International Journal of Applied Mathematics

Volume 38 No. 10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

ENHANCED ONDEMAND PREEMPTIVE SELF-SCHEDULING BASED ON MAX-QUEUING ENERGY-AWARE REQUEST PROTOCOL FOR LOAD BALANCING IN A DECENTRALIZED CLOUD ENVIRONMENT

S.Sivasankari[1,*], Dr. Vishwa priya V[2]

Research Scholar, Department of Computer Science, Vels Institute of Science Technology & Advanced Studies, Chennai, Tamil Nadu, India

Corresponding Mail Id: sankarivs96@gmail.com

Orcid : 0009-0000-0980-5348

Department of Computer Science and Information Technology, Vels Institute of Science Technology &Advanced Studies

Chennai, Tamil Nadu, India

Mail Id: vishwapriya13@gmail.com

Orcid: 0000-0002-4678-9516

Abstract

In recent years, the adoption of cloud computing has surged, with organizations and individuals increasingly leveraging cloud solutions for data storage and processing. This growing reliance on cloud services necessitates effective load balancing in decentralized cloud environments to optimize resource utilization, enhance system performance, and improve user experiences. However, current load-balancing techniques often fail to address scheduling performance issues, primarily due to task-shifting problems stemming from improper priority assignments. These deficiencies can lead to deadlocks, task collisions, and failures, increasing time complexity and hindering overall system efficiency. To address these issues, we propose a novel On-Demand Pre-Emptive Self-Scheduling (ODPSS) approach incorporating a Max-Queuing Energy-Aware Request Protocol (MQEARP) tailored for decentralized cloud environments. Our methodology begins with collecting virtual incoming request tasks from the web request handler within the cloud server, creating cloud request logs. We utilize a Task Time Intensity Rate (TTIR) to estimate task completion times, which are subsequently integrated into each task by establishing a max queue priority system. We employ Particle Swarm Optimization (PSO) to refine our approach further and evaluate critical server handling parameters, including energy consumption, processing time, queuing length, demand, and request delay dependencies. This evaluation yields an absolute mean rate based on the maximum count of requests the server handles, enabling us to prioritize and queue tasks effectively within the Max-Queuing Energy-Aware Request Protocol (MQEARP) framework. The OnDemand service is activated as the web server receives dynamic requests, allowing tasks to await processing in the queue. To manage this influx efficiently, our Enhanced Preemptive Self-Scheduling strategy facilitates the transition of OnDemand requests into a

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

prioritized queue, enabling the deployment of temporary Virtual Machines (VMs) to alleviate the cloud server's workload. After the extended OnDemand service is completed, these virtual servers are terminated, effectively reducing the operational burden on the cloud infrastructure. The objectives of the proposed technique are to enhance productivity, optimize server resource allocation by considering the varying importance of user tasks, and mitigate the risk of server failures. By addressing the identified challenges and filling existing research gaps, our proposed method aims to contribute significantly to the field of cloud computing, paving the way for more efficient and reliable load-balancing solutions in decentralized environments. Based on the performance analysis of the proposed method, such as delay tolerance, energy consumption, execution cost efficiency, self-scheduling, time complexity, and resource allocation number, cloud request logs are generated, and evaluation of the ODPSS method improves to 96.9%.

Keywords: Load balancing, self-scheduling, max-queuing, energy, cloud environment, virtual machine, request protocol, TTIR and ODPSS.

1. Introduction

The evolution of machines and sensors connected is changing almost every facet of human existence. It has had a significant socioeconomic influence and changed several industries. Although Cloud Computing (CC) can afford an extensive selection of efficient web-based facilities, it has grown steadily popular in recent years. The way businesses handle their computing needs has changed due to this technological advancement, which offers scalable and flexible outcomes to meet the cumulative needs of contemporary programs. Although cloud infrastructures manage workloads and resources effectively, several problems could arise. One essential method for dividing up incoming tasks or requests among several servers is Load Balancing (LB). This offers constant system stability, effective resource management, and system enhancement. Numerous factors, such as evolving labor force variations, diversity of resources, changing requirements for scalability, and shifting program difficulties, contribute to the complexity of LB in systems in the cloud.

Virtual Machines (VMs) with different durations, start times, and execution times are assigned jobs in the CC environment. As a result, it is crucial to distribute these loads between the VMs. To maximize their potential and boost system performance, LB must be implemented so that each VM is balanced. LB is a complex topic in CC. Deploying these properties in the cloud is challenging because the workload fluctuates based on user requirements. LB is a method that distributes requests between several machines using task scheduling so that a more significant number of jobs may be executed in less time while also monitoring the performance of VMs [3].

In the context of distributed computing, workflow is commonly used. Workflow activities must be appropriately mapped to CC resources because, in general, a job's speed and duration in a resource are inversely correlated. Resources may be in different locations at different times due to their inclusion in the proposed architecture. The energy and bandwidth utilization of resources is also restricted. So, it makes sense to cluster computing resources [4].

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Dynamic LB techniques yield fault-tolerant systems that scale with the number of VMs and jobs in an actual environment. Task completion is impeded by the overloaded VM, resulting in resource waste. The CC aspect of virtualization enables the user to be separate from the underlying work carried out by the other components inside the CC. Clusters are created by grouping various virtual computers. The cloud service provider that hides the existence of a physical server from the virtual instance it is operating on is the most secure in terms of security. If this is the case, a committed hacker will have difficulty attempting to hack the specific task available in the physical environment. To increase security, the work requires a Service Level Agreement with third parties [5]. The existing LB approaches frequently fail to address scheduling performance difficulties, mainly due to task-shifting issues caused by incorrect priority assignments. These flaws can cause deadlocks, task collisions, and failures, raising temporal complexity and reducing overall system efficiency.

This paper contributes to applying the ODPSS technique for gathering virtual receive request tasks from load-balancing request handlers on cloud servers and generating their corresponding records. Additionally, it utilizes the TTIR technique based on request protocols to estimate task completion times in maximum queue scheduling. The proposed method also incorporates the PSO algorithm to assess server processing parameters, including energy consumption, processing time, queue length, demand, and delay dependency. Furthermore, it integrates maximum queue tasks into self-scheduling according to request protocols to accomplish LB in cloud environments. This technique is improved by the MQEARP framework, which offers absolute average speedup based on the maximum number of features. Finally, by analyzing the differing priorities of user tasks, the suggested ODPSS method's effectiveness can increase productivity and optimize server resource allocation while lowering the chance of server tasks.

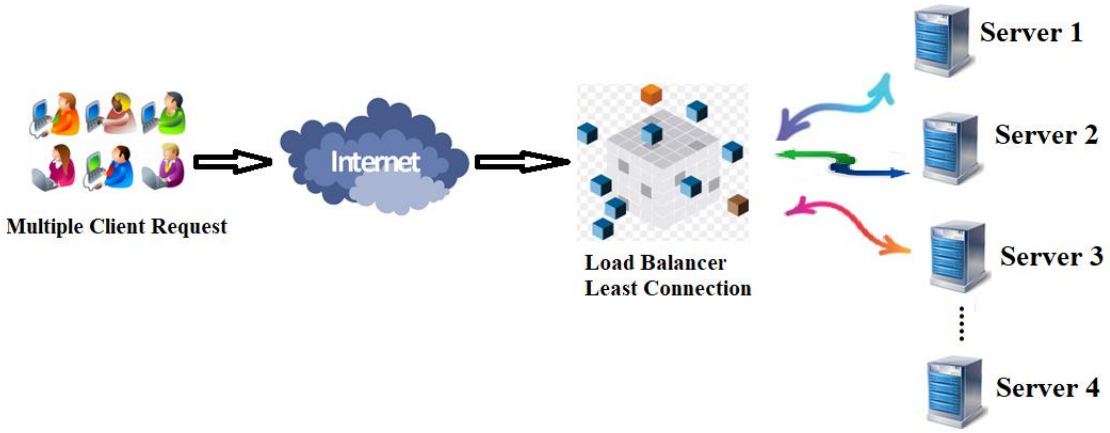

Figure 1. The Basic Architecture Diagram based on the Cloud Environment

As illustrated in Figure 1, the proposed method provides a basic architecture diagram based on the load context of the improved on-demand preemptive self-scheduling based on load-balanced request protocols in cloud servers. Furthermore, virtual incoming request tasks can be collected, and related logs can be created. Additionally, it uses essential algorithms to improve productivity, optimize server resource allocation, estimate task completion time,

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

analyze server performance parameters, and provide average speed based on features. Furthermore, resource allocation based on load-balancing techniques can be applied to enhance resource allocation, lower the likelihood of outages, and efficiently distribute multi-client requests among several servers.

