SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V12I8P118

Volume 12 Issue 8, 199-209, August 2025
© 2025 Seventh Sense Research Group®

Original Article

Intelligent Resource Scheduling in Mobile Edge Clouds Using Adaptive Queueing and Meta-heuristics

B.Teja Sree[1], G.P.S.Varma[2], Indukuri Hemalatha[3]

[1,2]*Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, A.P., India.*
[3]*Department of Information Technology, S.R.K.R. Engineering College, A.P., India.*

[1]*Corresponding Author : tejasree9479@gmail.com*

Received: 10 June 2025 Revised: 11 July 2025 Accepted: 12 August 2025 Published: 30 August 2025

Abstract *- The key aspect of Mobile Edge Computing (MEC) that has aroused considerable concern is how to efficiently schedule tasks and allocate resources to meet real-time applications' low latency and energy requirements. Classical scheduling algorithms such as First-Come-First-Served (FCFS), Shortest Job First (SJF), and Round Robin (RR) are largely ineffective for managing dynamic workloads and heterogeneous resource environments. This paper proposes A delay energy-efficient queuing-based Particle Swarm Optimization (DAEE-QPSO) algorithm that combines queuing theory and swarm intelligence to optimize task offloading decisions. Individual nodes of edges are represented by M/M/1 queues that allow the queuing delay and system congestion to be estimated perfectly. A hybrid fitness function is constructed that attempts to concurrently reduce the runtime and energy usage of tasks and constraints (meeting task deadlines and energy budgets). The proposed DAEE-QPSO algorithm shows the lowest average run time and energy consumption compared to FCFS and EADF-PSO. It also has high task throughput and maintains the greatest average energy saving of up to 29.8%, outperforming Energy Aware Double Fitness Function PSO (EADF-PSO) by only 23.3%. This result demonstrates its effectiveness in improving task efficiency in a mobile edge cloud. The suggested architecture is very applicable to energy-conscious and real-time usage in the cloud-to-edge environment.*

Keywords *- Edge computing, Mobile Edge computing, Task scheduling, Delay Awareness, Queuing theory, Cloud computing, Resource Allocation.*

1. Introduction

Cloud computing revolutionizes the digital world by enabling the availability of computer resources like memory, processor speed, and network capacity on demand through collaborative use of configurable resources. This model supports real-time applications, workflows, and elastic services due to its scalability and efficient resource management offered through flexible and cost-effective schemes [1]. Task scheduling and resource allocation are crucial for system performance because they directly impact efficiency, energy consumption, service latency, and customer satisfaction. Well-designed scheduling strategies ensure that computational tasks are allocated optimally across Virtual Machines (VMs), achieving the highest throughput and the lowest makespan [2]. Standard rule-based and fixed algorithms are not adaptable to the dynamic workloads and changing resource levels typical in real-time cloud environments [3]. Such strategies tend to fail under high loads or heterogeneous task environments, resulting in longer execution times and inefficient resource utilisation. To address these issues, researchers have recently turned to metaheuristic algorithms inspired by natural processes to improve task scheduling and resource allocation solutions [4]. They include such local search techniques as Genetic Algorithms (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO) that have been successful because they can escape local optima to offer near-optimal solutions to NP-hard scheduling problems [5]. The PSO has become a common method among them because it is simple, converges quickly and is efficient at searching a large solution space. Nonetheless, standard PSO has high chances of premature convergence and poor search-space exploitation, especially on complex, multi-objective problems such as cloud task scheduling [6]. To address these drawbacks, hybrid metaheuristic algorithms have been suggested, whereby the strengths of multiple optimisation strategies can be used in order to maximize exploration and exploitation abilities. As an example, by hybridizing PSO with local search strategies, mutation, or some other evolutionary algorithm can assist in striking the balance between convergence rate and quality of solutions [7]. In the proposed research, by using a hybrid PSO-based task scheduling framework, a solution is sought to achieve, which would optimise the use of resources and minimise energy usage, but would do so with a low makespan and execution time. The approach will seek to improve upon traditional scheduling methodologies and earlier metaheuristic approaches by incorporating problem-specific heuristics, extending the ability of PSO to adapt to the problem.

The evaluation indices, such as metrics energy consumed and CPU used, show a considerable improvement in experimental results with benchmark cloud datasets. With the ongoing development of cloud computing, an increasing demand arises for intelligent scheduling algorithms that can respond to or adapt to different workloads and infrastructure limitations. Using hybrid metaheuristics in task scheduling is a promising approach to improve the efficiency, scalability, and sustainability of managing cloud resources.

Addressing the energy characteristics of edge networks is crucial. Previous research has developed methods to enhance energy utilization at edge nodes [8-10] or create energy-efficient hardware support equipment. Edge node synchronisation via workload scheduling and assigning resources algorithms, which may improve their energy efficiency, has not gotten nearly enough attention. The traditional algorithms, like FCFS, RR and SJF, do not adjust to the changing workloads and resource limitations, which leads to ineffective throughput, increased execution time and increased energy.

Although having promising results in handling NP-hard scheduling problems, metaheuristic methods such as PSO, GA, and ACO have their serious shortcomings. Standard PSO includes the problem of early convergence and does not include the opportunities to consider queuing delays, which are imminent when real-time MEC environments are considered. In addition, the existing hybrid techniques tend to optimise either with the execution time or energy consumption separately, ignoring the paradigm of concomitant delay and energy-aware task scheduling. This leaves a research gap: the task scheduling solutions do not offer scalability and energy efficiency at the same time without failing to meet SLAs. Entering into this, one needs a queuing-theory-based, energy-delay-optimised scheduling framework.

To this point, this paper introduces a new algorithm of task scheduling, which is referred to as DAEE-QPSO or the effective capability of combining the ideas of the queuing theory with the PSO optimization framework to enhance task scheduling within a Mobile Edge Computing (MEC) system. Unlike traditional PSO-based algorithms that only look at energy use or execution time, the DAEE-QPSO algorithm uses a dual-objective fitness approach, meaning it assesses both energy consumption and delays caused by queuing. The algorithm does a good job of predicting because it treats the edge servers like M/M/1 queues, which helps it manage the workload across different resources effectively. This hybridisation helps DAEE-QPSO better optimise energy savings, response times, and throughput in resource-constrained, real-time MEC environments with considerably better results than before. In summary, the proposed DAEE-QPSO makes three main contributions:

- Introduced M/M/1 queuing theory into the PSO framework to accurately model resource congestion and waiting times, enabling delay-sensitive task allocation in MEC.
- Developed a hybrid fitness function that simultaneously reduces energy consumption and delay, resulting in improved efficiency for resource-constrained mobile devices.
- Designed a flexible scheduling model that dynamically adapts to workload changes and resource availability, ensuring high performance across varying task volumes and heterogeneous edge-cloud systems.

