile device requests faster than the existing CODM algorithm. Table 4 and Figure 5 represent the values and graph of the existing algorithm and proposed algorithm response time.

Table 4 Response Time for GAIIWPSO

Tasks (number)	Response Time (ms)	
	CODM	GAIIWPSO
10	5	4
20	7	5.5
30	12.5	11
40	18.4	17
50	22	19

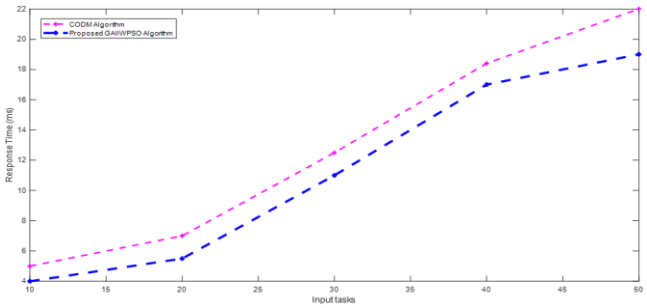

Fig. 5 Response Time for GAIIWPSO

VI. Conclusion

This research proposed the GAIIWPSO algorithm which minimized delay, mobile energy consumption, and response time of the mobile edge computing system. The algorithm achieves these by hybridizing GA and PSO with an enhancement of the inertia weight of PSO which improved the efficiency of the proposed algorithm resource allocation in the network. The algorithm performed better compared with IWPSO and CODM algorithms in terms of delay, energy consumption, and response time.

Authorship contribution statement

Nwogbaga, Nweso Emmanuel: Methodology, Software, Writing – original draft, Writing – review & editing. **Latip, Rohaya:** Project administration, Supervision, Resources, Formal analysis, Conceptualization. **Affendey, Lilly Suriani:** Supervision, Validation, Conceptualization, Formal analysis. **Rahiman, Amir Rizaan Abdul:** Writing – review & editing, Investigation, Validation. **Ituma, Chinagolum:** Validation, Resources, Project administration. **Ogbu, Henry Nwani:** Formal analysis, Investigation, Resources. **Agwu, Chukwuemeka Odi:** Software, Resources, Writing – review & editing. **Ogbaga, Ignatius Nwoyibe:** Formal analysis, Conceptualization, Software, Resources.

Declarations

No funding was received for conducting this study.

Financial interests

The authors have no relevant financial or non-financial interests to disclose.

Declaration of Competing Interest The authors declare that they have no known competing financial interest.

References

[1] E.C. Eze, S. Zhang, E. Liu, N.E. Nwogbaga, J.C. Eze, RECMAC: Reliable And Efficient Cooperative Cross-Layer MAC Scheme For Vehicular Communication Based On Random Network Coding Technique, In: 2016 22nd Int. Conf. Autom. Comput. ICAC 2016 Tackling New Challenges Autom. Comput., 2016: Pp. 342–347. Https://Doi.Org/10.1109/Iconac.2016.7604943.

[2] N.E. Nwogbaga, B.M. Emewu, I.N. Ogbaga, Critical Analysis Of Cloud Computing And Its Advantages Over Other Computing Techniques, J. Multidiscip. Eng. Sci. Technol. 3 (2016) 3955–3960.

[3] N.E. Nweke, F. Henry, Michael, J. S And Nwogbaga, Main Challenges Hampering Cloud Computing Adoption By Business Organizations, Res. J. Inf. Technol. 2 (2015) 1–11. Http://Www.Researchjournali.Com/Pdf/1225.Pdf.

[4] N.E. Nwogbaga, R. Latip, L.S. Affendey, A.R.A. Rahiman, Attribute Reduction Based Scheduling Algorithm With Enhanced Hybrid Genetic Algorithm And Particle Swarm Optimization For Optimal Device Selection, J. Cloud Comput. 11 (2022) 1–17. Https://Doi.Org/10.1186/S13677-022-00288-4.

[5] N.E. Nwogbaga, R. Latip, L.S. Affendey, A.R.A. Rizaan, Investigation Into The Effect Of Data Reduction In Offloadable Task For Distributed Iot-Fog-Cloud Computing, J. Cloud Comput. Adv. Syst. Appl. 10 (2021) 1–12. Https://Doi.Org/Https://Doi.Org/10.1186/S13677-021-00254-6.