2. Literature Survey

By identifying overloaded and underloaded nodes, an LB technique divides the load among them. This challenge has piqued scholars' interest in the last decade, and numerous solutions have been offered [6]. This absence motivated researchers to conduct the present investigation to compile and analyze previously published papers on fault tolerance LB methods in CC.

The literature has employed various LB strategies to improve end-user satisfaction, optimize cloud service efficiency overall, and manage and fulfill client requests for suitable cloud nodes [7]. An efficient LB technique increases efficiency and support utilization by evenly distributing workload throughout the system's nodes.

Based on task and workflow scheduling solutions for the cloud, the recommended PSO was thoroughly examined in the literature [8]. Furthermore, it analyses the goals, characteristics, and limitations of the PSO algorithms used to classify the suggested scheduling schemes.

Virtual Machine (VM) efficiency metrics are derived from their creation parameters and monitored in a CC environment [9]. Gene Expression Programming (GEP) is used to create symbolic regression frameworks that explain VM effectiveness and estimate VMH loads after LB.

The suggested method considers VM priority, resource allocation, and Quality of Service (QoS) task characteristics to optimize resources and improve LB [10]. The suggested LB technique addresses the challenges stated along with the current examination gap on the basis of the literature's conclusions.

The suggested approach makes use of hybrid approaches such as fuzzy-tOPSIS and PSO. The multi-objective Service Level Agreement (SLA)-based work scheduling problem is resolved in the cloud [11] through Fuzzy TOPSIS, which weighs the total energy, price, and implementation period as a gauge of objectivity.

The suggested Predictive Priority-based Modified Heterogeneous Earliest Finish Time (PMHEFT) methodology has resolved the glitch and can foresee the application's future resource requirements. They proposed the PMHEFT approach to reduce the make-span of a particular workflow application by enhancing the LB across all VMs [12].

They introduce LB for Resource Optimization (LBRO), a collaborative cloudlet platform designed to handle CC difficulties despite keeping customer needs foremost [13]. Because of the distance and reliance on the Internet, remote cloud solutions face network latency and capacity limitations.

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

They established the Resource Intensity Aware LB technique (RIALB). It calculates the best weights for considering communication costs and performance loss caused by migrations. However, the particular characteristics of clouds present severe obstacles to producing effective and efficient LB [14].

The standard Ant Colony-based Optimization (ACO) approach is split into two stages. The initial stage addresses the complex planning issue with an event-based scheduler. These two approaches constrain project planning and resource allocation [15]. Yet, it is also necessary to control the system's stability and be adaptable when updating it. Static data is utilized to optimize project schedules.

The Rock Hyrax methodology utilizes objective functions to match jobs with accessible capabilities. The goal function considers a range of QoS characteristics, including makespan, reaction time, and energy efficiency. To group related resources, scheduling also considers the behavior and characteristics of the node, including its processing capacity, storage, and network connectivity [16].

An energy-aware LB challenge with latency in a CC architecture. The LB issue mainly discusses two key performance metrics: response time and energy [17]. As a result, energy minimization should be considered when LB is critical for providing improved QoS.

Regarding the additional workload per VM, they offer a fair task LB technique to improve the architecture mean reaction time and makespan in the cloud architecture [18]. The concern is a fundamental finite-state Markov process with a balancing equation for each state.

Table 1. Cloud Performance based on Task Scheduling

Author	Year	Methodology	Limitation	Achieved Result
B. Kruekaew [19]	2022	Multi-objective task scheduling optimization based on the Artificial Bee Colony Algorithm (MTSO-ABC)	Although virtualization is critical in cloud technology, difficulties happen frequently, including poor scheduling or assigning tasks to VMs that cannot match the requests.	83%
Aggarwal, A [20]	2019	Self-Adaptive Fruit Fly Optimization (SA-FFO)	Yet, jobs that various users contribute may have variable memory space, data traffic, and reaction time requirements.	53%
Jafarnejad Ghomi, E [21]	2019	PSO	The manufacturing sector is a complex operation that uses cutting-edge technology.	48.6%

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

M. Menaka [22]	2024	Supportive PSO with Time-Conscious Scheduling (SPSO-TCS)	To resolve the resource allocation strategy of the cloud architecture, an exhaustive examination of the data caused by this process is required.	69%
S.Rekha [23]	2020	Multi-Level Queue Scheduling with PSO (MlQS-PSO)	Devices have an operating system, a small browser, and far less memory.	71%
Zi-Ren Wang [24]	2024	Multi Adaptive Learning for PSO(MALPSO)	Determining the best plan for resource allocation is critical for improving effectiveness.	73.2%
Kumar, M [25]	2020	PSO	Thus, a compromise approach is required to balance opposing goals.	74.6%
Shikha Chaudhary [26]	2023	Modified PSO	The selected method of scheduling defines whether to allocate assets to carry out tasks in a successful way to meet the needs of the end users, which has an impact on the general efficiency of the cloud system.	3.48%
M. Menaka [27]	2024	Time-Conscious Scheduling combined with Supportive Particle Swarm Optimization (SPSO-TCS)	Clients must connect to servers to utilize on-demand services. Yet, power usage has become an issue.	75%
Sirisha Potluri [28]	2020	Task Particle Optimization (TPO)	It could get complicated to solve efficiently and effectively as the issue's size and dimensions grow.	74.8%
Farid, M [29]	2020	PSO	Among the primary issues associated with the suggested solution is managing QoS concerning performance and reliability through the workflow scheduling procedure. This means using heterogeneous cloud resources to schedule complicated processes in a way that	30%

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

			simultaneously minimizes execution time and costs.	
Khaledian, N [30]	2024	PSO- Simulated Annealing (PSO-SA)	An IoT application may demonstrate numerous critical criteria at the same time, such as a deadline and runtime, while also facing resource constraints and excessive energy consumption. This has led to the adoption of a computing environment and scheduling as a primary concern.	9%

Table 1 shows that cloud performance can be evaluated regarding task scheduling using the previous section's methods, limitations, and implemented results.

A unique method called Priority-based LB (PLB) is developed for the CC environment. The PLB uses several queues to enable resilient and adaptable job scheduling. Previous studies have created numerous ways to prioritize jobs and map them to various cloud resources. A performance bottleneck remains due to insufficient attention devoted to underutilized resources and low-priority processes, eventually leading to a starving concern [31].

A creative approach for job scheduling that addresses time execution and energy usage by utilizing an energy-efficient dynamic decision-making process. The proposed method quickly adapts to mobile device energy, time computation, and CC workloads. Utilizing the suggested empirical methodology is task scheduling. Yet, while using a mobile phone for work, energy supply and transmission pose problems for time management and energy efficiency [32].

The author recommended employing Dynamic Voltage and Frequency Scaling (DVFS) in combination with a cost-to-time ratio modification strategy to address task scheduling on heterogeneous multiprocessor systems using an efficient Cuckoo Search methodology that uses adaptive discovery probability and Gaussian random walk. Nevertheless, the recommended approach often has time constraints. Since excessive energy consumption is a barrier to HPC systems, one critical research challenge for heterogeneous computational systems is how to provide services to applications to minimize energy consumption while meeting the time needs of the applications [33].

The author devised an optimization model called the Greedy-based Best Fit Decreasing (GBFD) method. The GBFD algorithm uses a cost efficiency factor determined by CN properties to choose an appropriate CN for each VM request. However, prior relevant works typically ignore the impact of dependability problems, like Computing Node (CN) recoveries and failures, which don't accurately represent real-world CDC scenarios [34].

