Analysis of Optimal Task Scheduling Using Optimization Algorithms in Cloud-Edge Computing

K. Vinothkumar[1], Dr. D. Maruthanayagam[2]

[1]Research Scholar, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India

[2]Head/Professor, PG and Research Department of Computer Science, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India

Abstract: ***Edge-cloud computing involves deploying a set of edge servers near mobile devices so that these devices can schedule tasks to the servers with low latency. One fundamental and critical problem in edge-cloud systems is determining how to dispatch and schedule tasks in such a way that the task response time (defined as the interval between the release of a task and the arrival of the computation result at its device) is minimized. Edge computing (EC), which distributes resources to the network edge, is gaining traction in applications requiring low latency and high reliability. Nowadays, EC provides resources in a decentralized manner; a large number of cloud-based services are deployed on the network's edge, as processing data at the edge can reduce costs. This paper presents an optimised scheduling algorithm based on the Lion Optimization Algorithm (LOA), PSO (Particle Swarm Optimization), Firefly Algorithm (FF), and Ant Colony Optimization (ACO) to improve cloud edge task scheduling. To achieve a better result in terms of reducing energy consumption and increasing the lifespan of the cloud-edge system after task completion (LOA). The simulation results demonstrate the efficacy of the comparison by employing research parameters such as Task delay, Task Completion Time, Execution Time, Average Make span, Energy consumption, Average response time, and Energy Latency.***

Keywords: Edge Cloud Computing, Lion Optimization Algorithm, Particle Swarm Optimization, Firefly Algorithm, Ant Colony Optimization, Optimization Algorithms, Energy Consumption, Average Makespan

1. Introduction

Edge computing is a new technology that performs data analytics and storage close to the data source (i.e., mobile devices) to reduce network latency. This computing paradigm has piqued the interest of academics and industry alike. Edge computing, due to its characteristics of being on the network's edge and close to mobile terminal equipment, can reduce data transmission delay and improve real-time performance of local businesses [2]. Edge computing is one of the potential solutions to these problems. Edge computing, in particular, is a distributed computing paradigm that enables computation and data storage to be brought as close to the relevant data sources as possible. Edge computing technology has been proposed to meet the requirements of low latency and reduced bandwidth consumption [3]. Edge-cloud data centres are a subset of remote cloud data centres that have fewer computational capabilities, slower processing speeds and less memory capacity than remote cloud data centres [4]. Mobile devices can offload tasks to edge clouds and receive computing results with low network latency. Even so, since these specially deployed servers are on a smaller scale than remote cloud data centres, the resource and computation capabilities of edge-clouds are relatively limited when compared to remote cloud data centres. Whatever given edge-cloud may not be able to support a wide range of mobile applications. As a result, edge-clouds are typically backed up by a remote cloud via the Internet, allowing them to offload some of their more demanding jobs to the remote cloud. Connecting multiple edge-clouds (e.g., via a metropolitan area network) in order to improve mobile user services by sharing and balancing workloads among the able to participate edge-clouds [5–7]. Edge computing brings the power of cloud data centres to the network's edge, bringing it closer to IoT devices. Although edge nodes have more resources than IoT devices, they fall short of cloud data centres. Many researchers are currently combining cloud data centres and edge nodes to fully leverage each other's advantages and compensate for their respective disadvantages. Hierarchical computing can be realised in the cloud and edge coexistence system, in which tasks can be processed opportunistically by both the edge node and the cloud server. Non-computationally intensive tasks, for example, can be processed at the edge node to achieve lower end-to-end latency and better energy efficiency. On the other hand, it is preferable to offload computationally intensive tasks to the cloud server in order to take advantage of its vast computing capacity. As a result, effective collaboration between cloud computing and edge computing is critical for performance enhancement. [8]. Resource allocation is important in edge computing for assigning tasks (generated by local devices) to the remote cloud (the network's central) or local servers/devices (the edge of the network). A common approach for resource management in edge computing is to assign tasks to remote cloud or local servers based on factors such as energy, bandwidth consumption, and latency. Based on their goals, these methods can be divided into three categories [9].

- Reducing energy consumption
- Improving system throughput
- Reducing task completion time

There's really edge computing technology, that also sends the service to be processed to a server closer to the user terminal, solving the problem that the terminal's processing power is insufficient, the hardware condition is poor, and the distance to the cloud is too long, resulting in an excessive delay response. Completing a business calculation requires communication and computing resources from the edge computing server throughout the process of applying edge computing technology. As a result, optimizing the computational resource allocation strategy can reduce the time and system loss required in the calculation process. The problem of optimising cloud edge computing resource allocation was solved by using mutual iteration of the lion optimization algorithm and the firefly algorithm, reducing calculation energy consumption and delay.

2. Optimization Algorithms

2.1 Lion Optimization Algorithm:

LOA is inspired by the nature of lions. Lions are known for their high levels of cooperation and aggression. Lions are divided into two social groups: resident lions (RL) and nomad lions (NL) (NL). Pride groups are common among RL. Whereas the second type moves out infrequently, either in pairs or individually. Lions hunt in packs, and several lionesses work together to encircle the prey from various angles in order to capture it. Male lions and a few female lionesses rest and await the hunter lionesses. The lions can mate at any time, and a lioness can mate with multiple partners. Lions use their urine to mark their territory. The lion's behaviours are mathematically defined in LOA in order to model the optimization algorithm. A collection of arbitrarily produced solutions known as lions generates the initial population in LOA. Some members of the initial community (percent N) are chosen as NL, while the remainder of the population (RL) is randomly divided into P subsets (prides). S represents the number of females, and the remaining lions are males. The best-attained solution in the previous iterations for each individual lion is known as the best-visited location, and it is updated during the optimization process. The lion's behaviours are mathematically defined in LOA in order to model the optimization algorithm. A collection of arbitrarily produced solutions known as lions generates the initial population in LOA. Some members of the initial community (percent N) are chosen as NL, while the remainder of the population (RL) is randomly divided into P subsets (prides). S represents the number of females, and the remaining lions are males. The best-attained solution in the previous iterations for each individual lion is known as the best-visited location, and this is revised during the optimisation problem. Every pride removes the young males from the group, and it becomes NL when they reach maturity and their strength is lower than the resident males. Furthermore, an NL (both male and female) moves at random in search of a better location (solution). In line with this, some resident females migrate from one pride to another or change their lifestyle to become NL and vice versa. The weakest lion dies or is killed due to a variety of factors, including a lack of food and competition. In the current study, the above procedure was repeated until the termination criterion was met. The various phases of the LOA are described below [10].