[6] Z. Liao, X. Mi, Q. Pang, Y. Sun, History Archive Assisted Niching Differential Evolution With Variable Neighborhood For Multimodal Optimization ☆, Swarm Evol. Comput. 76 (2023) 101206. Https://Doi.Org/10.1016/J.Swevo.2022.101206.

[7] X. Chen, H. Yan, Y. Zheng, M. Karatas, Integration Of Machine Learning Prediction And Heuristic Optimization For Mask Delivery In COVID-19, Swarm Evol. Comput. 76 (2023) 101208. Https://Doi.Org/10.1016/J.Swevo.2022.101208.

[8] G. Rivera, L. Cruz-Reyes, E. Fernandez, C. Gomez-Santillan, N. Rangel-Valdez, C.A. Coello, An ACO-Based Hyper-Heuristic For Sequencing Many-Objective Evolutionary Algorithms That Consider Different Ways To Incorporate The DM ' S Preferences, Swarm Evol. Comput. 76 (2023) 101211. Https://Doi.Org/10.1016/J.Swevo.2022.101211.

[9] Z. Chen, Y. Wang, T.H.T. Chan, X. Li, S. Zhao, A Particle Swarm Optimization Algorithm With Sigmoid Increasing Inertia Weight For Structural Damage Identification, Appl. Sci. 12 (2022). Https://Doi.Org/10.3390/App12073429.

[10] J. Eberhart, Ressell C.; Kennedy, Particle Swarm Optimization, In: Proc. IEEE Int. Conf. Neural Networks, 1995: Pp. 1942–1948. Https://Ci.Nii.Ac.Jp/Naid/40021910174/.

[11] M. Suri, Raunaq Singh; Dubey, Vikrant; Kapoor, Nishant Raj; Kumar, Aman; Bhushan, Optimizing The Compressive Strength Of Concrete With Altered Compositions Using Hybrid PSO-ANN, In: Int. Conf. Inf. Syst. Manag. Sci., Springer, 2023: Pp. 163–173.

[12] Z. Lin, H. Yanwen, X. Jie, F. Xiong, L. Qiaomin, W. Ruchuan, Trust Evaluation Model Based On PSO And LSTM For Huge Information Environments, Chinese J. Electron. 30 (2021) 92–101. Https://Doi.Org/10.1049/Cje.2020.12.005.

[13] S. Zhao, W. Xu, L. Chen, The Modeling And Products Prediction For Biomass Oxidative Pyrolysis Based On PSO-ANN Method: An Artificial Intelligence Algorithm Approach, Fuel. 312 (2022). Https://Doi.Org/10.1016/J.Fuel.2021.122966.

[14] Y. Shi, R. Eberhart, Modified Particle Swarm Optimizer, Proc. IEEE Conf. Evol. Comput. ICEC. (1998) 69–73. Https://Doi.Org/10.1109/Icec.1998.699146.

[15] R.C. Eberhart, Y. Shi, Tracking And Optimizing Dynamic Systems With Particle Swarms, Proc. IEEE Conf. Evol. Comput. ICEC. 1 (2001) 94–100. Https://Doi.Org/10.1109/Cec.2001.934376.

[16] J. Xin, G. Chen, Y. Hai, A Particle Swarm Optimizer With Multi-Stage Linearly-Decreasing Inertia Weight, Proc. 2009 Int. Jt. Conf. Comput. Sci. Optim. CSO 2009. 1 (2009) 505–508. Https://Doi.Org/10.1109/CSO.2009.420.

[17] M.S. Arumugam, M.V.C. Rao, On The Performance Of The Particle Swarm Optimization Algorithm With Various Inertia Weight Variants For Computing Optimal Control Of A Class Of Hybrid Systems, Discret. Dyn. Nat. Soc. 2006 (2006) 1–17. Https://Doi.Org/10.1155/DDNS/2006/79295.

[18] M. Nikabadi, A.; Ebadzadeh, Particle Swarm Optimization Algorithms With Adaptive Inertia Weight: A Survey Of The State Of The Art And A Novel Method, In: IEEE J. Evol. Comput., 2008.