2. Related works

Over the years, the research has primarily focused on the offloading of applications from user devices to edge devices. In [11], the authors have suggested a bioinspired adaptive resource scheduling algorithm that can guarantee Quality of Service (QoS) in an MEC environment. The mechanism deploys an approach based on swarm intelligence in an optimized Ant Colony Optimization (ACO) algorithm to assign resources among edge nodes dynamically. Various QoS parameters considered in this model include latency, bandwidth utilization and energy consumption, and with that, the system will be able to respond in real time based on the changing network requirements and workloads. Its adaptive quality in the algorithm aids in alleviating congestion and enhances load balancing between heterogeneous edge devices. Benchmark simulations based on performance evaluation show that the offered solution is much superior to the conventional scheduling approaches in its response time reduction and guarantees of quality of service. This bioinspired scheduler promises to be used in real-world smart cities and IoT applications, where responsiveness to errors and edge-wallet resources streamlines efficient resource utilization.

In [12], the authors were directed toward the new Genetic Algorithm with Skew Mutation (GA-SM), proposing to use it to optimize the unrelated resource-aware task offloading. The suggested solution solves the problems related to a wide range of computing capacities, networking bandwidths, and latencies of edge and cloud nodes. In contrast to the classic genetic algorithm, GA-SM incorporates another mutation operator (a skew mutation operator) that keeps the population diversity and prevents premature convergence to improve the quality of solutions. Its algorithm causes the execution of tasks and resource utilization to be optimized simultaneously, task by task, with smart edge/cloud-based offloading according to real-time limitations and differences among devices. The results from the GA-SM simulations show that it performs much better than the basic GA, PSO, and ACO in reducing makespan, improving energy efficiency, and increasing the success rate of offloading. It is very suitable for smart cities and industrial IoT real-time applications, regardless of how high the task arrival rate or how unstable the network conditions are.

In [13], the authors explained a scheduling system for Mobile Crowd Computing (MCC) that takes into account multiple factors and available resources to handle tasks in situations where resources change a lot and are limited, using a method based on rules of thumb. The model combines important parameters of decision-making, namely device mobility, the level of energy, processing power, and urgency of the task, to ensure adequate and effective scheduling. The ranking of offered mobile devices to carry out the tasks is achieved through the customized heuristic function ranking by using a weighted multicriteria decision analysis. The algorithm focuses on maximizing the minimization of task failures and total makespan and optimally balances the use of such devices and the long-term lifetime of the system as a whole. The test results from a simulation of MCC show that the suggested scheduling method is much better in several ways, such as success rate, execution time, and energy use, compared to older scheduling methods.

In [14], the authors present a variety of strategies for task offloading demand for service delivery with low latency and power consumption in a distributed computing world. The model presented suggests a two-tier optimization scheme that synchronizes the choice between the edge nodes and the cloud servers, depending on the nature of the given tasks, their data size, the computing power required, and the time limit. Using a priority-driven scheduling implementation, the system identifies the best site to execute (edge or cloud) and dynamically allocates any computational and bandwidth resources available. Latency-sensitive utility has also been used as part of the approach to accommodate the tradeoffs of response time and resource expenses. The simulation results show that using this method leads to much less delay in completing the task and greatly increases the amount of work done compared to standard single-layer or non-adaptive offloading methods.

In [15], the authors conducted a review of resource scheduling algorithms into three types: classical, metaheuristic, and artificial intelligence (AI), and looked at the pros and cons of each when managing large and changing cloud environments. Metaheuristics methods are applied to and provide flexible solutions to NP-hard scheduling problems, as demonstrated by the authors. Moreover, applications based on AI, particularly reinforcement learning and deep learning models, can enable both predicted and independent management of resources in response to unpredictable workloads. The document discusses hybrid approaches that integrate AI and metaheuristics to balance exploration and exploitation, aiming to enhance scalability, energy efficiency, and Quality of Service (QoS).

In [16], the authors provided a modified Ant Colony Optimization (ACO) method for scheduling resources effectively by making use of a Space Information Network (SIN). This paper responds to the distinctive issues of SINs, such as high-latency links and dynamic topology, as well as the lack of edge resources, with the formulation of a resource-aware ACO model that takes the communication delay, the computational capacity, and the task priority into account for the heuristics used by the model. The suggested modification to the algorithm includes adding dynamic pheromone updates and balancing loads, which helps the algorithm make better task offloading decisions. It dynamically chooses promising paths and edge nodes to execute tasks such that the network is less latent and congested and consumes low energy. Simulation studies were conducted in real-world write-read conditions.

3. Materials and Methods

Figure 1 represents a new Delay Aware Energy-Efficient Queuing-based Particle Swarm Optimization (DAEE-QPSO) system proposed for resource scheduling in MEC settings. This is a dynamic framework that performs resource-aware job assignment, optimizing the performance of targeted key performance indicators, e.g., energy consumption, task delay, queuing time, and resource utilization across heterogeneous edge or cloud nodes.

The architecture combines queuing theory with the flexibility of PSO search capability to build better schedules and give high responsiveness to real-time situations. Every edge node will be accounted for as a single server queuing system (M/M/1), with the rate of task arrival (λ) and the rate of service tasks (μ) being observed. Based on these parameters, a computation of expected waiting time is done, which is a substitute vital element of the particle fitness function.

3.1. Task Analyzer Module

This module is the first unit of processing in an edge computing system, taking the role of interpreting and profiling incoming computational jobs. It derives important features, including the number of CPU cycles it needs, the necessary memory space, task volume, the level of priorities, energy limitations, and the deadline of execution. Using such parameters, the module will classify tasks into categories (e.g., delay-sensitive, energy-intensive) to support better scheduling decisions. This profiling helps the downstream components, e.g., schedulers and optimizers, assign suitable resources to tasks. The module will enhance decision-making quality by clarifying each task before allocating or offloading.