Cloud federation allows for resource pooling; however, the current techniques of building federations require complicated computation to ensure federation stability. To solve this

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

shortcoming, the author presented Cloud Light-Federation Sharing (CLFS), a Pareto optimum resource-sharing solution that allows each cloud to choose the best strategy for itself and form a federation without the need for laborious computations to divide up service requests and earnings. Furthermore, Cloud Cooperative-Federation Sharing (CCFS), an ideal resource-sharing solution where cloud providers fully cooperate and share revenues fairly, was developed [35].

3. Proposed Methodology

In this section, we use the proposed ODPSS approach to collect virtual receive request tasks from web request handlers on cloud servers and generate their logs. The task completion time can be estimated by establishing a maximum queue priority system based on TTIR technology integrated with each task. Moreover, utilizing the PSO algorithm, the presented method can estimate server processing parameters, including energy consumption, processing time, queue length, demand, and delay dependencies. In addition, an MQEARP framework can be created to prioritize and order tasks and assign an absolute average speed based on the maximum number of functions. Furthermore, according to the proposed method, the virtual server can be shut down when the extended on-demand service ends and the operational load on the cloud infrastructure can be effectively reduced. The effectiveness of the proposed ODPSS method can improve productivity and server resource allocation by taking into account the different importance of user tasks and reducing the risk of server failure.

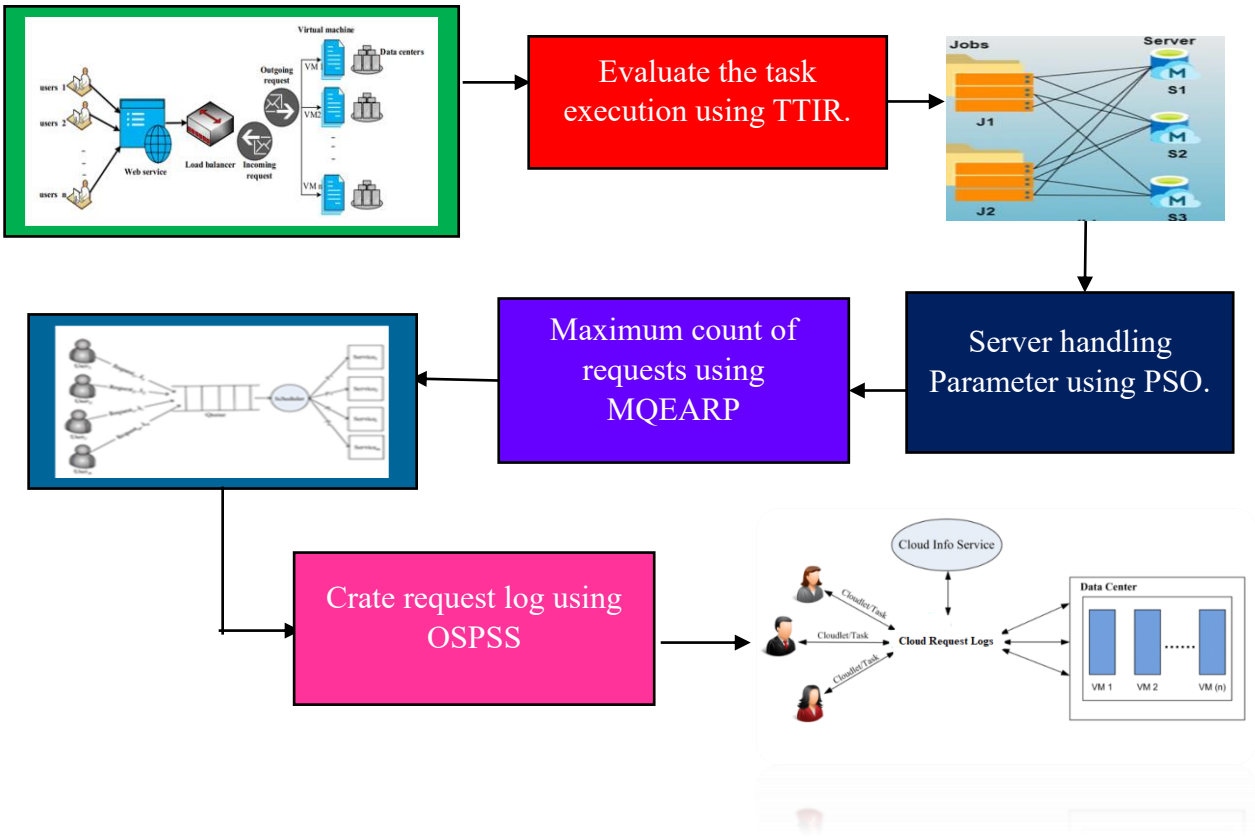

Figure 2. The Proposed Architecture Diagram for ODPSS

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

According to the proposed ODPSS method's architecture diagram, productivity is improved, as shown in Figure 2. Furthermore, it estimates the task execution time using the proposed TTIR technique, optimizes the server processing parameters based on PSO, calculates the maximum number of requests using the MQEARP method, and calculates the absolute average rate. Moreover, to maximize the number of requests the server handles, a self-scheduling strategy can also be used to prioritize requests according to demand by creating a cloud request log using the proposed formal ODPSS method. In addition, resource allocation can be improved by targeting the importance of different user tasks.

3.1 Task Time Intensity Rate (TTIR)

This section uses the task time-intensity rate to evaluate the execution time and generate maximum queue priority settings. To assess the scheduling time of real-time tasks based on the TTIR technique, the intensity ratio can be calculated by integrating different arrangements for real-time tasks in the current scheduling process. Furthermore, real-time jobs are scheduled on virtual machines using the proposed TTIR approach, thus reducing the expected execution time of the submission queue. Moreover, task completion times and intensity ratios can be analyzed by developing a maximum queue priority system for real-time task scheduling in cloud environments.

Algorithm: TTIR

Input: Initialize the number of requests

Output: Intensity rate of execution time ← I_R

Start

Step1: Request time←number of request time of R_1

Step2: Calculate the time scheduling task ←assigned scheduling time of R_1

For each request, R_i, to the assigned time, do

$T_s \leftarrow$ compute time scheduling task toR_i

If $scheduling\ time\ task \neq T_s$, then

Step 3: Estimate the intensity ratio execution time← $Exe(t)$

$$Exe(t) = E_t + T(R_i, 1)$$

$$T(R_i) = ETI_R.w(R)$$

Return I_R

End if

End for each

End

A TTIR pseudocode representation of the algorithm can estimate the time it takes to process a request and complete a task. Let's assume E, E_t −eexecution time task,

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

(I_R) −intensity rate, R_i −request task time, $c(R_i, 1)$ −communication request time, $\mathrm{T_s}$ −task scheduling, and R_1 −request time, w-weight.

3.2 Particle Swarm Optimization (PSO)

In this section, the particle swarm optimization algorithm evaluates critical server processing parameters by analyzing the dependencies of energy consumption, processing time, queue length, demand, and request delay. The PSO algorithm can estimate these server processing parameters based on the social behavior of bird flocks. Additionally, they propose a population-based search function to create clusters from individual particles using the PSO method. When the PSO optimization problem differentiates between a specific task or several sequences of functions, particle swarm optimization can analyze the candidate particles in the swarm based on parameters. In addition, each particle in the PSO system transits a multi-dimensional probe region to change its position in space, searching for individual experiences and neighboring particles. Particles can provide an optimal solution for optimally arranging neighboring particles. By measuring the performance parameters of the PSO algorithm on the server functions, the fitness function adjusts the output of each element that characterizes the optimization problem.