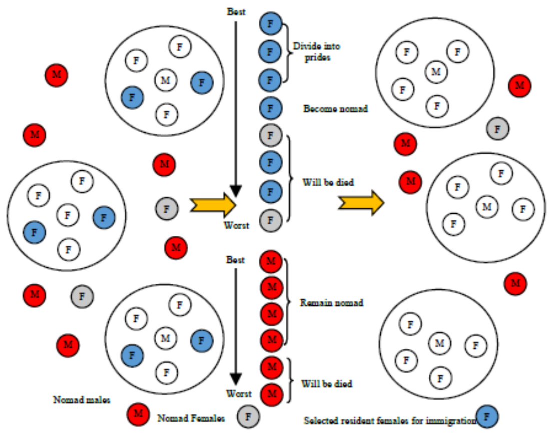

Figure 1: Nature of LOA

Lion Optimization process:
The lion algorithm's process is as follows [11]:
- Create random populations
- Create prides and lions.
- One lion particle = Choose a female lion at random for hunting. Each female lion chooses the best spot in the pride. The weakest lion pride is expelled from the population and becomes a nomad. Each pride assesses the rate of immigration and becomes a nomad.
- Evaluate the fitness function to select the best females and fill the empty places which of the female lions which are migrated from the territory

```
Algorithm:
Input: Different set of factors
$(As_{i},.Ck_{i},.Po_{i},.C_{i}$
$.Q_{i}\}.\{\sigma_1.\sigma_2.R.\beta_2.D_r.v_3\}.\{e_1,\omega_1\}.\{m,k,l,\gamma_g.F.G.E\}$
Output: Optimal set of factors for edge devices
$i \leftarrow 1 : j \leftarrow 1 : St \leftarrow 0 : Dt \leftarrow 0$;
while($i<N$)
Compute$_h$ Mop,. $S_i$ and $D_i$
        $i \leftarrow i+1$:
        While ($j \le N$)
        $Dt \leftarrow Dt^{-1} + D_\gamma$.
        $St \leftarrow St^{-1} + S_r$
        Compute $T_w.T_{it}g_{ar}.C.Cr$ and $\tau_{high}$;
        Initialize the number of lions (edge devices) and
         iterations:
          Create an atbitrazy result for every lion: Assign
         pride and nomad lions; while (tr< max iteration)
         for (every pride)

 A few females are chosen at random to hunt;

 The rest of the females travel toward the most excellent
 sites in the region:

 Every male roam in %R of the region;

 %Ma of female's note with only one inhabitant males: The
 weakest male is rejected by pride and tutus into a nomad:
for (every nomad lion)

Males and females travel arbitrary distances in exploring
space:
%Ma of female's mate with only one male:
Nomad males hit pride;
```

```
        for (every pride)
    % I of females immiigrate from pride and tum into nomads:
    Each nomad lion gender is ranked based on its objective range:

    Most excellent females are chosen and circulated to pride. satisfying empty sites:

    Nomad lions with the minimum objective content will be
    Rejected depending on the highest allowed quantity of every gender;
        end for
            end for
             end for
                end while
obtain the optimal set of $T_{QP}$,$T_{EC}$,$T_{CE}$, $E_i^s$ , $E_{i,k}^c$ ,$T_{high}$ for edge devices;
cloud decides which tasks will be carried out at the edge devices;
            end while
        end while
end
```

Initialization: Initially, the population is generated arbitrarily across the solution space. Each outcome is known as a lion (edge devices). A lion (edge devices) is denoted in a d-dimensional optimization dilemma, i.e., d set of determining factors, as:

$$\text{Lion (edge devices)} = [l_{1,,,,,,}l_d] \tag{1}$$

The fitness range of every edge device (lion) is determined by assessing the objective function given in (16) as:

$$f\text{ (edge device)} = f(l_{1,,,,,,,,,,}l_d) \tag{2}$$

In the initial stage, d_{pop} solutions are generated arbitrarily in exploring space, and a percentage d of completed results is arbitrarily chosen as migrant edge devices. The remaining population is divided in to the pride at random. Evey solution had a specific gender and remained stable throughout the optimization task. Each lion observes its most excellent entered site while searching. Each pride's region is created based on such observed sites. As a result, for each pride, observed sites (the best entered sites) generate that pride's region via its representatives [12].

Migration : Motivated by the lion swap life and migratory behaviour in nature, when one lion travels from one pride to another or switches its lifestyle and resident female becomes nomad and vice versa, it increases the diversity of the target pride by its position in the previous pride. The lion's migration and switch lifestyle, on the other hand, create a bridge for exchange of information. The maximum number of females in each pride is determined by S% of the pride's population. Some females were chosen at random to be nomads by migration operators. The number of migrated females in each pride is equal to the number of surplus females in each pride multiplied by the maximum number of females in a pride. Once selected females migrate from prides and become nomads, they are separated into new nomad females and old nomad females based on their fitness. The best females from among the m are then chosen at random and distributed to prides to fill the migratory females' empty spots. This procedure maintains the diversity of the whole population and share information among prides [13].

Movements towards Safety: Only a few female lions hunt for prey, while the rest stay in safe territory. The best positions for each territory are calculated and saved. A high victory rate indicates that the lions have strayed from the optimal point. Lower values indicate that lions are roaming for improvement, and thus competition evaluation indicates success.

$$\text{Female Lion}' = \text{Female lion} + 2D \times \text{rand}(0,1)\{R1\} + U(-1,1) \times \tan(\theta) \text{ X } D \tag{3}$$
$$\times \{R2\}\ \{R1\}.\{R2\} - \quad 0,\|[R2]\| -1$$

where Female Lion is the female lion's current position, and D represents the distance between the female lion's current position and the chosen point chosen through tournament selection within the pride's territory. R1 is a vector whose initial point is the female lion's previous location and whose direction is toward the chosen position. R2 is perpendicular to R1.

Hunting: Some females in each pride look for prey in a group to provide food for their pride. These hunters use specific strategies to encircle and capture their prey. When hunting, lions generally followed similar patterns. Stander classified the lions into seven stalking roles, classifying them as Left Wing, Center Wing, and Right Wing. During hunting, each lioness adjusts her claim based on her own[14].

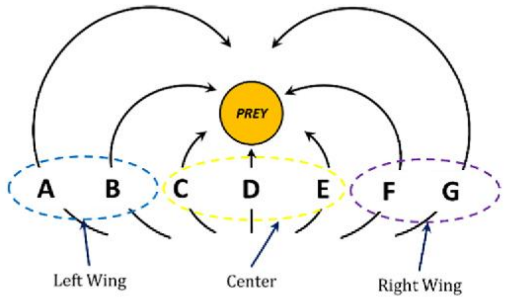

Figure 2: Hunt Process

```
Divide hunters into n subgroups randomly generate a prey
For i=1 to hunters (H)
        Move→ ith Hunter
If(i_pos>last_pos)
        Hunt=1
Else
        Hunt=0
End for
```

Mating: Mating includes two primary steps known as crossover and mutation, which are found to be significant operators for any evolutionary optimization. We follow the maximum natural littering rate, i.e., four cubs (mostly) in a lioness pregnancy, and thus we get four cubs 'X cubs' from the crossover, which is uniform in nature with random crossover probability Cr. The crossover operation can be represented mathematically as follows:

$$X^{male} \quad + X^{\;cubs} \quad (p) = Bp^{o} \;\; B\; p\; 0\; X^{female} \; {}^{:}p\; 1,2,3,4 \qquad (4)$$

where B is the one's complement of B, vector operator" represents Hadamard product or schur product, and Xcubs(p) is the pth cub obtained from crossover The Xcubs are subjected to uniform mutation with mutation probability as Mr, resulting in an equal number of new cubs Xnew. The Xnew (from mutation) and Xcubs (from crossover) are placed in the cub pool and processed further. A secondary step known as gender clustering is also included in this procedure to extract a single male and female creature from the cub stream.Based on the lion's physical nature, we select the cubs, which have the first and second best fitness, as the male cub *X m_ cub* and female cub *X f _ cub*, respectively. Once the *X m_ cub* and *X f _ cub* are obtained, set their ages (commonly referred as cubs' age) *A cub* as zero.