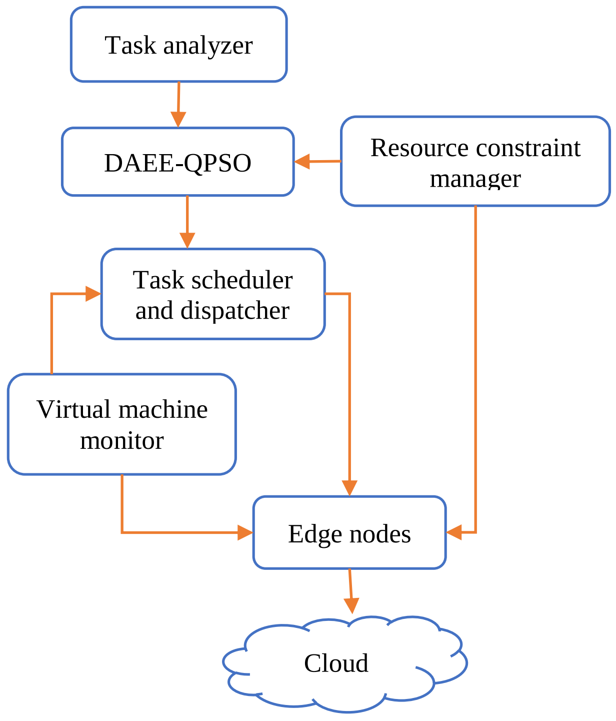

Fig. 1 Proposed framework for resource scheduling in MEC environment

3.2. Virtual Machine (VM) Monitor Module

The module is essential for real-time resource management as it constantly monitors the status and the performance of all the active virtual machines in the cloud or the edge environment. It tracks variables like CPU usage, memory availability, bandwidth usage, response times, and system loads. A resource pool at the system's center refreshes this information based on each virtual machine's current capabilities. The module also ensures that the scheduling algorithm considers real-time variability, leading to the intelligent and balanced assignment of more work. It also identifies VMs that are either over-provisioned or under-provisioned and helps prevent bottlenecks caused by performance constraints, ensuring effective resource use across the infrastructure.

3.3. Resource Constraint Manager

This module is responsible for enforcing system limitations and operational policies during task scheduling and allocation. It ensures that tasks are assigned to edge nodes only when the required computational and energy resources are available. This module evaluates constraints such as maximum CPU load, memory thresholds, energy budgets, network latency, and task deadlines. Filtering out unsuitable candidate nodes based on these criteria narrows the decision space for the scheduler and improves efficiency. The module also aids in maintaining SLAs by preventing overcommitment of resources and ensuring that critical tasks meet their performance requirements within defined system constraints.

3.4. Queuing-based Particle Swarm Optimization (QPSO) with Delay Awareness Module

This is a learning intelligent optimization algorithm whose underlying model is a combination of mathematical queuing theory models with the adaptive search ability of PSO to provide a solution to task scheduling problems generated in cloud and MEC, as shown in Figure 2. The concept of traditional PSO is based on imitating the social behavior of flocks and swarms, where each solution (particle) explores the search space by estimating its velocity and position based on its personal best and the best solution found by the entire swarm. Although PSO successfully searches vast solution spaces, it usually lacks awareness about the domain, e.g., patterns of task arrival and resource service behaviour. To solve this problem, QPSO uses queuing models, like M/M/1 or M/G/1 queues, in the fitness estimation process to consider things like resource congestion, task waiting time, and system throughput.

Each resource (e.g., a VM or an edge node) in QPSO is modelled as a queue having an arrival rate λ and a service rate μ. The arrival of tasks is allocated to the resources, and the waiting time, as well as the time of completion of the service of the tasks, are calculated with the help of the queuing formulas. These values compute more feasible scheduling intensities like makespan, turnaround time, and energy consumption.

So, the Fitness in QPSO is a compound value that combines these queuing-based parameters into a single measure of how good a particular scheduling solution is. As an example, the overall estimated delay of a given task involves the queue wait time as well as the execution time, and the cost of the usage of energy is estimated according to the processing time and the rate of resource consumption.

This queuing integration permits QPSO to dynamically avoid resource congestion, share the load, and minimise SLA violations. In addition, swarm-based exploration in each iteration ensures that a variety of solutions are explored; hence, the probability that the resource-task assignment would converge towards an optimal or near-optimal solution is high. A deadline-aware task sorting step based on the Earliest Deadline First (EDF) strategy is added to improve task allocation further before the QPSO evaluation starts. This step orders tasks by their deadlines to prioritise time-sensitive jobs in resource mapping, boosting SLA compliance and system responsiveness. The most important application of QPSO would be in such real-time, heterogeneous platforms as MEC, where both the arrival rate of tasks and their resource status vary significantly, and the response time and energy efficiency are paramount aspects of performance.

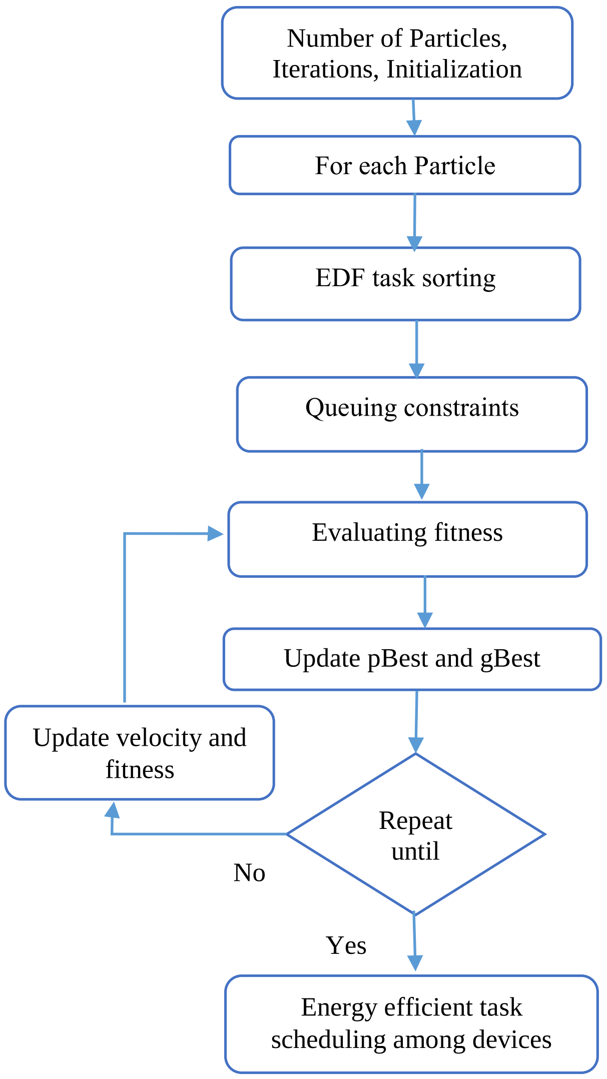
Number of Particles, Iterations, Initialization
For each Particle
EDF task sorting
Queuing constraints
Evaluating fitness
Update pBest and gBest
Update velocity and fitness
Repeat until
No
Yes
Energy efficient task scheduling among devices