The position and velocity criteria for each job can be the maximum number of iterations the PSO method searches for perfect self-scheduling until it reaches a stopping criterion or a sufficient average fitness value completion period. In addition, the velocity and position of the working particle must be calculated, as shown in Equations 1 and 2.

$$\mathrm{V_{id}^{t+!}} = I_R \mathrm{w_{id}^{t+!}} + \mathrm{C_l} * \mathrm{r_{li}} * (\mathrm{P_{id}} - \mathrm{X_{id}^{t}}) + \mathrm{C_2} * \mathrm{r_{2i}} * \left(\mathrm{P_{gd}} - \mathrm{X_{id}^{t}}\right) \quad (1)$$

$$\mathrm{X_{id}^{t+1}} = \mathrm{X_{id}^{t}} + \mathrm{V_{id}^{t+1}} \quad (2)$$

Step 1: Initialize each particle's position and velocity with swarm self-scheduling

Step 2: Evaluate the fitness of the scheduled particles by mean completion time$\leftarrow \mathrm{F_v}$

For compute fitness$\mathrm{X_i} > \mathrm{P_B}$

$\mathrm{P_B(i)} = \mathrm{X_i}$

If evaluating the fitness function $\mathrm{P_B(i)} > \mathrm{G_B(i)}$

$\mathrm{G_B(i)} = \mathrm{P_B(i)}$

Step 3: Particle velocity can be optimized using equation 1.

Step 4: Use Equation 2 to optimize particle self-scheduling positions

Step 5: If the termination criteria are not met, go to steps 2 and 3.

Return$\leftarrow \mathrm{F_v}$

Stop

Important parameters can be estimated from the task handler on the server until the time to complete the average fit value is sufficient to analyze the particle velocity and position. The

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

maximum number of iterations can also be analyzed to search for a self-schedule. Let's assume t-iteration, d-dimension, w-inertia weight, C_1, C_2 −acceleration constants, R-random values, V-velocity, P-particle, P_{gd} −global best position, G_B −Global best, F_v − fitness value, $P_B(i)$ −particle best position.

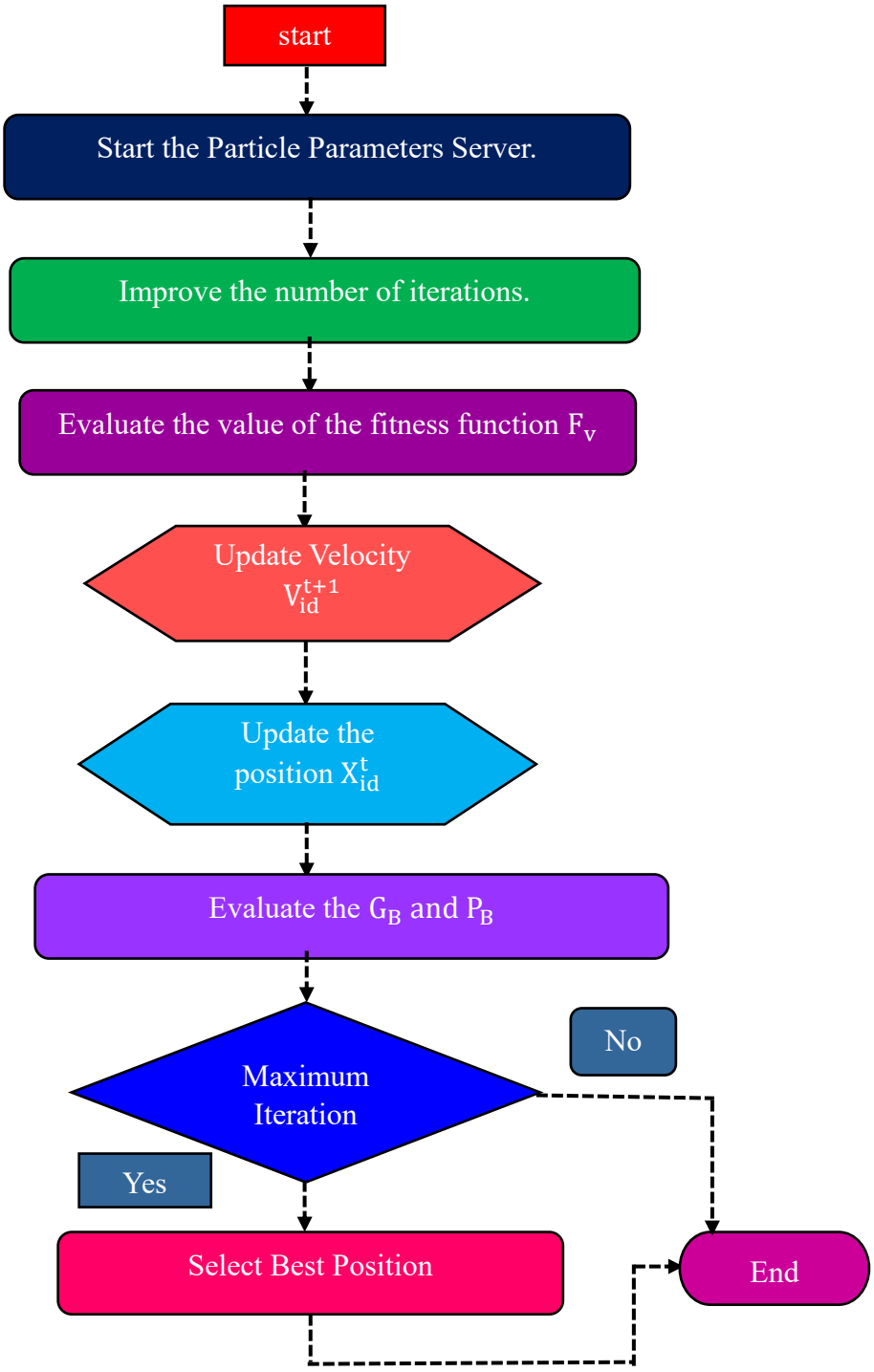

Figure 3. The Flowchart Diagram for PSO

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Based on the flowchart of PSO, the particle parameters can be analyzed by starting the particle parameter server, optimizing the number of iterations to evaluate the value of the fitness function, and selecting the optimal position, as shown in Figure 3.

3.3 Max-Queuing Energy-Aware Request Protocol (MQEARP)

In this section, we use the Max-Queuing Energy-Aware Request Protocol (MQEARP) to handle the maximum count of requests the server handles. This approach is a clever solution to these problems by utilizing the strength of the cloud environment. MQEARP promises to increase network longevity, enhance performance, and adjust to the dynamic nature of optimizing energy consumption and utilizing CC resources. In addition to addressing the urgent energy efficiency problem, this protocol opens the door for further advancements in wireless network routing.

Because of the global nature of client requests and the limitless capabilities of cloud architecture, both the number of queuing models and the source of customers are unrestricted. The protocol usually distributes the load among nodes to increase the network's lifespan and guarantee dependable communication. The following basic equations could be applied in such a protocol. The energy consumed by a node to transmit a packet can be represented as equation 3,

$$G_{da}(p) = F_v\left(G_{elec}.u + \varepsilon_{amp}.u.p^2\right) \quad (3)$$

$G_{da}(p)$ is the transmission energy over the distance p; u is the number of bits in the packet; p is the distance to the receiver; G_{elec} is the energy consumed per bit by the transmitter or receiver electronics; and ε_{amp} is the energy consumed by the transmit amplifier. The residual energy of a node after transmitting a packet can be represented in below equation 4,

$$G_q = E_s - \sum G_{da}(p) \quad (4)$$

Here, G_q represents the residual energy of the node, E_s is meant for the initial energy of the node, and the total energy consumption in the transmission is represented by $\sum G_{da}(p)$. The queue length H of a node, which represents the number of packets waiting to be transmitted, is represented in equation 5,

$$H = H_k + \lambda - \mu \quad (5)$$

here, the previous queue length is represented by H_k, λ, and μ is known for the packet and service packet arrival. A combined metric for selecting the next hop node, considering both energy and queue length, is illustrated in equation 6,

$$L_x = \alpha.\frac{G_q}{E_s} + \beta.\left(1 - \frac{H_x}{Q_{max}}\right) \quad (6)$$

here, L_x is known for node x metric, and the α and β are known as weighting factors. Equations 7 below illustrate the routing decision to select the next hop, N. It can be based on the node with the maximum C_x value:

$$N = argmax_x\{C_x\} \quad (7)$$

Received: August 08,2025

International Journal of Applied Mathematics

Volume 38 No. 10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

To handle the maximum count of requests, the server capacity can be determined by the below equation 8,

$$B = \frac{\sum_{x=1}^{n} \mu_x}{\lambda} \tag{8}$$

here, B is meant for the capacity to handle the request, $\sum_{x=1}^{n} \mu_x$ is meant for the amount of service rate of whole nodes, and the whole arrival rate of requests is illustrated as λ. The protocol balances energy consumption and queue length to maximize the network's lifetime and handle the maximum number of requests.