***Defense*:** Lions value this behaviour. Mature male lions engage in combat with other lions. Losers either become nomads or flee the territory. When nomadic lions win a battle, they take over the territory of the loser. Thus, LOA defends lions in two ways: against newly matured resident males and nomadic males. LOA thus finds the strongest lion in the group[15].

Termination Criteria

When any of following two termination criteria is met, the algorithm execution is terminated; otherwise, the process is repeated from equation5, after storing X male and f.

$$(\text{X male}).f\,(X\ male) \leq e\ \text{T} \qquad (5\ )$$

$$N\,f > N\,f^{\,max} \qquad (6)$$

Nfmax and eT are the maximum number of function evaluations and target error, respectively.

2.2 Firefly Algorithm:

The firefly algorithm (FA) is a met heuristic algorithm inspired by the flashing behaviour of fireflies. The Firefly Algorithm (FA) is a population-based technique that investigates firefly foraging behaviour to find the global optimal solution based on swarm intelligence. The primary function of the firefly's flash is to attract other fireflies. Fireflies use their flashing signal to attract mating partners and prey, as well as to share food with others. The Firefly Algorithm generates a random initial population of viable candidate solutions. All fireflies in the population are handled in the solution search space with the goal of collectively sharing knowledge among fireflies to guide the search to the best location in the search space. Every particle in the inhabitants is a firefly which it moves in a search domain with a regularly updated attractiveness based just on firefly's and its neighbours' knowledge [16].

The standard Firefly algorithm is based on firefly flashing patterns, which act as a signal system to attract other fireflies. They use the flash light to attract mates or prey, and the following assumptions are made to execute the firefly algorithm:

1) A firefly is concerned about another firefly.
2) The attraction between two fireflies is proportional to their brightness and the distance between them. As a result, the brighter firefly can attract nearby fireflies, and if none are brighter than the others, their movement will be random.
3) When the brightness of both fireflies is equal, the fireflies move at random [17].Algorithm:

```
Create and initialize N firefly particles
Determine the light intensity for each firefly
Determine the distance between each toe fireflies
Repeat
        For j=1;N
If(Ii<Ij) move firefly I towards firefly j end if
Update the attractiveness with distance r by exp[-γr]
        Evaluate the new solution and update light intensity
        End for j
End for i
Rank the fireflies and find the current global best
Until Termination condition is met
```

Fireflies will be drawn to other fireflies that are brighter. Light intensity can be interpreted as an objective function with a value corresponding to the observed problem's objective function. The light intensity is proportional to the value of the objective function in the maximisation problem, but inversely proportional in the minimization problem. The Firefly algorithm's goal in scheduling is to minimize makes span so that the intensity of the light is inversely proportional to makes span. Solutions with a short life expectancy will have brighter light intensity, attracting solutions with a longer life expectancy [18].

The firefly algorithm is an efficient met heuristic that enhances the standard firefly algorithm with a new position-based mapping method and a linear movement scheme. To be more specific, we first use a novel position-based mapping method to convert a firefly to a corresponding solution, which is regarded as a task permutation, with the currently best solution in mind. In addition, unlike the traditional movement strategy [19], we build a linear movement strategy that reduces the computational burden when updating the firefly positions, allowing us to speed up the searching procedure.

The FF algorithm is developed based on these rules to obtain the optimal solution. This

Algorithm comprises four important steps:

Step 1: The FF population is randomly initialised and consists of a set solution in this step.
Throughout our algorithm, the population consists of fog devices or cloud servers.
Step 2: The distance between any two FF I and j at xi and xj is calculated as follows:

$$V=\|x_i - x_j\| = \sqrt{\sum_{k=1}^{D}(xi - xj,k)2} \qquad (7)$$

Here, *D* is the optimization parameter, which is equivalent to the number of computational tasks in this study.

Step 3: As the distance increases, the attractiveness of the FF, which is the objective function of our algorithm, decreases exponentially. At a distance v, Equation (8) gives the FF's attractiveness.

$$\beta(v_{i,j}) = \beta_{0e}^{-\gamma 2ij} \quad (8)$$

Where is the FF's brightness at distance v, and 0 is the FF's brightness at initial attractiveness when v = 0. The theoretical value of the light absorption coefficient is the variance of attractiveness, and its value influences the speed of algorithm convergence. Almost all of the time, the values of 2[0.01, 100].

Step 4: The FF attraction and randomization walkthrough Levy flights in this step. The FF's movement is calculated using distance and attractiveness as follows:

$$x_{i =} \ x_i + \ \beta_0 e\text{-}\gamma r^2 ij(x_j - x_i) \ + \ \alpha \in_i \quad (9)$$

Where I is a random number vector derived from a Gaussian or uniform distribution, and α is the randomization variable. In our algorithm, the FF movement is equivalent to the task movement to the fog or cloud servers [20].

Light intensity and attractiveness of firefly:
The Firefly algorithm is founded on two key concepts: variation in light intensity and desirability formulation. For the sake of simplicity, it is assumed that the attractiveness of the firefly is determined by its bright light, which is linked to the objective function [21].

A firefly's brightness I can be chosen as I(x) f(x) for a maximisation problem at a specific location x.

Because attractiveness is relative, it should be judged by other fireflies, and it will vary with distance rij between firefly I and firefly j. As previously stated, light intensity decreases with distance from its source, and light is also absorbed by air, so attractiveness should be allowed to vary with varying degrees of absorption. So, in its most basic form, light intensity I(r) follows the inverse square law.

$$I(r) = \frac{Is}{r2} \quad (10)$$

Is represents the intensity at the source. The light intensity I varies with distance r, with a constant light absorption coefficient, i.e. Where Io is the initial light intensity, the combined effect of the inverse square law and absorption can be approximated as the following Gaussian form to avoid the singularity at r = 0 in the expression Is/r2.:

$$I = I0e\text{-}\gamma r^2 \quad (11)$$

The desirability of a firefly is proportional to the light intensity seen by adjacent fireflies, which can be expressed as:

$$\beta \quad = \quad \beta 0e^{-\gamma r2} \quad (12)$$

where β0 is the attractiveness at r = 0. Since it is often faster to calculate1/(1 + r2) than an exponential function, the above function, if necessary, can be approximated as shown in (13).

$$\beta = \frac{\beta 0}{(1+\gamma r2)} \quad (13)$$

In the real-time implementation, β (r) is the attractiveness function, which can be any monotonically decreasing function, such as the one shown below.

$$\beta(r) = \beta 0e^{-\gamma rm} \quad (m\geq 1) \quad (14)$$

For a fixed, the characteristic length becomes

$$r= \ \gamma\text{-}^{1/m} \rightarrow 1,m \rightarrow \infty \quad (15)$$

Conversely, γ can be used as typical initial value for a specific length scale Γ in an optimization problem. That is

$$\frac{1}{r^m} \quad (16)$$

The distance between any two fireflies is calculated using Cartesian distance method

$$r_{i,j} \ = ||x_i - x_j|| = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k}} \quad (17)$$

in(10) xi,k is the kth component of spatial coordinate xi of ith firefly. In 2-D case, we have

$$r_{i,j} \ = \sqrt{(X_i - X_j)^2 - (y_i - y_j)2} \quad (18)$$

Firefly i is attracted to brighter firefly j and its movement is determined by

$$x_i = x_{i\ +} \beta 0e\text{-}\gamma r^2_{i,j}(x_j - x_i) + \alpha \in_i \quad (19)$$