Fig. 2 Flowchart of delayed awarness queuing-based PSO for energy-efficient task scheduling

Let there be a set of tasks and a set of computing resources (e.g., VMs or edge nodes):

$$T = \{t_1, t_2, \ldots\ldots, t_n\} \tag{1}$$

$$R = \{r_1, r_2, \ldots\ldots, r_n\} \tag{2}$$

Each task requires a certain amount of computational resources and arrives at a certain time. Each resource r_j is modeled as an $M/M/1$ queue, where λ_j is the arrival rate to resource r_j, and μ_j is the service rate at resource r_j. The expected waiting time W_{qj} for each task in queue at r_j is given by:

$$W_{qj} = \frac{\lambda_j}{\mu_j(\mu_j - \lambda_j)} \tag{3}$$

(assuming $\lambda_j < \mu_j$ for stability)

The completion time (turnaround) for a task t_o $_{be}$ offloaded to resource r_j is:

$$C_{ij} = W_{qj} + \frac{C_i}{\mu_j} \tag{4}$$

Where C_i is the computation size of task t_i. The energy consumed by executing t_i on r_j is:

$$E_{ij} = P_j \times \left(\frac{C_i}{\mu_j}\right) \tag{5}$$

Where P_j is the power consumption of resource r_j.

Each particle in QPSO represents a mapping X $\{x_1, x_2, \ldots, x_n\}$, where $x_i \in R$ indicates the assigned resource for task ti. The fitness function for a particle (solution) is defined as:

$$Fitness(X) = \alpha.\sum_{i=1}^{n} C_{ix_i} + \beta.\sum_{i=1}^{n} E_{ix_i} \tag{6}$$

Where α, β are weights to balance time and energy, and x_i is the resource assigned to task t_i. QPSO optimizes this function by iteratively updating positions and velocities of particles using PSO rules, while considering queuing delays and energy constraints.

To further improve SLA adherence, a deadline violation penalty can also be incorporated ny updating Equation(6) as:

$$Fitness(X) = \alpha.\sum_{i=1}^{n} C_{ix_i} + \beta.\sum_{i=1}^{n} E_{ix_i} \quad \eta + \sum_{i=1}^{n} nmax\,(0, Cixi - Di) \tag{7}$$

Where α, β are weights to balance time and energy, and x_i is the resource assigned to task ti. QPSO optimizes this function by iteratively updating positions and velocities of particles using PSO rules, while considering queuing delays and energy constraints.

Algorithm 1: Delay Awareness Energy Efficient Queuing-based PSO (DAEE-QPSO)

```
Input: $T = \{t_1, t_2, ..., t_n\}$      # Task list
  $R = \{r_1, r_2, ..., r_m\}$     # VM or edge node list
  PSO parameters: $w, c_1, c_2, max_{iter}, num_{particles}$
Initialize:
  Initialize particles with random task-to-VM assignments
  Initialize pBest[$i$] for each particle
  Initialize gBest = best(pBest)
Repeat until $max_{iter}$:
Sort tasks $T$ by ascending order of $D_i$
    TotalWq = 0
    TotalEnergy = 0
    Makespan = 0
    For each task $t_i$ in sorted $T$:
        assigned resource rj = $x_i$ (current particle assignment)
        Estimate queue length $L_r$ at $r_j$
        $\lambda_j$ = task arrival rate to $r_j$
        $\mu j$ = service rate of $r_j$
        Compute expected queuing delay using M/M/1:
            $Wq_i = \lambda_j / [\mu_j (\mu_j - \lambda_j)]$
        $ExecutionTime_i = C_i / \mu j$
        $CompletionTime_i = Wq_i + ExecutionTime_i$
        $Energy_i = Pj * ExecutionTime_i$
        Add deadline penalty if $completiontime_i > D_i$
        if $CompletionTime_i > D_i$:
            $Penalty_i = \delta * (CompletionTime_i - D_i)$
        Else:
            $Penalty_i = 0$
        TotalWq += $Wq_i$
        TotalEnergy += $Energy_i$
        Makespan = max(Makespan, $CompletionTime_i$)
    end for
    Evaluate Fitness:
        Fitness[i] = $\alpha$ * TotalWq + $\beta$ * Makespan + $\gamma$ * TotalEnergy + $\eta$ * TotalPenalty
    if Fitness[i] better than pBest[i], update pBest[i]
    if Fitness[i] better than gBest, update gBest
  end for
  for each particle i:
    Update velocity and position using PSO:
        v[i] = w*v[i] + c1*rand()*(pBest[i] - position[i]) + c2*rand()*(gBest - position[i]) position[i] = position[i] + v[i]
  end for
Return gBest mapping as optimal task-resource assignment
```

3.5. Task Scheduler and Dispatcher Module

This module is responsible for executing the scheduling plan generated by the hybrid PSO optimiser optimization engine. It maps the best tasks to virtual machines (VMs) or edge nodes and initiates the actual process of placing tasks within the selected resources. With this module, job dispatches will be carried out based on existing policies, which could be the level of priority, a deadline, and the availability of resources. It enables preemptive and non-preemptive scheduling, providing real-time and batch job processing flexibility. It is also the responsibility of the dispatcher to supervise the progress of the task being carried out and even reallocate tasks when they fail to fulfil or are running late. This module minimises latency, provides high throughput, and provides consistent delivery of services across the cloud and edge infrastructures by efficiently coordinating and executing scheduling decisions.

3.6. Working Procedure of Algorithms

Here is a structured explanation of the three components as part of an integrated framework for mobile edge computing (MEC):

3.6.1. Task Offloading Optimization for Mobile Devices

This Algorithm 2 finds out the number of tasks that different mobile devices should offload to an edge server to balance their time and energy consumption. Each device always computes the time it takes to perform a task locally compared to when the task is offloaded and the energy consumed in both scenarios. It then synthesises these into a cost function whereby they are weighed depending on predetermined preferences. The algorithm determines the optimal offloading proportion for a specific device by analysing the costs associated with different offloading ratios. When the value computed is not within the valid range, it is modified to be within the possible limit. Following this, the algorithm checks whether the chosen solution satisfies the device's time and energy constraints. In case the solution cannot be found, the algorithm will choose an alternative one, whatever best suits the requirements: the full local execution, full offloading, or a hybrid one. Finally, it displays the chosen offloading ratio for each device.