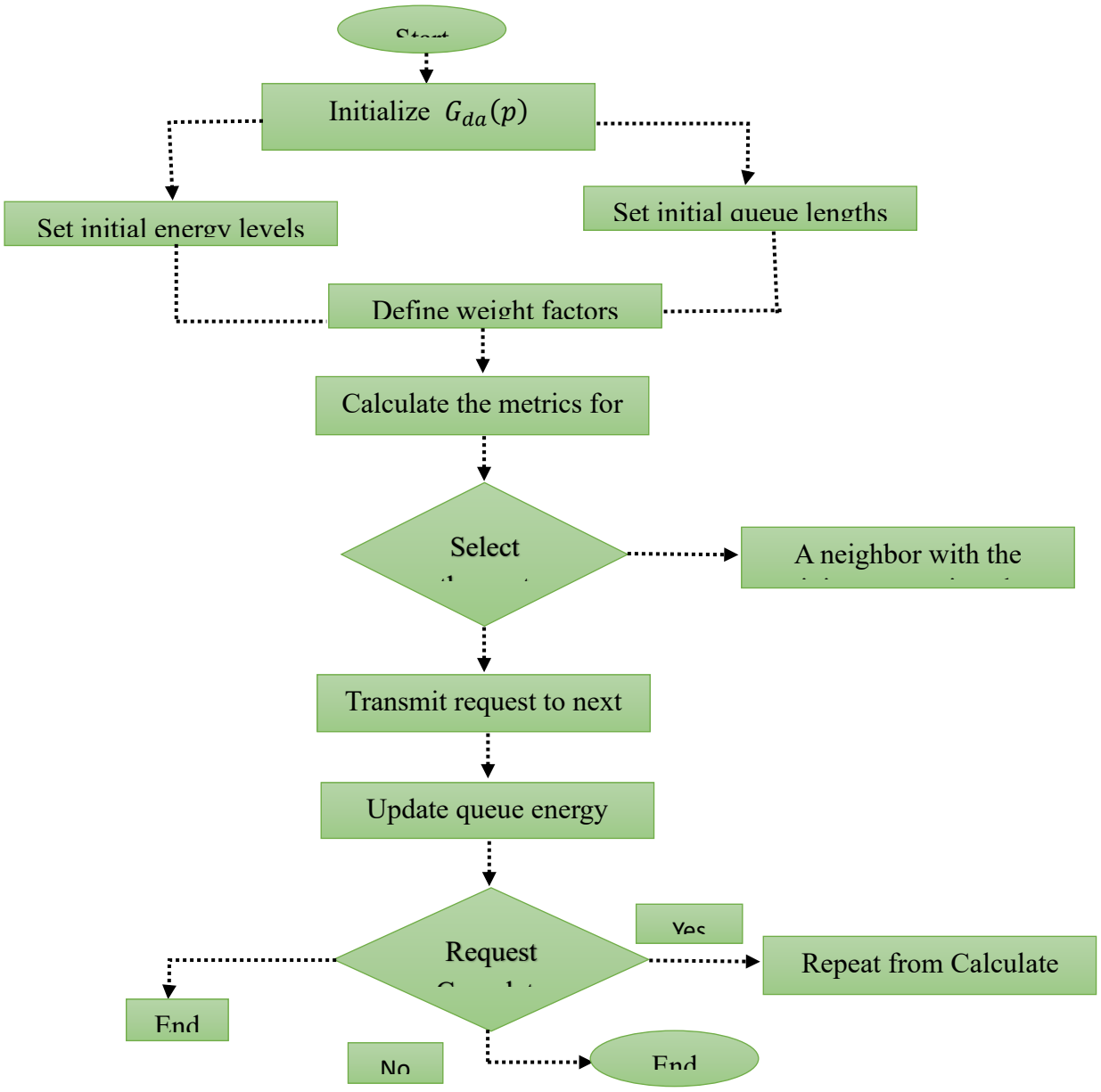

Figure 4: Flowchart Diagram of the MQEARP

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

In Figure 4, they illustrate the flowchart of the MQEARP method. This flowchart illustrates how the protocol controls energy and queue length to provide effective request processing while preventing overloads and using less energy.

3.4 OnDemand Pre-emptive Self-Scheduling (ODPSS) method

This segment mainly focuses on the on demand pre-emptive self-scheduling method, designed for container cloud tasks. In this section, we initially clarify the task, excluding the issue of the container cloud. The OnDemand service is activated as the web server receives dynamic requests, allowing tasks to await processing in the queue. To manage this influx efficiently, our Enhanced Preemptive Self-Scheduling strategy facilitates the transition of OnDemand requests into a prioritized queue, enabling the deployment of temporary VMs to alleviate the cloud server's workload.

Virtualization technology offers a successful approach to managing dynamic resources in a CC environment. Encapsulating the service in VMs and mapping it to every physical server might help manage the issue of heterogeneity and platform irrelevance of users' needs more effectively. Here, $Q = \{Q_1, Q_2, \dots, Q_N\}$ represents the system's whole physical machines. The number of physical machines is defined as N, and the physical machine stands as $Q_x (1 \le x \le N)$. The v_x is illustrated as the amount of VMs physical machines. We illustrate the VMs in the physical machine as Q_x $M_x = \{M_{x1}, M_{x2}, and\ M_{xv}\}$. Also, we organize M as the existing $R = \{R_1, R_2 \dots, R_N\}$ replaces the mapping solution after every physical machine M is arrangeM_xd. The mapping solution is illustrated as $R_{x\ when\ the\ M\ is\ arranged\ to\ Qx}$.

A physical machine's load may typically be calculated by aggregating the loads of the VM operating on it. We assume that D is the optimal observation period based on historical data. In other words, based on historical data, the monitoring zone is T from the current time. Time D can be divided into n periods according to the fluctuating physical machine load legislation. Hence, we outline $D = [(d_1 - d_0), (d_2 - d_1), \dots, (d_n - d_{n-1})]$. To time u, we refer $(to\ d_u - d_{u-1})$ as an outline. Yet the loads of VM are moderately constant each time; at that time, we are able to outline the load of u as $M(x, u)$. Furthermore, we accomplished that in D, the average value of M_x on Q_x in equation 9.

$$\overline{M_x(x, D)} = B\left(\frac{1}{D}\sum_{u=1}^{n} M(x, u) \times (d_u - d_{u-1})\right) \quad (9)$$

A physical machine's load is frequently determined by adding the loads of its VM that are executing on it following the system structure. Thus, we can infer that the actual machine Q_x load is illustrated in equation 10,

$$Q(x, D) = \sum_{y=1}^{v_x} \overline{M_x(y, D)} \quad (10)$$

Now, V requires to be deployed for the VM. We are able to compute the load of the V using pertinent data because the resources required via contemporary employment VM have previously been determined. Therefore, when V is set up to represent a real machine, each actual machine's load should be shown in equation 11,