[19] W. Al-Hassan, M.B. Fayek, S.I. Shaheen, PSOSA: An Optimized Particle Swarm Technique For Solving The Urban Planning

Problem, 2006 Int. Conf. Comput. Eng. Syst. ICCES'06. (2006) 401–405. Https://Doi.Org/10.1109/ICCES.2006.320481.
[20] C. Guimin, H. Xinbo, J. Jianyuan, M. Zhengfeng, Natural Exponential Inertia Weight Strategy In Particle Swarm Optimization, Proc. World Congr. Intell. Control Autom. 1 (2006) 3672–3675. Https://Doi.Org/10.1109/WCICA.2006.1713055.
[21] Y. Feng, G.F. Teng, A.X. Wang, Y.M. Yao, Chaotic Inertia Weight In Particle Swarm Optimization, In: Second Int. Conf. Innov. Comput. Inf. Control. ICICIC 2007, 2007. Https://Doi.Org/10.1109/ICICIC.2007.209.
[22] R.F. Malik, T. A Rahman, S.Z.M. Hashim, R. Ngah, New Particle Swarm Optimizer With Sigmoid Increasing Inertia Weight, Int. J. Comput. Sci. Secur. IJCSS. 1 (2007) 35–44.
[23] K. Kentzoglanakis, M. Poole, Particle Swarm Optimization With An Oscillating Inertia Weight, Proc. 11th Annu. Genet. Evol. Comput. Conf. GECCO-2009. (2009) 1749–1750. Https://Doi.Org/10.1145/1569901.1570140.
[24] L. Gao, Yue-Lin; Xiao-Hui, An; Jun-Min, A Particle Swarm Optimization Algorithm With Logarithm Decreasing Inertia Weight And Chaos Mutation, In: 2008 Int. Conf. Comput. Intell. Secur., 2008.
[25] H.R. Li, Y.L. Gao, Particle Swarm Optimization Algorithm With Exponent Decreasing Inertia Weight And Stochastic Mutation, 2009 2nd Int. Conf. Inf. Comput. Sci. ICIC 2009. 1 (2009) 66–69. Https://Doi.Org/10.1109/ICIC.2009.24.
[26] J. Zhang, J. Sheng, J. Lu, L. Shen, UCPSO: A Uniform Initialized Particle Swarm Optimization Algorithm With Cosine Inertia Weight, Comput. Intell. Neurosci. 2021 (2021). Https://Doi.Org/10.1155/2021/8819333.
[27] A.T. Kiani, M.F. Nadeem, A. Ahmed, I.A. Khan, H.I. Alkhammash, I.A. Sajjad, B. Hussain, An Improved Particle Swarm Optimization With Chaotic Inertia Weight And Acceleration Coefficients For Optimal Extraction Of PV Models Parameters, Energies. 14 (2021). Https://Doi.Org/10.3390/En14112980.
[28] A. Agrawal, S. Tripathi, Particle Swarm Optimization With Adaptive Inertia Weight Based On Cumulative Binomial Probability, Evol. Intell. 14 (2021) 305–313. Https://Doi.Org/10.1007/S12065-018-0188-7.
[29] M. Asif, A.A. Nagra, M. Bin Ahmad, K. Masood, Feature Selection Empowered By Self-Inertia Weight Adaptive Particle Swarm Optimization For Text Classification, Appl. Artif. Intell. 36 (2022). Https://Doi.Org/10.1080/08839514.2021.2004345.
[30] Z. Kuang, Z. Ma, Z. Li, X. Deng, Cooperative Computation Offloading And Resource Allocation For Delay Minimization In Mobile Edge Computing ☆, J. Syst. Archit. 118 (2021) 102167. Https://Doi.Org/10.1016/J.Sysarc.2021.102167.
[31] X. Deng, Z. Sun, D. Li, J. Luo, S. Wan, User-Centric Computation Offloading For Edge Computing, IEEE Internet Things J. 8 (2021) 12559–12568. Https://Doi.Org/10.1109/JIOT.2021.3057694.
[32] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z.H. Abbas, S. Khaf, A Deep Learning Approach For Energy Efficient Computational Offloading In Mobile Edge Computing, IEEE Access. 7 (2019) 149623–149633. Https://Doi.Org/10.1109/ACCESS.2019.2947053.