Firefly movement: If firefly j is brighter than firefly I the firefly I is attracted and moved to it. The firefly I to firefly j movement can be described as

$$xi = xi + \beta 0 \exp(-\gamma r\ 2\ i\ j)(xj - xi) + \alpha(rand - 0.5) \quad (20)$$

The first term in (12) is the current position of a firefly, the second term is the firefly's attractiveness to light intensity seen by neighbour fireflies, and the third term is the firefly's random movement when there is no brighter firefly. The coefficient is a randomizatiosn parameter with the value [0,1], and rand is a random number with the value [0,1]

The movement of a firefly *i* which is attracted by a more attractive i.e., brighter firefly *j* is given by the following equation [2, 3, 20]:

$$x_i = x_i + \beta 0 * \exp(-\gamma r^2_{ij}) * (x_j - x_i) + a * (rand - 1/2) \quad (21)$$

Where the first term refers to a firefly's current position, the second term refers to a firefly's attractiveness to light intensity seen by adjacent fireflies, and the third term refers to.

In the absence of brighter fireflies, this is used to generate random movement. The coefficient is a randomization parameter determined by the problem of interest, whereas rand is a uniformly distributed random number generator in the space [0,1]. As we will see in this algorithm implementation, we will use 0 = 1.0, [0, 1], and the attractiveness or absorption coefficient =1.0, which ensures that the algorithm quickly converges to the Ideal solution.

The brightness and attractiveness of firefly: When comparing the brightness of two fireflies, the locations of the fireflies must be taken into account. As the distance between two fireflies grows, their attractiveness and brightness decrease dramatically. Furthermore, if a firefly does not encounter another firefly in its immediate vicinity, it will fly in an unknown direction. The algorithm compares the appeal of the new and old firefly positions. If the new position has a higher attractiveness value than the current position, the firefly is moved to it; otherwise, the firefly remains in its current position. The termination criterion of the FA is based on an arbitrary number of iterations or a predetermined fitness value.

According to the following equation, the brightest firefly moves at random: Each firefly has a value that indicates how well it can attract other fireflies in the swarm. As shown by the comparison, the attractiveness will diverge with its distance factor dij at the locations Xi and Xj, between the two corresponding fireflies I and j.

$$dij = |Xi - Xj| \quad (22)$$

The attractiveness function β of the firefly is computed as:

$$\beta = \beta 0\ e{-}\gamma\ r2 \quad (23)$$

where $\beta 0$ is attractiveness at r=0 and is coefficient of light absorption.

The movement of the less bright firefly toward the brighter firefly is computed by

$$Xi=Xi+\beta 0\ e{-}\gamma r2ij\ (Xj-Xi) + \alpha\ (r\ and-1/2) \quad (24)$$

where α is the randomization parameter and rand is a randomly selected number in the interval [0, 1].

2.3 Ant Colony Optimization:

Ants interact by leaving pheromone trails, so where ants go within and around their ant colony is a stigmergic system. Numerous ant species deposit a substance called pheromone on the ground as they walk from or to a food source. Other ants can detect this pheromone, and its presence influences their path-finding behavior, as they tend to follow high pheromone concentrations. The pheromone deposited on the ground creates a pheromone trail, enabling the ants to locate great food sources that have previously been recognized by other ant.[22]

$$p_{i,j} = \frac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum_{h\in s}[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}} \quad (25)$$

Where is all the pheromone and is the reciprocal of the distance between the two nodes. This probability determines ant k at node I to choose node j with the highest p[i][j].The Ant System ACO algorithm was the first to be proposed. Its main feature is that all the ants who have completed the tour keep updating the pheromone values. This same pheromone update for ij, that is, for the edge connecting cities I and j, is as follows: [23]

$$\tau_{i,j}(t) = (1-\rho).\tau_{i,j}(t+1) + \sum_{i=1}^{n}\Delta\tau_{ij}^{k} \quad (26)$$

Where ρ is the evaporation rate, m is the number of ants, and Δτkij is the quantity of pheromone per unit length laid on edge (i,j) by the kth ant

$\Delta\tau_{ij}^{k}$ ={ Q/L_k if ant k used edge (i,j) in its tour

0 otherwise (27)

Where Q is a constant and Lk is the tour length of the kth ant. We choose =1 and =1 to solve our assignment probability. 0, the initial amount of pheromone is also equal to1. Assuming that there are no pheromone evaporations is equal to 1 [24].

Every ant in an ant colony optimization system is a computation agent. Iteratively creates the optimal solution to the issue. Solution nations are the solutions obtained at intermediate states. Every ant in each generation begins to move from state I to state 'j' in search of a locally optimal solution. As just a result, in each iteration, each ant uses probability to compute a set of feasible solutions to its current state and moves to one of the optimal solution states. The probability of P_{ijk} moving to state j from state I for every ant k is generally determined by two parameters: the coefficient of vaporisation phenomenon or pheromone concentration denoted by ij indicating the previous move and visibility of the move denoted by ij indicating the past worth of the move. In each iteration, these values are updated to find the best optimal solution. To achieve an optimal solution, their value may increase or decrease in each iteration based on their effect. In general, Equation gives the probability of ant k moving from state I to state j (28).

$$P_{ij}^{k} = \frac{(\tau_{iz}^{\alpha})(\eta_{iz}^{\beta})}{\sum VM\_\varepsilon listallowedk(\tau_{iz}^{\alpha})(\eta_{iz}^{\beta})} \quad (28)$$

- •τij denotes the pheromone concentration for the transition from state I to state j.
- 0≤ α is a parameter that controls the pheromone concentration's influence.
- ηij represents the desirability of transitioning from state I to state j. It is computed using priori information, i.e., a priori knowledge, typically 1/dij, where d is the distance).
- β1 is a parameter that controls the effect of ηij.
- VM_list allowed, where allowed kVM_list is a list of VMs allowed by ant k.
- The overall result of this update is that when one ant discovers a path to food sourced from the colony, the remaining ants are likely to follow that path, and the positive feedback of previous ants who have already discovered the path eventually leads to all ants following the shortest path [25].

Algorithm:

Input: An instance p of a CO problem model p = (S,f,Ω)

```
    intializePheromoneValues (T)
  $S_{bs} \leftarrow$ null
  While termination conditions not met do
  $\mathfrak{S}_{iter} \leftarrow \emptyset$
  For j=1,…,$n_a$ do
  S ← ConstructSolution (T)
  If S is a valid solution then
  S ← localsearch(S)  {optional}
  If (f(S) < f($S_{bs}$)) or ($S_{bs}$ = null) then $S_{bs} \leftarrow$ S
  $\mathfrak{S}_{iter} \leftarrow \mathfrak{S}_{iter} \cup$ { S }
  End if
  End for
  Applypheromoneupdate(T, $\mathfrak{S}_{iter}$, $S_{bs}$ )
  End while
```

Output: the best so far solution S_{bs}

ACO is inspired by the foraging behavior of ant colonies. Indeed, real ants have inspired many researchers [26], and the ants approach has been used by many researchers for problem solving in a variety of fields. This strategy is known as ACO after its inspiration. The ants collaborate to find new food sources while also using existing food sources to shift food back to the nest.