Received: August 08,2025

International Journal of Applied Mathematics

Volume 38 No. 10s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

$$Q(x,D) = \begin{cases} Q(x,D) + M \\ Q(x,D) \end{cases} \tag{11}$$

There is typically a shift in the system burden when M is arranged to Q_x. Thus, load adjustment is required to achieve LB. Following the arrangement of M to Q_x, the load variation of mapping solution R_x in D is,

$$\sigma_x(D) = \sqrt{\frac{1}{N}\sum_{x=1}^{N}\left(\overline{Q(D)'}Q(x,D)'^2\right)} \tag{12}$$

here,

$$\overline{Q(D)'} = \frac{1}{N}\sum_{x=1}^{N} Q(x,D) \tag{13}$$

We describe the cloud request log of V's need to migrate to achieve LB in a given mapping solution to the total V. The cost divisor $\rho(R_x)$ to reach LB R_x is therefore defined for each mapping solution R_x as follows:

$$\rho(R_x) = \frac{V'}{V} \tag{14}$$

This primary goal is to request a handler within the cloud server, creating cloud request logs. $\rho(R_x)$ in LB by identifying the R_x to achieve optimal system LB. Using ODPSS, we may get the optimal mapping solution R'_x from R_x. The technique was applied to resolve the container cloud task scheduling demand. To update the probability of advantageous scheduling activities in this method, use equation 15; if not, use equation 16.

$$\begin{cases} \rho_x(R_x) = q_x(d) + \lambda_1(1 - q_x(d)) \\ \rho_y(R_x) = (1-\lambda_1)q_y(d) \quad y \neq x \end{cases} \tag{15}$$

$$\begin{cases} \rho_x(R_x) = (1-\lambda_2)q_x(e) \\ \rho_y(R_x) = \lambda_2/(s-1) + (1-\lambda_2)q_y(e) \quad y \neq x \end{cases} \tag{16}$$

The equation above has s as the number of demands, $q_x(d)$ as the probability that demand x will be chosen at time d, and λ_1 and λ_2 as the reward and punishment factors, respectively. Through several cycles, the reward-penalty approach can find the roughly ideal solution to scheduling demands. Its probability update randomness is robust, though. Additionally, the likelihood reward granularity lacks distinction between various acts. These cause issues with high unpredictability and poor directiveness in the result space when looking for the best demands, which slows down the method's convergence rate.

A reward base odd $f(r_x, e_y)$ accompanying via nodes and tasks is added to every scheduling demand by the container cloud task scheduling features to improve the algorithm's optimisation capability. Equation 17 is deployed to update the scheduling demand possibility for schedules that align with the system goal.

$$\begin{cases} q_{e_y}^{r_x}(d+1) = q_{e_y}^{r_x}(d) + \lambda_1 f(r_x, e_y) \\ q_{e_u}^{r_x}(d+1) = q_{e_u}^{r_x}(d) + \lambda_1 \left(f(r_x, e_u) - \frac{1}{E-1}\right) \\ 1 \leq u \leq E \text{ and } e_u \neq e_y \end{cases} \tag{17}$$

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

where $f(r_x, e_y)$ is the reward base probability associated to task r_x, E is the total number of nodes, λ_1 is the reward factor, and $q_{e_y}^{r_x}(d)$ is the chance of demand r_x scheduling to node e_y at time d. Equation 18 updates the scheduling demands probability for the specified action; equation 9 is utilized in other cases. The odds of each scheduling demand are normalized by Equation 18, which is

$$\sum_{d=1}^{E} q_{e_y}^{r_x}(d+1) = 1 \tag{18}$$

Equation 19 is deployed to update the scheduling demands possibility for schedules that are not in line with the system goal.

$$\begin{cases} q_{e_y}^{r_x}(d+1) = q_{e_y}^{r_x}(d) - \lambda_2 q_{e_y}^{r_x}(d) \\ q_{e_u}^{r_x}(d+1) = q_{e_u}^{r_x}(d) + \lambda_2 \left(\frac{1}{E-1} - q_{e_u}^{r_x}(d) \right) \\ 1 \le u \le E \ and \ e_u \ne e_y \end{cases} \tag{19}$$

The other notations are the same as in equation9, and λ_2 is the penalty factor regarding the chosen scheduling operation. If the scheduling demand probability needs to be updated, use equation 18 (upper); otherwise, use equation 19 (lower).

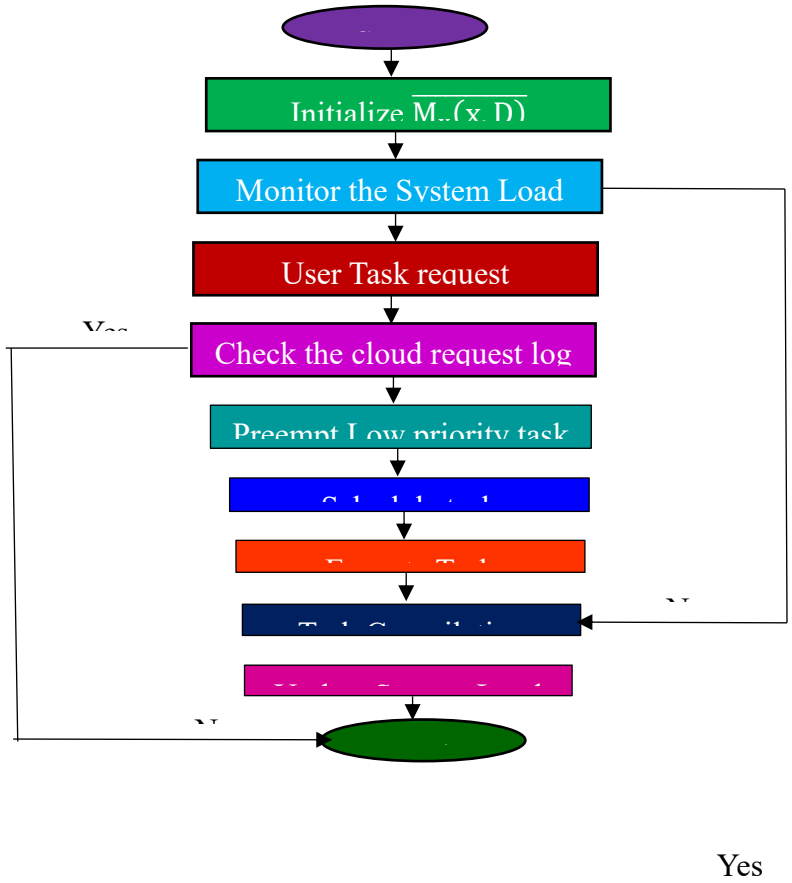

Figure 5 Flowchart Diagram of the proposed method

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Figure 5 is the flow chart of the method, which can generate the most reasonable probabilistic reward and punishment behaviors based on the latest state of the resource environment, which significantly increases the adaptability of the learning automaton in the dynamic scheduling scenario of container cloud tasks.

In this section, the framework presents a maximum order energy-aware request protocol to effectively prioritize the order and analyze the absolute average rate of server processing requests based on the maximum number.

4. Result and Discussion

In this section, the maximum number of requests the server handles can be estimated using an energy-aware routing protocol to improve scheduling performance and resource allocation targeting various critical user tasks. Furthermore, the framework enables the identification, evaluation, comparison, and analysis of the technologies proposed in cloud environments. Nevertheless, the self-programming performance of this method can be improved by using the C# language to describe the test results of the model. Moreover, the existing techniques ACO, SA-FFO, and GPFT can be compared with the proposed method. In addition, the proposed TTIR, PSO, MQEARP, and ODPSS techniques are compared, and the performance analysis is performed in terms of time complexity, cost efficiency, energy consumption, delay tolerance, resource allocation, and self-scheduling performance.

Table 2. Simulation Parameter Description

Parameter	Variable
Simulator	Visual Studio 6.0
Simulation area	$500\text{m} \times 500\text{m}$ and $1000\text{m} \times 1000\text{m}$
Language	C#
The Number of Requests	5000
Speed of Request	35
Communication range	370
Data Rate	5
Packet Interval	0.3
Simulation time (s)	250
The initial energy of the request	3 Joules
Transmitting Request	47dBm (LB), 25dBM (UR)
Packet transmission size	16.64Kbps
Total Simulation time	2000sec

Received: August 08,2025

International Journal of Applied Mathematics
Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Comparison of simulation results using C# language-based Visual Studio 6.0 simulator with other analytical performance results using the proposed ODPSS algorithm described in simulation parameters Self-programming performance simulator in Visual Studio 6.0. In addition, the Load Balancer (LB) can improve the performance of User Requests (UR) by forwarding requests forwarded in self-scheduling.