[33] S.U.R. Malik, H. Akram, S.S. Gill, H. Pervaiz, H. Malik, EFFORT: Energy Efficient Framework For Offload Communication In Mobile Cloud Computing, Softw. - Pract. Exp. 51 (2021) 1896–1909. Https://Doi.Org/10.1002/Spe.2850.
[34] R. Aldmour, S. Yousef, T. Baker, E. Benkhelifa, An Approach For Offloading In Mobile Cloud Computing To Optimize Power Consumption And Processing Time, Sustain. Comput. Informatics Syst. 31 (2021) 100562. Https://Doi.Org/10.1016/J.Suscom.2021.100562.
[35] R. Aldmour, S. Yousef, M. Yaghi, G. Kapogiannis, MECCA Offloading Cloud Model Over Wireless Interfaces For Optimal Power Reduction And Processing Time, 2017 IEEE Smartworld Ubiquitous Intell. Comput. Adv. Trust. Comput. Scalable Comput. Commun. Cloud Big Data Comput. Internet People Smart City Innov. Smartworld/SCALCOM/UIC/ATC/Cbdcom/IOP/SCI 2017 - Conf. Proc. 2021 (2018) 1–8. Https://Doi.Org/10.1109/UIC-ATC.2017.8397639.
[36] Q. Zhu, B. Si, F. Yang, Y. Ma, Task Offloading Decision In Fog Computing System, (2017) 59–68.
[37] E.F. Coutinho, F.R. De Carvalho Sousa, P.A.L. Rego, D.G. Gomes, J.N. De Souza, Elasticity In Cloud Computing: A Survey, Ann. Des Telecommun. Telecommun. 70 (2015) 289–309. Https://Doi.Org/10.1007/S12243-014-0450-7.
[38] M.S. Aslanpour, S.S. Gill, Q. Mary, A.N. Toosi, Performance Evaluation Metrics For Cloud , Fog And Edge Computing : A Review , Taxonomy , Benchmarks And Standards For Future Research, Internet Of Things. (2020). Https://Doi.Org/10.1016/J.Iot.2020.100273.
[39] A. Ahmad, O. Omar, Energy Consumption In Internet Of Things (IOT), Int. J. Sci. Eng. Res. 7 (2019) 3–11.
[40] S. Singh, I. Chana, M. Singh, R. Buyya, SOCCER: Self-Optimization Of Energy-Efficient Cloud Resources, Cluster Comput. 19 (2016) 1787–1800. Https://Doi.Org/10.1007/S10586-016-0623-4.
[41] Q. You, B. Tang, Efficient Task Offloading Using Particle Swarm Optimization Algorithm In Edge Computing For Industrial Internet Of Things, J. Cloud Comput. 10 (2021). Https://Doi.Org/10.1186/S13677-021-00256-4.
[42] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, A Survey On The Computation Offloading Approaches In Mobile Edge Computing: A Machine Learning-Based Perspective, Comput. Networks. 182 (2020). Https://Doi.Org/10.1016/J.Comnet.2020.107496.
[43] S. Singh, P. Garraghan, R. Buyya, ROUTER: Fog Enabled Cloud Based Intelligent Resource Management Approach For Smart Home Iot Devices, J. Syst. Softw. 154 (2019) 125–138. Https://Doi.Org/10.1016/J.Jss.2019.04.058.
[44] S.H.H. Madni, M.S.A. Latiff, Y. Coulibaly, S.M. Abdulhamid, Recent Advancements In Resource Allocation Techniques For Cloud Computing Environment: A Systematic Review, Cluster Comput. 20 (2017) 2489–2533. Https://Doi.Org/10.1007/S10586-016-0684-4.
[45] S.S. Gill, I. Chana, M. Singh, R. Buyya, Chopper: An Intelligent Qos-Aware Autonomic Resource Management Approach For Cloud Computing, Cluster Comput. 21 (2018) 1203–1241. Https://Doi.Org/10.1007/S10586-017-1040-Z.