Algorithm 2: Task offloading optimization for mobile devices

Input: Set of mobile devices $\mathcal{M}$, For each device $\mu \in \mathcal{M}$: CPU cycles required κ_μ, Data size to transmit δ_μ, Device CPU frequency φ_μ, Edge server CPU frequency ψ, Uplink rate χ_μ, Local per-cycle energy η_μ, Per-bit transmission energy θ_μ, Maximum energy λ_μ, Task deadline Δ_μ, Weights α, β, γ.

Output: Optimal offloading ratio π_μ^* for each device

Procedure:

```
1:  for each device $\mu \in \mathcal{M}$ do
2:  Compute local execution time
3:      $T_\mu^{\text{local}} \leftarrow \frac{(1-\pi_\mu)\kappa_\mu}{\varphi_\mu}$
4:  Compute offloaded execution time
5:      $T_\mu^{\text{edge}} \leftarrow \frac{\pi_\mu \kappa_\mu}{\psi} + \frac{\pi_\mu \delta_\mu}{\chi_\mu}$
6:  Compute total energy consumption
7:      $E_\mu \leftarrow \eta_\mu (1-\pi_\mu)\kappa_\mu + \theta_\mu \pi_\mu \delta_\mu$
8:  Define the total cost function
9:      $f(\pi_\mu) \leftarrow \alpha T_\mu^{\text{local}} + \beta T_\mu^{\text{edge}} + \gamma E_\mu$
10:  Take the derivative and find the stationary point
11:  Compute $\frac{df}{d\pi_\mu}$, set to zero and solve for $\pi_\mu$:
12:  $\frac{df}{d\pi_\mu} = -\alpha \frac{\kappa_\mu}{\varphi_\mu} + \beta \left(\frac{\kappa_\mu}{\psi} + \frac{\delta_\mu}{\chi_\mu}\right) - \gamma \eta_\mu \kappa_\mu + \gamma \theta_\mu \delta_\mu = 0$
13:  Let the solution be $\pi_\mu^*$
14:  Clip to feasible bounds
15:      if $\pi_\mu^* < 0$, set $\pi_\mu^* \leftarrow 0$
16:      else if $\pi_\mu^* > 1$, set $\pi_\mu^* \leftarrow 1$
17:  Verify constraints
18:      if $\eta_\mu (1-\pi_\mu^*)\kappa_\mu + \theta_\mu \pi_\mu^* \delta_\mu > \lambda_\mu$ or
         $\frac{(1-\pi_\mu^*)\kappa_\mu}{\varphi_\mu} + \frac{\pi_\mu^* \kappa_\mu}{\psi} + \frac{\pi_\mu^* \delta_\mu}{\chi_\mu} > \Delta_\mu$
```

```
19: Not feasible: fallback to local/hybrid/edge as required
20: Choose the best feasible $\pi_\mu \in \{0,1\}$ or minimize within [0,1]
21:     end if
22:     end for
23:     return all $\pi_\mu^*$
```

3.6.2. Local Computing Feasibility Check

This Algorithm 3 attempts to determine whether or not a mobile device can process its respective given task uniquely on its own. It computes the local execution time and the amount of energy required by each device. It then compares these values with the device deadline and available energy. In case either the time or energy requirement is excessive, the task will be flagged as not suitable to execute locally. Given that the two requirements are met, the Job can be performed without the device offloading any element to an edge server.

```
Algorithm 3: Local computing feasibility check
Input: $\mathcal{M}, \kappa_\mu, \delta_\mu, \psi, \chi_\mu, \theta_\mu$
Output: Partition of local/edge execution per device
1 for each device $\mu \in \mathcal{M}$ do
2     Select $\pi_\mu \in [0,1]$
3     Compute:
              Local: $(1-\pi_\mu)\kappa_\mu$ cycles
              Edge: $\pi_\mu\kappa_\mu$ cycles, $\pi_\mu\delta_\mu$ bits
4     Compute delays and energies as in Algorithm 1
5     Check constraints
6 end for
7 return optimal split
```

3.6.3. Edge Offloading Model

This algorithm (Algorithm 4 divides tasks between an edge server and a mobile device. It evaluates a set of possible splits (ranging from no offloading to full offloading) per device and determines the number of CPU cycles and amount of data that is processed locally or at the edge. Next, it calculates the delay and energy costs of each split using the calculations that were used in Algorithm 1. The process eliminates those divisions that could not satisfy the constraints of the device. The algorithm then finds the best method of distributing tasks among the devices, keeping the right ratio between local and edge processing with the aim of getting the optimal result.

This algorithm determines how each mobile device divides its computing tasks between local processing and offloading to an edge server, aiming to minimise both time and energy consumption. Each device starts by collecting key variables such as CPU requirements, data volume, CPU clock speeds, transmission rates, and energy properties. The algorithm will first calculate the time and energy the device will use to finish the task locally. It then calculates time and energy when the entire task is forwarded to the edge server, both in data transmission and edge computation. Then, regardless of the selected offloading fraction, the algorithm evaluates the total time and energy in case of any combination of local and edge processing.

A cost function is set up to integrate the time and the energy, whereby weight factors of the goals of the systems are matched. The algorithm then checks that the total time does not exceed the task deadline and that energy use stays within budget.