Table 3. The Comparison Method of Performance Evaluation

Number of Requests	Methods	Performance Evaluation			
		Energy Consumption	Resource Allocation	Self-Scheduling	Time Complexity
500	ACO	81.12	84.15	85.47	34.2
1000	SA-FFO	83.7	86.23	87.9	31.4
2000	GPFT	85.3	88.46	90.4	27.6
3000	SPSO-TCS	88.1	90.47	92.6	24.9
4000	SPSO-TCS	90.7	92.6	93.8	20.9
5000	ODPSS	93.9	94.7	96.9	16.8

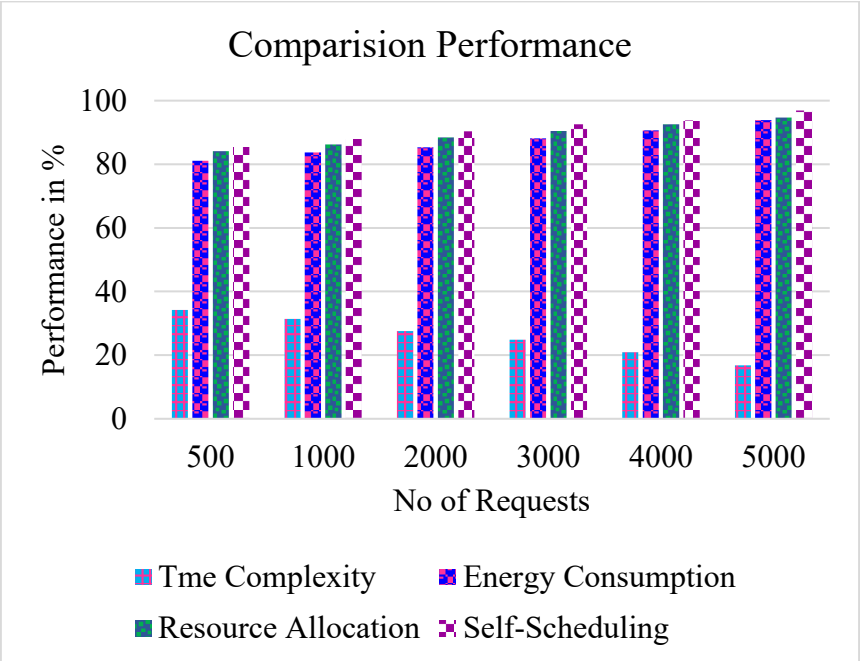

Figure 6. Analysis of Comparison Performance

The performance evaluation methods summarized in Table 3 and Figure 6 demonstrate that creating cloud request logs enhances the proposed ODPSS method within the cloud

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

environment. By analyzing these four parameters, which are performance evaluation, energy consumption, resource allocation, self-scheduling, and time complexity, the proposed method can be compared with existing methods to improve resource allocation. The parameter estimation improves the self-scheduling efficiency by 96.9% of the proposed ODPSS method and over the previous techniques ACO, SA-FFO, GPFT, SPSO-TCS and SPSO-TCS (85.47%, 87.9%, 89.4%, 91.3% and 93.8%).

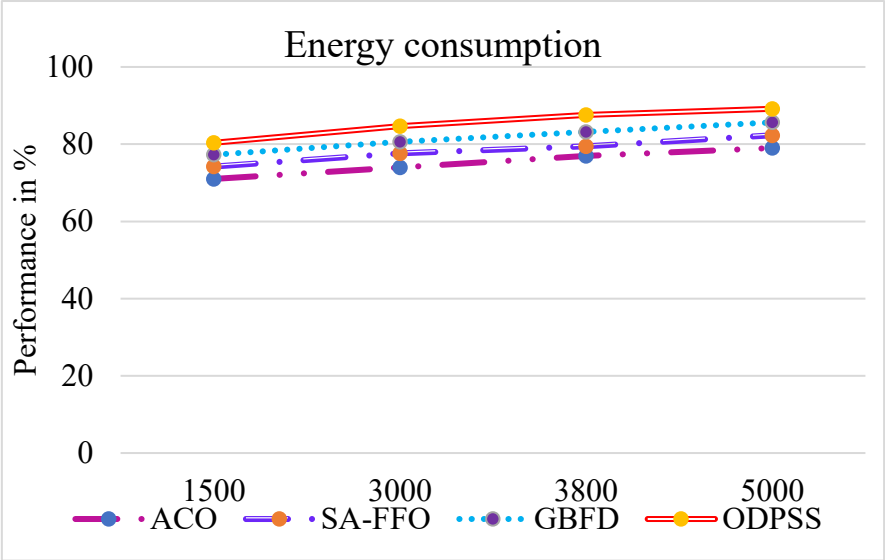

Figure 7. Analysis of Energy Consumption

As shown in Figure 7, using the energy consumption analysis method, the proposed ODPSS method in the cloud environment generates cloud request records with self-scheduling based on queue request protocol. Furthermore, when the proposed ODPSS method was used to estimate the energy consumption, the performance increased to 89.16%. Moreover, the energy consumption performance is 79%, 82.3%, and 85.64% compared to the proposed method using conventional techniques such as ACO, SA-FFO, and GBFD.

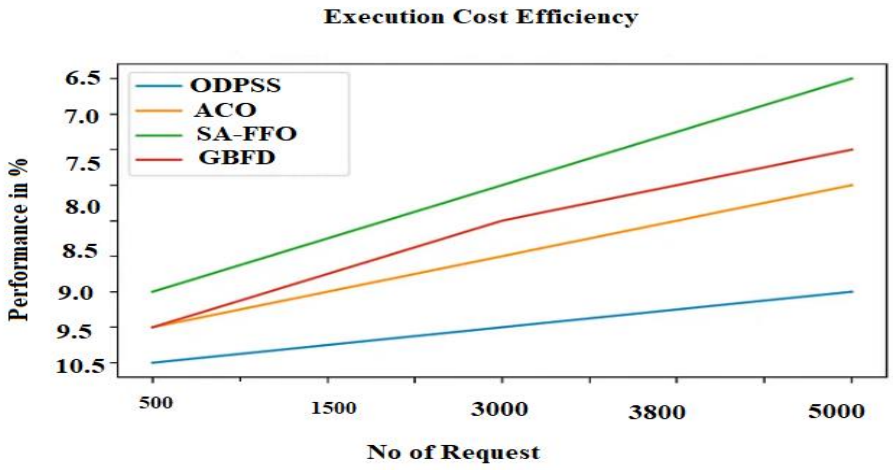

Figure 8. Analysis of Execution Cost Efficiency

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The proposed ODPSS method in the cloud environment uses processing cost efficiency analysis to generate cloud request records with self-scheduling based on queue request protocol, as illustrated in Figure 8. Furthermore, the implementation cost efficiency is 10.1%, 9.7%, and 8.1% compared to the proposed method using conventional techniques such as ACO, SA-FFO, and GBFD. Moreover, when the implementation cost was evaluated using the proposed ODPSS method, the efficiency was 7.2%.

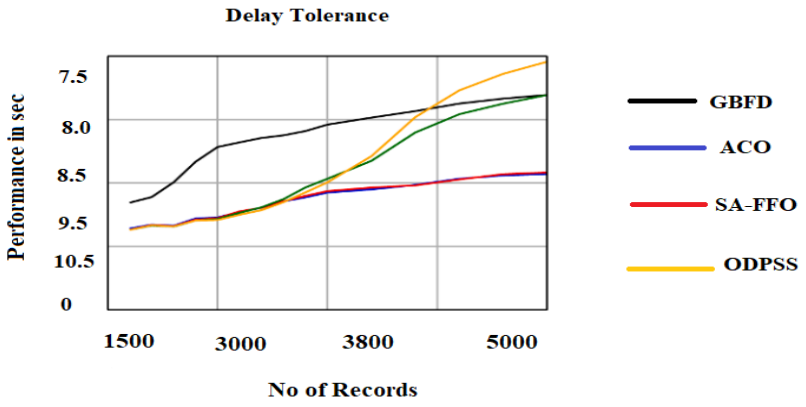

Figure 9. Analysis of Delay Tolerance

The ODPSS method proposed for the cloud environment employs processing delay tolerance analysis to create cloud request logs with self-scheduling based on a queue request protocol, as demonstrated in Figure 9. In addition, the implementation delay tolerance was 12.9%, 9.4%, and 6.4% compared to conventional techniques like ACO, SA-FFO, and GBFD. Furthermore, when assessing the implementation cost with the ODPSS method, the delay tolerance request logs were 5.1%.