The movements of these ants independently update a solution set. There are two types of ant traversal in this system.

1) Forward movements-Ants move forward to extract food or search for food sources.
2) Backward movements-In this type of movement, ants traverse back to the nest after gathering food from food sources.

The ACO is a unique algorithm for several reasons, including the fact that the optimum solution is built not by a single entity but by several entities that traverse the length and breadth of the network and then build upon a solution individually. Many researchers [27] have improved upon the ACO's pheromone updating phenomenon in order to improve on the results. It has been used by researchers to improve various tasks such as task scheduling and optimizations in cloud edge computing [28].

Every ant in a generation constructs a solution step by step by making several probabilistic choices. In general, ants that find a good solution mark their paths through the decision space by sprinkling pheromone on the path's edges. The pheromone attracts the next generation of ants, causing them to search the solution space near previous good solutions. Aside from pheromone values, ants are usually guided by some problem-specific heuristic when evaluating trial solutions. To use ACO to solve a Cost-Performance Trade-off Problem (CPTOP), the problem must first be transformed into a Travel Salesman Problem (TSP). Furthermore, the cost and performance objectives should be merged into a single optimization problem [29].

2.4 Particle Swarm Optimization

Kennedy and Eberhart [30] proposed particle swarm optimization (PSO) as a population-based stochastic optimization algorithm. The technique is used to solve optimization problems by mimicking the social behavior of bird flocks, fish schools, and other animal societies that cooperate and share information to improve their position without relying on a leader. In this technique, a population of individuals, which are politician running alternatives defined as particles, move in a provided solution space based on their current position Xik and current velocity Vi k for the kth iteration. Depending on the optimization problem, the quality of each particle is measured using a predefined fitness function. The movement of each particle is determined by its best known personal position pBesti, as well as the best known global position gBesti for the entire swarm. This process guides the swarm to the best position after several iterations of the search process. The velocity and position of the particle are described further below:

$$v_i^{k+1} = \omega V_i^k + c1r1(pBest_i - X_i^k) + c_2r_2(gBest_i - X_i^k) \quad (29)$$

$$X_i^{k+1} = X_i^k + ViK \quad (30)$$

Where ω is the inertia weight, r1 and r2 are random numbers between (0,1) and c1 and c2 are the learning factors.[31].

Algorithm:

```
Initialize Population
for t=1 : maximum generation
   for i=1 : population size
         if $f(x_{i,d}(t)) < f(p_i(t))$ then $p_i(t) = x_{i,d}(t)$
         $f(p_g(t)) = \min(f(p_t(t)))$
         end
         for d=1 : dimension
            $\nu_{i,d}(t+1) = \omega \nu_{i,d}(t) + c_1r_1(p_i - x_{i,d}(t)) + c_2r_2(p_g - x_{i,d}(t))$
            $x_{i,d}(t+1) = x_{i,d}(t) + \nu_{i,d}(t+1)$
            if $\nu_{i,d}(t+1) > \nu_{max}$ then $\nu_{i,d}(t+1) = \nu_{max}$
            else if $\nu_{i,d}(t+1) > \nu_{max}$ then $\nu_{i,d}(t+1) = \nu_{max}$
            else if $\nu_{i,d}(t+1) < \nu_{min}$ then $\nu_{i,d}(t+1) = \nu_{min}$
 end
if $x_{i,d}(t+1) < x_{max}$ then $x_{i,d}(t+1) = x_{max}$
else if $x_{i,d}(t+1) < x_{min}$ then $x_{i,d}(t+1) = x_{min}$
            end
         end
      end
end
```

In PSO, a swarm of q particles is defined first. A candidate solution is represented by a particle i(i 1,..., q). The positions of the particles xik move over time, or more precisely the algorithm's iteration steps k, according to their velocity of displacement vik in equation (26) and the displacement rule in equation (26) (27). These particles attempt to follow the population's best leader gbest (which has the best solution) and improve their own best results pbest. If a particle outperforms the leader, it becomes the population's new gbest leader. The operation is repeated until the number of iterations reaches a certain limit kmax or a certain appropriate situation has been managed to reach.
(31)

$$x_i^{k+1} = x_i^k + \upsilon_i^{k+1} \quad (32)$$

Consider n tasks to assign to m cloud resources. The expression Sp represents a solution (29). This expression is written as a vector of dimension n. The n elements of the vector represent the tasks, and the value g of a j-th element represents the resource Rj that will process the task, as shown in equation (29):

$S_p = \{T_1^{Ra}, T_2^{Rm},,,,,,,, T_j^{Rg},,,,,,T_n^{Rf}\} = [a,m,,,,,,g,,,,,,f]$ (33)

In our approach, we consider that this solution S_p is equivalent to the position x_i of the particle i of the PSO. x_i is represented by a vector of size n. A particle move with a v_i velocity. v_i is also represented by a vector of size n[32].

Self-organization in swarms through three basic ingredients as follows.

1) Strong dynamical nonlinearity (often involving positive and negative feedback): positive feedback promotes the formation of convenient structures, whereas negative

feedback counterbalances positive feedback and aids in the stabilisation of the collective pattern.
2) Exploration and exploitation balance: SI identifies an appropriate balance to provide a valuable mean artificial creativity approach.
3) Numerous interactions: swarm agents use information from neighbouring agents to publish awareness all through the network [33].

PSO algorithm can thus be summarized as follows: PSO algorithm is a swarm-based search process in which each individual is called a particle defined as a potential solution of the optimised problem in D dimensional search space, and it can memorise the optimal position of the swarm and its own, as well as the velocity. The particle information is combined in each generation to adjust the velocity of each dimension, which is used to compute the particle's new position. Particles in the multidimensional search space constantly change their states until they reach balance or optimality, or go beyond the computation limits. The objective functions introduce a unique connection between the different dimensions of the problem space. Many empirical evidences have demonstrated that this algorithm is a useful optimization tool.

The Improvement of Particle Swarm Optimization Algorithm

Inertia weights

An inertia weight ω is a proportional agent that is related to the previous time's speed, as well as the method for changing the speed is as follows:

$$\upsilon_{id}^{k+1} = \omega v_{id}^{k} + c_1 r_1^{k} (pbest_{id}^{k} - x_{id}^{k}) + c_2 r_2^{k} (gbest_{d}^{k} - x_{id}^{k}) \quad (34)$$

Inertia weights can be used to control the influence of the previous speed on the current speed. The larger is, the greater the PSO's searching ability for the whole, and the smaller is, the greater the PSO's searching ability for the partial. In general, is equal to 1, so there is a lack of searching ability for the partial at later stages of the several generations. According to the experimental results, PSO has the fastest convergence rate when is between 0.8 and 1.2. During the experiment, is restricted from 0.9 to 0.4 according to the linear decrease, which causes PSO to search for a larger space at the start and quickly locate the position to the most idealist solution. As ω is decreasing, the particle's speed will also slow down in order to find the delicate partial. The method accelerates convergence and improves the PSO function. Whenever the problem to be solved is very complex, this method causes PSO's searching ability again for entire at a later period after several generations to be insufficient, and the most optimist solution cannot be found, so inertia weights can be used to solve the problem [34].