Solving for the point where the cost is lowest, it identifies the best offloading fraction for each device. If the solution falls outside valid limits, it defaults to either full local or full offload. In the end, this approach helps each device finish its task efficiently by balancing execution time and energy use within system limits.

```
Algorithm 4: Edge offloading model
Inputs: Set of mobile devices: $\mathbf{\Phi} = \{\theta_1, \theta_2, ..., \theta_m\}$, For each device $\theta_i$, Required CPU cycles $\psi_i$, Data size to transfer $\chi_i$ (bits), Local CPU frequency $\omega_i$ (Hz), Edge server CPU frequency $\Omega_i$ (Hz), Transmission rate $\tau_i$ (bits/s), Offloading fraction: $\lambda_i \in [0,1]$ ($\lambda_i = 0$: local, $\lambda_i = 1$: full offload, $0 < \lambda_i < 1$ hybrid), Energy/cycle (local) $\varepsilon_i$, Energy/bit (transmit) $\zeta_i$, Task deadline $\Delta_i$ (s), Energy budget: $\Xi_i$ (J).
Outputs: Optimal offloading fraction $\lambda_i$*, Minimum total cost (execution time & energy)
Step 1: Initialize for each device $\theta$ and assign all parameters $\psi_i, \chi_i, \omega_i, \Omega_i, \tau_i, \varepsilon_i, \zeta_i, \Delta_i, \Xi_i$
Step 2: Compute local execution time
    $\theta L_i \leftarrow \psi_i / \omega_i$ (Local only)
Step 3: Compute offloaded execution time
    - $\theta O_i \leftarrow (\chi_i / \tau_i) + (\psi_i / \Omega_i)$ (Transmission + Edge computation)
Step 4: Compute local execution energy
    - $\eta L_i \leftarrow \varepsilon_i * \psi_i$
Step 5: Compute offloaded execution energy
    - $\eta O_i \leftarrow \zeta_i * \chi_i$
Step 6: Total execution time (Hybrid offload)
    - $\theta T_i \leftarrow (1 - \lambda_i) * (\psi_i / \omega_i) + \lambda_i * [(\chi_i / \tau_i) + (\psi_i / \Omega_i)]$
Step 7: Total energy consumption (Hybrid offload):
    - $\eta T_i \leftarrow (1 - \lambda_i) * (\varepsilon_i * \psi_i) + \lambda_i * (\zeta_i * \chi_i)$
Step 8: Formulate cost function
    $C_i(\lambda_i) \leftarrow \alpha * \theta\_T_i + \beta * \eta\_T_i$ (where $\alpha$, $\beta$ are weighting factors for time/energy tradeoff)
Step 9: Set constraints
    $\theta T_i \leq \Delta_i$ (Task deadline constraint)
    $\eta T_i \leq \Xi_i$ (Energy budget constraint)
    $0 \leq \lambda_i \leq 1$ (Fraction bound)
Step 10: Find Optimal $\lambda_i$*
    Take $\partial C_i(\lambda_i)/\partial\lambda_i = 0$, solve for stationary point $\lambda_i$*
    if $\lambda_i^* < 0$, set $\lambda_i^* = 0$ (full local)
    if $\lambda_i^* > 1$, set $\lambda_i^* = 1$ (full offload)
    else, use $\lambda_i$* as optimal
```

3.7. Baseline Algorithms

Baseline algorithms like SJF, Round Robin, FCFS, and EADF-PSO provide foundational comparisons to evaluate performance improvements in advanced scheduling models, highlighting efficiency, energy savings, and responsiveness gains.

3.7.1. Shortest Job First (SJF) [17]

This is the algorithm in scheduling that chooses the Job that has the shortest run time to execute first. This scheme is good when the lengths of jobs are known beforehand to reduce the average waiting time. Nevertheless, SJF cannot be used in a real-time system or with dynamic workloads because it might starve the longer tasks. It is not responsive to a heterogeneous or unpredictable task arrival in a cloud environment due to its static nature, which is therefore less efficient in such an environment.

3.7.2. Round Robin (RR) [18]

This is a scheduler for time sharing, i.e., it gives each task a rigid time quantum and rotates through tasks in a circular queue. It is equal and straightforward, such that no task will be left unattended. RR does add context switching overhead; more work is more expensive. It can operate efficiently in interactive systems, but it is not versatile enough to handle the complexity of the task or time pressure. RR may cause inefficient energy consumption in cloud or edge computation and increase the makespan in high-load cases.

3.7.3. First-come, first-served (FCFS) [19]

This is a non-preemptive approach that follows the sequence of execution as they are received. It is simple to administer and fair since it does not favour any activity.

However, it is affected by the converse effect, as short tasks wait behind long tasks, resulting in poor average turnaround time and energy inefficiency. The absence of latency-sensitive or energy-limited applications hinders the advancement of FCFS in various cloud or mobile edge systems.

3.7.4. Energy Aware Double Fitness Function PSO (EADF-PSO) [20]

This will be a metaheuristic approach based on the PSO process, but its features will include a dual fitness criterion that will take into account the execution time as well as the energy consumed. It distributes tasks among computing nodes with optimised tradeoffs to achieve better performance than the static heuristics.

EADF-PSO lowers energy consumption and makespan, whereas it is unaware of queuing delay and dynamic workload balancing. Although it performs well in the case of the static cloud, its scaling might fail at high levels of concurrency or when using limited edge resources.

4. Results and Discussion

The simulation was implemented on EdgeCloudSim to estimate the work of the proposed DAEE-QPSO algorithm, which emulates a hierarchical edge cloud design that entails mobile gadgets, edge servers, and clouds based on data centres. The virtual machines (VMs) are held constant in this experimental setup to represent a problem with limited resources that is often observed in real-time mobile edge computing. At the same time, the amount of work given slowly increases from 50 to 250 in steps of 25 to see how the scheduling algorithm responds to different amounts of requests. This setup simulates real-life situations where there is pressure from waiting in line and competition for resources, making it possible to effectively test how well the proposed algorithm handles delays and meets Service Level Agreements (SLAs). Important performance indicators (average execution time, energy consumption, task throughput, and SLA violation rate) are used to indicate the system's efficiency.

4.1. Analysis of Performance Metrics

Performance evaluations are essential in Mobile Edge Computing (MEC), and they include average running time, average energy consumption, task throughput rate, and energy saving. Mean run-time shows the speed at which a task will run, which directly affects application responsiveness, particularly in real-time or latency-sensitive applications. Average energy use is essential in extending the battery life of mobile devices and lowering the energy costs of operations in edge servers. Task throughput rate describes the number of tasks that can be effectively processed depending on the unit of time between them; the larger the rate, the higher the resource utilisation rate and system scalability. Finally, average energy saving measures the effectiveness of an algorithm in energy saving in the system. The success in improving these metrics means that the MEC infrastructure will be able to serve all the users, operate effectively, and be sustainable. Therefore, optimising such meeting points is crucial to developing adaptive, energy-efficient, high-performance edge-cloud systems to support contemporary applications.

Figure 3 compares the average runtime of five scheduling techniques for the number of tasks between 50 and 250. FCFS exhibits the greatest run time, rising from 148.9 milliseconds (50 tasks) to 676.3 milliseconds (250 tasks) because of its strict, first-in-the-queue scheduling, which amounts to long queues. Round Robin (RR) is a little bit faster, though, and it improves (135.3 ms to 641.8 ms), but its time-slicing overhead is inefficient when the task volume increases. Shortest Job: shorten the runtime (e.g., 122.5 ms at 50 tasks, 602.3 ms at 250 tasks) by preferring the shorter jobs, but it declines under the unbalanced workload. EADF-PSO, based on the use of smart metaheuristic mapping, significantly minimises runtime as compared to 98.7 ms in terms of runtime reduction to 432.5 ms, which is an improved adaptation to task heterogeneity.