To execute task scheduling in the cloud edge, Particle Swarm Optimization (PSO) is used. Because customers' demands must be met to the greatest extent possible in the cloud environment, selecting a strategy for task scheduling of workflow is critical [35]. Workflow is a computational model of a working process that represents the logics and rules that govern how to organise the front and back ends of a working process and calculates it in a computer using the appropriate model [36]. The main issue that the workflow should address is that it should do everything possible to achieve the goal of a service and transmit information, resources, or tasks automatically according to specific rules using computers between a large numbers of participants [37].

3. Experimental Results

Inside this test, we used CloudSim 3.0 to implement the algorithms for tasks that need to be processed in the cloud, by adding the bindCloudletToVM method in the Datacenter Broker class; the LOA algorithm based on the catastrophe PSO, ACO, FF algorithm is added to carry out the simulation experiment. Data such as resource computing power and task calculations are derived from MATLAB data that has been randomly generated. We select a different number of tasks, and the experimental data from various iteration times is analyzed and compared with the time-based algorithm (LOA) and simple ACO, FF, PSO algorithms under another data circumstances. (LOA) task scheduling algorithm that minimizes the completion time.

Moreover, our simulations were performed on a PC with Intel Core i5-7400 processor @ 3.0 GHz CPU and 8 GB of RAM.

In the laboratory, cloud computing, edge computing, and user terminal environments were built, and the connection between the actual edge and the cloud data centre was simulated using bandwidth controllable network equipment, with the end environment being a local area network of computer equipment components connected to the edge environment. Figure 3 depicts the complete experimental verification environment.

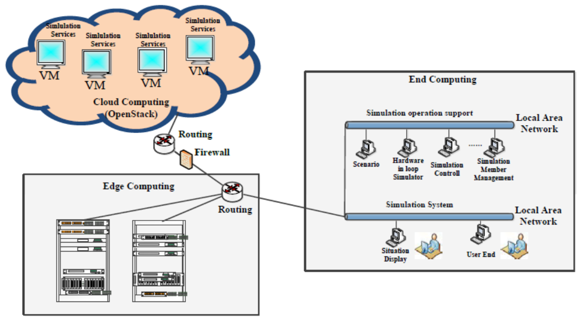

Figure 3: Experiments Environment

Task delay:

Figure 4 depicts the task delays under various offered loads in order to investigate the performance limits of various scheduling schemes. Each result represents a 107-slot simulation run. Because the optimal throughput region is defined as the set of arrival rate vectors in which queue lengths and thus delays remain finite, we can **consider the traffic load as the boundary of the ideal throughput region. Because LOA has a considerably higher throughput, despite the fact that the end-to-end delay appears to be similar between FIREFLY and LOA. LOA is a vast improvement over FF, ACO, and PSO.** A comparison of the delay rates in edge computing task scheduling is calculated via the following formula.

Delay rate =∈ turnRoundTime – ∈ length Xlength /∈ length 100%, Where turnRoundTime is the turnaround time of the tasks and length is the length of the tasks.

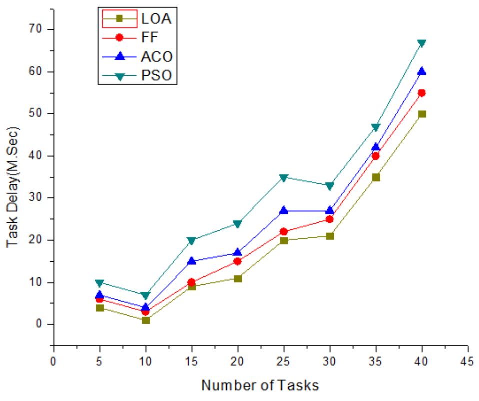

Figure 4: Comparison of Task Delay

Task Completion Time:
The completion time is the time difference between the starting and finishing times. The graph below depicts the task completion time as well as the energy consumption. Figure 5 depicts the iterative process of combining ACO, PSO, Firefly, and LOA to achieve the best total time and lowest total cost. In terms of completion time, the LOA algorithm outperforms the ACO, PSO, and Firefly algorithms. The eight tasks arrived four times at 0, 220, 440, and 660 Figure 4 shows that when the task volume is small, the completion time of each algorithm is not significantly different. that the four algorithms have the same completion time in the first iteration, but as the number of iterations increases, the convergence rate and accuracy outperform the edge cloud scheduling results of LOA, ACO, PSO, and Firefly. As the task volume increased, so did the completion time of each algorithm. LOA outperforms the second fastest **FF, PSO, ACO algorithm by 6.6 %faster than the second fastest FF, PSO, and ACO algorithm.**

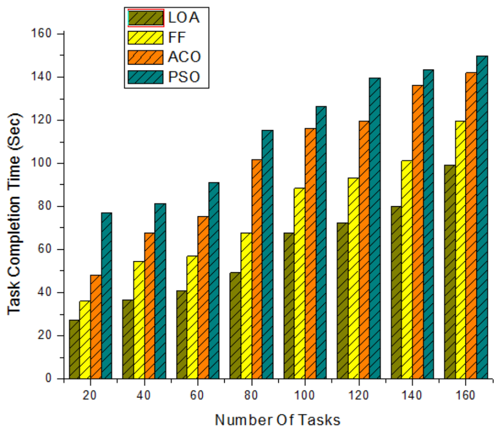

Figure 5: Task Completion Time

Execution Time:
TASK requests with a growing number of cloud servers and a fixed number of edge servers and tasks. It demonstrates that the execution times of requests decrease as the number of cloud servers increases. This is due to the task requests being distributed across multiple cloud servers. However, when comparing execution times with increasing cloud servers (Figure 5) to increasing edge servers (Figure 6), there is little difference, despite the fact that the processing capability of the cloud servers is greater than that of the edge servers. This is because all requests will first be routed to an edge-server, which will determine whether the request should be executed on the edge or sent to the cloud. Thus, increasing the number of cloud servers has no significant impact because requests must wait in the edge-decision server's queue, resulting in an increase in execution time. It is clear from the results that as user requests increase, so does the time required for resource allocation. The time required to allocate resources is determined by resource availability as well as the distance between the resources and the client (request place). Figure 5 depicts the iterative process of combining ACO, PSO, Firefly, and LOA to achieve the best completion time and total cost. In terms of execution time, the **LOA algorithm beats the ACO, PSO, and Firefly algorithms**.

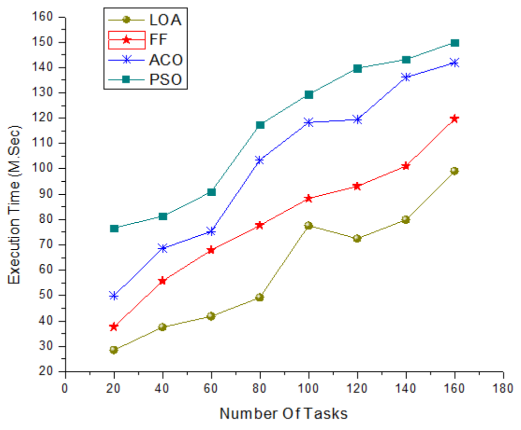

Figure 6: Execution Time

Average Make span:
The graph compares the make span to time. The results demonstrate that **the proposed LOA algorithm outperforms PSO, ACO, and FF.** The following experiments compared the average make span with various tasks set. Figure 7 depicts the average make pan of the LOA, PSO, ACO, and FF algorithms. It can be seen that as the number of tasks increases, **LOA takes less time than FF, PSO, and ACO algorithms. This implies that the LOA algorithm outperforms the ACO, PSO, and FF** algorithms.

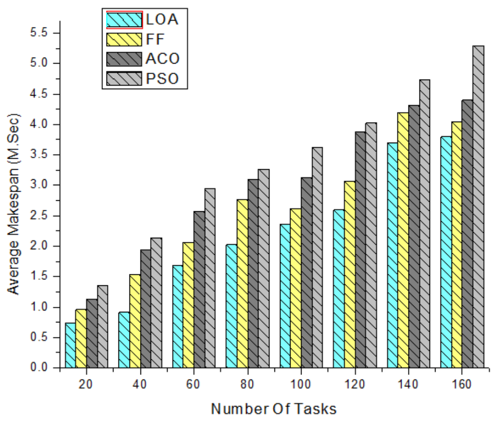

Figure 7: Average Makespan

Energy consumption:
It is an obvious feasibility of the ACOBF exhibition in terms of meeting the person's time constraints. Even though previously stated, duties sent to the Cloud are expected to be independent of one another. The results explain the ACO, PSO, FF, and LOA algorithms. When a comparable number of task units/tasks are received by the Cloud, make span and energy increase dramatically, **whereas in the case of LOA, make span and energy either reduce or vary.** This one is due to the algorithm's ability to preserve the convergence achieved by having the starting point close to the minimum. Figure 8 depicts the LOA algorithm's behaviour in comparison to previous schemes for an average energy consumption scenario with a variable number of nodes. According to the findings, the LOA algorithm significantly reduces network energy usage when compared to traditional methods. Just a subset of tasks is held accountable for achieving balanced energy consumption.

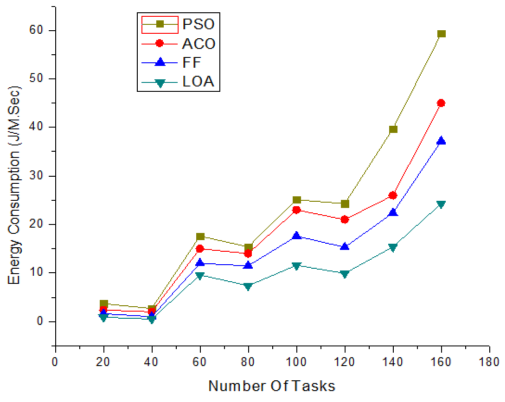

Figure 8: Energy Consumption

Average response time:
Figure 9 depicts the average reaction time results. Because the RR algorithm was specifically designed for time sharing, it outperforms all other approaches in terms of average response time.

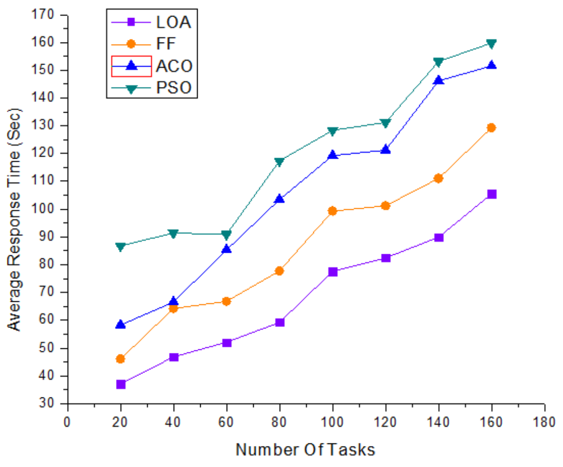

Figure 9: Average Response Time

Using a fair resource allocation strategy ensures that each service request has a set time frame. If the service processing **time exceeds the system's time quantum,** the **sample is removed from the average response time calculation.** For instance, if the previous service request had a lengthy task schedule, the average response time would increase. Because the proposed LOA algorithm sequentially inserts tasks into the queue, its average response time is slightly longer than that of FF, PSO, and ACO. Although the priority algorithm can sort tasks based on the priority of service requests, low-priority tasks are easily delayed, increasing the average response time.

Energy Latency:
It is the time it takes the edge nodes to complete the tasks. In terms of latency, Figure 10 compares the efficiency of ACO and LOA-based task execution optimization. When compared to the ACO, PSO, and FF algorithms, **LOA-based task execution optimization in a cloud-edge computing system achieves less latency.**

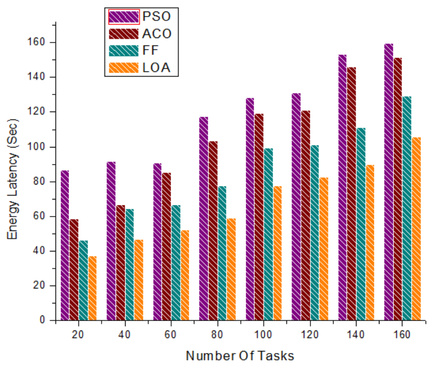

Figure 10: Energy Latency

4. Conclusion

To develop task scheduling techniques in cloud edge computing, various optimization algorithms have been used. The PSO, FF, ACO, and LOA algorithms were used to develop a task-scheduling algorithm for cloud edge computing in this paper. The cloud server can select which tasks will be performed at the edge devices, according to this optimization solution. Finally, **the results demonstrated that the proposed algorithm is more efficient.** Using the

Lion Optimization Algorithm to reduce energy consumption and make span of the cloud-edge system after task completion time. Experiments show that the existing **ACO, PSO, LOA, and FF algorithms** satisfy the requirements of users in task scheduling in cloud edge computing effectively.

References

[1] M. Satyanarayanan, "the emergence of edge computing," computer, vol. 50, no. 1, pp. 30 39, 2017.

[2] Shida lu1, rongbin gu1, hui jin2, liang wang1, xin li 2, (member, ieee), and jing li,"qos-aware task scheduling in cloud-edge environment" 2021.

[3] Sapienza, M.; Guardo, E.; Cavallo, M.; La Torre, G.; Leombruno, G.; Tomarchio, O. Solving critical events through mobile edge computing: An approach for smart cities. In Proceedings of the 2016 IEEE International Conference on Smart Computing (SMARTCOMP), St Louis, MO, USA, 18–20 May 2016; pp. 1–5.

[4] Lin, Yezhi.; An analytic computation-driven algorithm for Decentralized Multicore Systems. Future Gener. Comput. Syst. 2019, 96, 101–110.

[5] M. Jia et al., "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," in INFOCOM 2016.

[6] Z. Xu et al., "Efficient algorithms for capacitated cloudlet placements," IEEE Trans. on Parallel and Distributed Systems, vol. 27, no. 10, 2016.

[7] Capacitated cloudlet placements in wireless metropolitan area networks," in Local Computer Networks (LCN) 2012.

[8] Jinke Ren, Guanding Yu, Yinghui He, and Geoffrey Ye Li, Collaborative Cloud and Edge Computing for Latency Minimization", IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, 2019

[9] HealthEdge: Haoyu Wang∗, Jiaqi Gong†, Yan Zhuang∗, Haiying Shen∗, John Lach, "Task Scheduling for Edge Computing with Health Emergency and Human Behavior Consideration in Smart Home",

[10] Karuppaiah Geetha, Veerasamy Anitha, "An evolutionary lion optimization algorithm-based image compression technique for biomedical applications", 2019, https://doi.org/10.1111/exsy.12508

[11] K. Kaur and Y. Kumar, "Swarm intelligence and its applications towards various computing: A systematic review," in Int. Conf. on Intelligent Engineering and Management (ICIEM), IEEE, vol. 2013,no. 7, pp. 1–6, 2020.