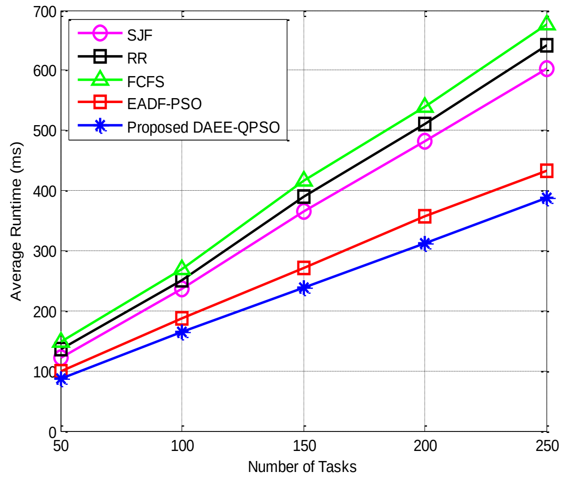

Fig. 3 Performance comparison of the average run time of different scheduling algorithms

The proposed DAEE-QPSO brings the best results with execution times of 87.6 ms (50 tasks) and 388.1 ms (250 tasks). Its strength is that it is a hybrid of queue awareness and energy delay optimisation, providing efficient adaptive load sharing with varying task loads.

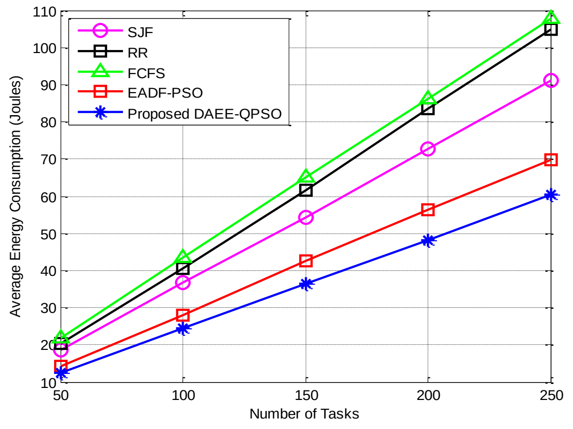

Fig. 4 Performance comparison of the average energy consumption of different scheduling algorithms

Figure 4 presents a comparison of the average energy consumption of five scheduling methods during different stages of increasing task load. First-Come-First-Served (FCFS) records the highest energy consumption with an increase of 21.7 J (50 tasks) compared to 107.8 J (250 tasks) because of the longer execution time, regardless of the urgency of tasks and the system workload. Round Robin (RR) also shows a similar pattern, with energy use ranging from 20.3 J to 104.9 J, because it follows a sequence of rotating tasks, which leads to more context switching and wasted resources. Shortest Job First (SJF) also performs slightly better, with energy consumption ranging from 18.5 J to 91.2 J, but it still lacks dynamic energy control because it focuses on executing shorter jobs. The EADF-PSO method, which relies on metaheuristics, reduces energy consumption (from 14.2 J to 69.8 J) by efficiently utilising task peculiarities and information about system processes for effective mapping. Nevertheless, the proposed DAEE-QPSO achieves the lowest energy consumption, which is 12.6 J and does not exceed 60.3 J even when processing 250 tasks. This is better because of its energy and delay dual fitness optimization and queue awareness scheduling.

Figure 5 shows the average task throughput rate (tasks/sec) of five scheduling algorithms. FCFS produces the lowest throughput, fluctuating between 0.36 and 0.37 tasks/sec due to its complete ignorance of queue loads. Round Robin (RR) takes a slight step forward (0.380.39 tasks/sec), yet context-switching overhead is large. SJF is slightly improved (0.41 0.42 tasks/sec) as shorter jobs are favoured, but there is no energy or load balance focus. With the help of metaheuristic optimisation, EADF-PSO records a dramatic improvement (0.520.53 tasks/sec) based on intelligent allocation of tasks.

The proposed DAEE-QPSO is the best approach because it retains 0.58-0.59 tasks/sec compared to all other approaches since it considers both queuing delay modelling and energy delay fitness optimization training, which entails the optimised, balanced task assignment and least idle time in edge-cloud nodes.

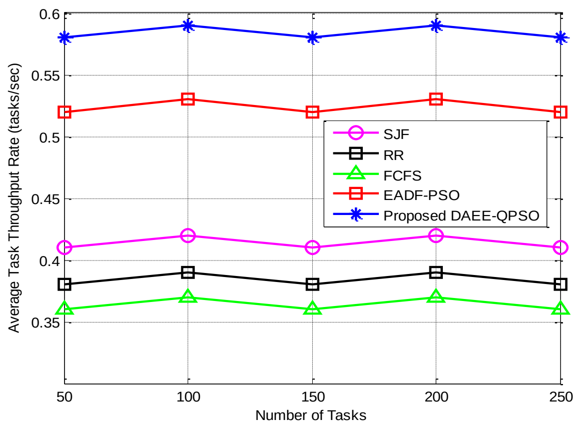

Fig. 5 Performance comparison of the average task throughput rate of different scheduling algorithms

Figure 6 provides a comparison of the average energy saving of the five different scheduling algorithms according to the task count between 50 and 250 tasks. The lowest savings in FCFS are between 9.5 and 10.1% since they are inefficient in handling tasks and have a long downtime in making use of resources. Round Robin (RR) is slightly better (10.8 to 11.3%), but its efficiency comes with the cost of energy overhead due to numerous context switches.

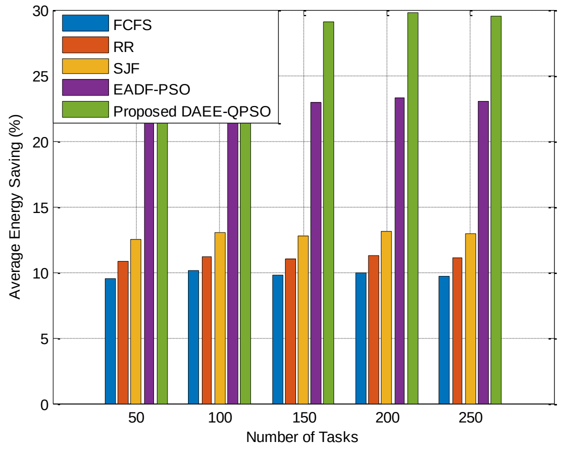

Fig. 6 Performance comparison of the average energy saving of different scheduling algorithms

SJF can marginally (12.5% to 13.1%) exceed the others in finishing shorter tasks in less time, but it is not energy-aware. EADF-PSO records significant energy savings (22.6 to 23.3%) as a result of optimization in task-to-resource mapping accomplished through a dual fitness function. The proposed DAEE-QPSO achieves, with a queuing delay model and a dynamic energy-delay tradeoff, the best marinating savings (28.7–29.8%), considering offloading rates that are intelligent and a secure balance on resources in a very busy mobile edge environment.