[12] S. Saranya and 2N. Sabiyath Fatima,"Efficient Handling of Medical Data Classification in Cloud-Edge Network using Optimization Algorithm", Journal of Computer Science, 2021, 17 (11): 1116.1127

[13] Maziar Yazdani, FariborzJolai, "Lion Optimization Algorithm(LOA): A nature-inspired metaheuristic algorithm", 2015, Journal of Computational Design and Engineering,

[14] K. Thenmozhi, S. Udhaya, n. Vinothini, V. Nagaraju,"an adaptive wsn clustering scheme using lion optimization algorithm to maintain coverage area in wireless sensor network", international journal on recent researches in science, engineering & technology (ijrrset)

[15] S.Silambarasan, m. Savitha devi," enhanced lion swarm optimization algorithm with centralized authentication approach for secured data transmission over wsn, ictact journal on communication technology, september 2021, volume: 12, issue: 03, issn: 2229-6948(online)

[16] Adil Yousif Aboalgassim Alfaki,"Job Scheduling Approaches Based On Firefly Algorithm For Computational Grid", 2013

[17] Ibrahim Ahmed Saleh, Omar Ibrahim Alsaif Sundus Abduttalib Muhamed, Essa Ibrahim Essa,"Task Scheduling for cloud computing Based on Firefly Algorithm,: journal of physics,2019

[18] Debbie Kemala Sari, Sumiharni Batubara, Raden Roro Sindyastuti S," Scheduling design for Job Shop Production Using Firefly Algorithm to Minimize Mean Tardiness,"Proceedings of the International Conference on Industrial Engineering and Operations Management Dubai, UAE, March10-12,2020

[19] R. SundarRajan, V. Vasudevan, and S. Mithya, "Workflow scheduling in cloud computing environment using firefly algorithm," in International Conference on Electrical, Electronics, and Optimization Techniques, 2016, pp. 955–960.

[20] Rabab Farouk Abdel-Kader, Noha Emad El-SayadID*, Rawya Yehia Rizk,"Efficient energy and completion time for dependent task computation offloading algorithm in industry 4.0", PLOS ONE, 2021

[21] Sankalap Arora, Satvir Singh "The Firefly Optimization Algorithm: Convergence Analysis and Parameter Selection", International Journal of Computer Applications (0975 – 8887), Volume 69– No.3, May 2013

[22] An Introduction to Ant Colony Optimization",Marco Dorigo and Krzysztof Socha IRIDIA, Universit´e Libre de Bruxelles, CP 194/6, Av. Franklin D. Roosevelt 50, 1050 Brussels, Belgium http://iridia.ulb.ac.beApril 30, 2007,pp.6-7.

[23] M. Dorigo, L M Gambardella "Ant colonies for the traveling salesman problem". BioSystems, 1997 [11] "An Ant Colony Optimization Algorithm for Solving Travelling Salesman Problem", Krishna H. Hingrajiya, Ravindra Kumar Gupta, Gajendra Singh Chandel

[24] D.Prasanth Rao, Mr.E.Madhukar," A compartive study between Hungarian and Antcolony optimization algrorithm for task scheduling in the Cloud, *International Journal Of Modern Engineering Research*, Volume 11, Issue 2, February-2020 ISSN 2229-5518

[25] Sambit Kumar Mishra, Bibhudatta Sahoo and P. Satya Manikyam, "Adaptive Scheduling of Cloud Tasks Using Ant Colony Optimization",

[26] M. Dorigo, V. Maniezzo and A. Colorni, Ant System: Optimization by a Colony of Cooperating Agents, IEEE Transactions on Systems, Man, and Cybernetics, PP. 29-41, 1996.

[27] C.W. Chiang, Y.C. Lee, C.N. Lee and T.Y. Chou, Ant Colony Optimization for Task Matching and Scheduling, IEE Proceedings on Computers and Digital Techniques, 153 (6), pp. 373-380, 2006.

[28] J. Sun, S. Xiong and F.M. Guo, A New Pheromone Updating Strategy In Ant Colony Optimization, Proceedings of the International Conference on Machine Learning and Cybernetics, pp. 620-625, 2004

[29] Mingzhang Wu, Janne Koljonen and Timo Mantere," Addressing Resource Allocation Issues in Cloud Computing Environment with Ant Colony Optimization, Proceedings ISBN 978–952-5183-54-2,

[30] J. Kennedy and R. Eberhart, "Particle swarm optimization in: Neural networks," in Proceedings IEEE International Conference on 1995, 1942, pp. 1942–1948.

[31] Dineshan Subramoney, Clement Nyirenda, "PSO-Based Workflow Scheduling: A Comparative Evaluation of Cloud and Cloud-Fog Environments,

[32] Multi-Criteria Aware Task Scheduling Algorithm Based on PSO For Fog Computing, ISSN: 2509-0119. © 2021 International Journals of Sciences and High Technologies, Vol. 30 No. 1 December 2021, pp.205-219

[33] Yudong Zhang,1 Shuihua Wang,1,2 and Genlin Ji1,"A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications", Hindawi Publishing Corporation Mathematical Problems in Engineering Volume 2015, Article ID 931256, 38 pages

[34] Qinghai Bai, "Analysis of Particle Swarm Optimization Algorithm", journal of computer and information science, vol.3, no.1, 2010

[35] Y. Fang, F. Wang, J. Ge. "A task scheduling algorithm based on load balancing in cloud computing", Web Information Systems and Mining. Springer Berlin Heidelberg, **(2010)**, pp.271 277.

[36] M. Xu, L. Cui, H. Wang, Y. Bi. "A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing", Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications, **(2009)** August 10-12; Chengdu, China.

[37] A Vouk M. "Cloud computing–issues, research and implementations", CIT. Journal of Computing and Information Technology, vol. 16, no. 4, **(2008)**, pp. 235-246.

Author Profile

K. Vinothkumar received his **M.Phil** Degree from Periyar University, Salem in the year 2015. He has received his **M.Sc.,** Degree from Hindusthan College of Arts and Science, Coimbatore affiliated to Bharathiar University, Coimbatore in 2013. He is pursuing his **Ph.D** Degree (Part-Time) in Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India. He is working as **HOD Cum Assistant Professor** in Department of Computer Science at Kavitha's College of Arts and Science (Co-Ed.,), Vaiyappamalai, Tiruchengode, Namakkal. His current research of interests includes Edge Computing, Cloud Computing, Mobile Computing and Ad hoc Networks.

Dr. D. Maruthanayagam received his **Ph.D** Degree from Manonmaniam Sundaranar University, Tirunelveli in the year 2014. He received his **M.Phil** Degree from Bharathidasan University, Trichy in the year 2005. He received his **M.C.A** Degree from Madras University, Chennai in the year 2000. He is working as **HOD Cum Professor**, PG and Research Department of Computer Science, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India. He has above **22 years** of experience in academic field. He has published **7 books**, more than **60 papers** in International Journals and more than **30 papers** in National & International Conferences so far. His areas of interest include Computer Networks, Grid Computing, Cloud Computing and Mobile Computing.