5. Conclusion

In conclusion, an efficient and scalable task scheduler, DAEE-QPSO, will be used to handle resource distribution challenges within a mobile edge cloud scenario. The algorithm with the queuing delay modeling shows better performance, less runtime, energy consumption, fewer tasks in the backlog and more energy savings and throughput by integrating in the PSO. Baseline algorithms like SJF, RR, FCFS, and EADF-PSO comparative experiments indicate DAEE-QPSO is improving on a majority of the performance measurements provided in every single instance.

The average run time of DAEE-QPSO is always the lowest, decreasing constantly with task load, from 87.6 ms to 388.1 ms, compared to FCFS (676.3 ms) and EADF-PSO (432.5 ms). DAEE-QPSO again outperforms RR (20.3 J) and FCFS (21.7 J) in average energy usage, needing 12.6 J in 50 jobs. E-E-QPSO consistently achieves 0.58-0.59 tasks/sec, compared to SJF (0.41) and FCFS (0.36). Finally, DAEE-QPSO saves the most energy at 29.8%, compared to 23.3% for EADF-PSO. Optimized energy-delay trade and queue-aware decision-making allow real-time IoT and smart city applications to handle more variable workloads and resources dynamically. Future work can provide an expansion of the model by incorporating reinforcement learning to allow adaptation to the parameters it uses or multi-objective optimization to enable even more flexibility and efficiency in scheduling.

References

[1] Peter Mell, and Tim Grance, "The NIST Definition of Cloud Computing," *National Institute of Standards and Technology*, pp. 1-7, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[2] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan, "GreenCloud: A Packet-Level Simulator of Energy-Aware Cloud Computing Data Centers," *The Journal of Supercomputing*, vol. 62, pp. 1263-1283, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[3] Mohamed Abd Elaziz et al., "Task Scheduling in Cloud Computing Based on Hybrid Moth Search Algorithm and Differential Evolution," *Knowledge-Based Systems*, vol. 169, pp. 39-52, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[4] Sukhpal Singh, and Inderveer Chana, "A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges," *Journal of Grid Computing*, vol. 14, pp. 217-264, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[5] AR. Arunarani, D. Manjula, and Vijayan Sugumaran, "Task Scheduling Techniques in Cloud Computing: A Literature Survey," *Future Generation Computer Systems*, vol. 91, pp. 407-415, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[6] Mahendra Bhatu Gawali, and Subhash K. Shinde, "Task Scheduling and Resource Allocation in Cloud Computing Using a Heuristic Approach," *Journal of Cloud Computing*, vol. 7, pp. 1-16, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[7] Mohit Agarwal, and Gur Mauj Saran Srivastava, "A PSO Algorithm Based Task Scheduling in Cloud Computing," *International Journal of Applied Metaheuristic Computing*, vol. 10, no. 4, pp. 1-17, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[8] Pengfei Wang et al., "Joint Task Assignment, Transmission, and Computing Resource Allocation in Multilayer Mobile Edge Computing Systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2872-2884, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[9] Binh Minh Nguyen et al., "Evolutionary Algorithms to Optimize Task Scheduling Problem for the IoT Based Bag-of-Tasks Application in Cloud–Fog Computing Environment," *Applied Sciences*, vol. 9, no. 9, pp. 1-20, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[10] Xudong Niu et al., "Workload Allocation Mechanism for Minimum Service Delay in Edge Computing-Based Power Internet of Things," *IEEE Access*, vol. 7, pp. 83771-83784, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[11] Gagandeep Kaur, Balraj Singh, and Muhammad Faheem, "Bioinspired Adaptive Resource Scheduling for QoS in Mobile Edge Deployments," *IET Communications*, vol. 19, no. 1, pp. 1-14, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[12] Ming Chen et al., "Genetic Algorithm with Skew Mutation for Heterogeneous Resource-Aware Task Offloading in Edge-Cloud Computing," *Heliyon*, vol. 10, no. 12, pp. 1-17, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[13] Pijush Kanti Dutta Pramanik, Tarun Biswas, and Prasenjit Choudhury, “Multicriteria-Based Resource-Aware Scheduling in Mobile Crowd Computing: A Heuristic Approach,” *Journal of Grid Computing*, vol. 21, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[14] Qi Zhang et al., “Task Offloading and Resource Scheduling in Hybrid Edge-Cloud Networks,” *IEEE Access*, vol. 9, pp. 85350-85366, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[15] Rajni Aron, and Ajith Abraham, “Resource Scheduling Methods for Cloud Computing Environment: The Role of Meta-Heuristics and Artificial Intelligence,” *Engineering Applications of Artificial Intelligence*, vol. 116, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[16] Yufei Wang et al., “Resource Scheduling in Mobile Edge Computing Using Improved Ant Colony Algorithm for Space Information Network,” *International Journal of Satellite Communications and Networking*, vol. 41, no. 4, pp. 331-356, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[17] Jia Ru, and Jacky Keung, “An Empirical Investigation on the Simulation of Priority and Shortest-Job-First Scheduling for Cloud-Based Software Systems,” *2013 22nd Australian Software Engineering Conference*, Hawthorne, VIC, Australia, pp. 78-87, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[18] Ashkan Emami Ale Agha, and Somayyeh Jafarali Jassbi, “A New Method to Improve Round Robin Scheduling Algorithm with Quantum Time Based on Harmonic-Arithmetic Mean (HARM),” *International Journal of Information Technology and Computer Science*, vol. 5, no. 7, pp. 56-62, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[19] A.V. Karthick, E. Ramaraj, and R. Ganapathy Subramanian, “An Efficient Multi Queue Job Scheduling for Cloud Computing,” *2014 World Congress on Computing and Communication Technologies*, Trichirappalli, India, pp. 164-166, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[20] Yao Lu et al., “EA-DFPSO: An Intelligent Energy-Efficient Scheduling Algorithm for Mobile Edge Networks,” *Digital Communications and Networks*, vol. 8, no. 3, pp. 237-246, 2021. [CrossRef] [Google Scholar] [Publisher Link]