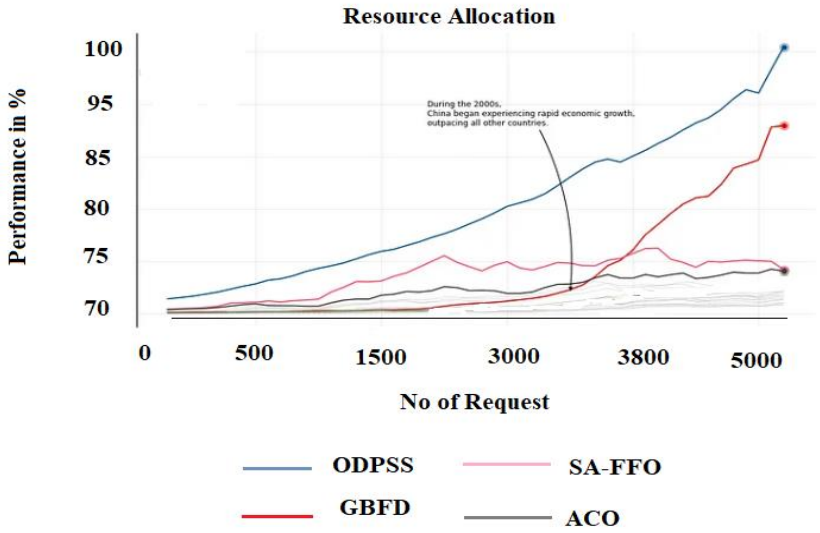

Figure 10. Analysis of Resource Allocation

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The ODPSS method for cloud environments utilizes processing resource allocation analysis to create a cloud request log with self-scheduling based on the queue request protocol, as illustrated in Figure 10. It improves resource allocation efficiency by 79.8%, 81.7%, and 85.9% compared to traditional methods like ACO, SA-FFO, and GBFD. In addition, the ODPSS method achieves a 92.5% rate of analyzing resource allocation request records when evaluating processing.

Figure 11. Analysis of Self-Scheduling Performance

As shown in Figure 11, the cloud queue automatically indexes and creates request records based on the request protocol. In addition, compared to traditional technology, the improvements in self-scheduling capabilities were 81.8%, 87.5%, and 90.7%, respectively. The proposed ODPSS method for the cloud environment is processed using self-scheduling analysis. By analyzing the various priorities of user tasks, the performance of the proposed ODPSS approach can improve productivity and reduce risks for server workloads. Furthermore, evaluating the proposed ODPSS approach improves the self-scheduling efficiency by 96.9% by generating cloud request logs from server handlers.

5. Conclusion

In conclusion, ODPSS technology collects virtual fetch request tasks from the load-balancing request handler on the cloud server and generates the corresponding logs. Furthermore, it uses a request protocol-based TTIR technique to estimate the task completion time in maximum queue scheduling. The proposed method incorporates the PSO algorithm to estimate server processing parameters, including energy consumption, processing time, queue length, demand, and delay dependence. In addition, it integrates the self-scheduling of maximum queue tasks according to request protocol to complete LB in a cloud environment. This technique is enhanced by the MQEARP framework, which provides an absolute average velocity based on the maximum number of features. Finally, the proposed ODPSS approach's performance can improve productivity and server resource allocation while considering different priorities of user tasks. Therefore, the cloud request records are generated and evaluated based on the performance analysis of the proposed approach, including delay tolerance, energy

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

consumption, cost efficiency of operation, self-scheduling, time complexity, and number of resource allocations. By analyzing different approaches to user tasks, the proposed ODPSS method outperforms the previous ACO, SA-FFO, and GBFD techniques by 96.9%.

Reference

1. S. M. Alamouti, F. Arjomandi and M. Burger, "Hybrid Edge Cloud: A Pragmatic Approach for Decentralized Cloud Computing," in IEEE Communications Magazine, vol. 60, no. 9, pp. 16-29, September 2022, doi: 10.1109/MCOM.001.2200251.
2. Li, P.; Wang, H.; Tian, G.; Fan, Z. Towards Sustainable Cloud Computing: Load Balancing with Nature-Inspired Meta-Heuristic Algorithms. Electronics 2024, 13, 2578. https://doi.org/10.3390/electronics13132578
3. Pradhan, A., & Bisoy, S. K. (2022). A novel load balancing technique for cloud computing platform based on PSO. Journal of King Saud University - Computer and Information Sciences, 34(7), 3988-3995. https://doi.org/10.1016/j.jksuci.2020.10.016
4. Hajvali, M., Adabi, S., Rezaee, A., & Hosseinzadeh, M. (2023). Decentralized and scalable hybrid scheduling-clustering method for real-time applications in volatile and dynamic Fog-Cloud Environments. Journal of Cloud Computing, 12(1), 1-35. https://doi.org/10.1186/s13677-023-00428-4
5. Belgaum, M. R., Soomro, S., Alansari, Z., Musa, S., Alam, M., & Mohd, M. (2019). Load Balancing with preemptive and non-preemptive task scheduling in Cloud Computing. ArXiv. https://doi.org/10.1109/ICETSS.2017.8324145
6. V. Mohammadian, N. J. Navimipour, M. Hosseinzadeh, and A. Darwesh, "Fault-Tolerant Load Balancing in Cloud Computing: A Systematic Literature Review," in IEEE Access, vol. 10, pp. 12714-12731, 2022, doi: 10.1109/ACCESS.2021.3139730.
7. M. A. Shahid, N. Islam, M. M. Alam, M. M. Su'ud and S. Musa, "A Comprehensive Study of Load Balancing Approaches in the Cloud Computing Environment and a Novel Fault Tolerance Approach," in IEEE Access, vol. 8, pp. 130500-130526, 2020, doi: 10.1109/ACCESS.2020.3009184.
8. Masdari, M., Salehi, F., Jalali, M. et al. A Survey of PSO-Based Scheduling Algorithms in Cloud Computing. J Netw Syst Manage 25, 122–158 (2017). https://doi.org/10.1007/s10922-016-9385-9.
9. L. -H. Hung, C. -H. Wu, C. -H. Tsai and H. -C. Huang, "Migration-Based Load Balance of Virtual Machine Servers in Cloud Computing by Load Prediction Using Genetic-Based Methods," in IEEE Access, vol. 9, pp. 49760-49773, 2021, doi: 10.1109/ACCESS.2021.3065170.
10. D. A. Shafiq, N. Z. Jhanjhi, A. Abdullah and M. A. Alzain, "A Load Balancing Algorithm for the Data Centres to Optimize Cloud Computing Applications," in IEEE Access, vol. 9, pp. 41731-41744, 2021, doi: 10.1109/ACCESS.2021.3065308.
11. Jitendra Kumar Samriya, Narander Kumar ⇑Jitendra Kumar Samriya, Narander Kumar, "An optimal SLA based task scheduling aid of hybrid fuzzy TOPSIS-PSO algorithm in cloud environment," 2214-7853/ 2020.https://doi.org/10.1016/j.matpr.2020.10.082.

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

12. M. Sohani and S. C. Jain, "A Predictive Priority-Based Dynamic Resource Provisioning Scheme with Load Balancing in Heterogeneous Cloud Computing," in IEEE Access, vol. 9, pp. 62653-62664, 2021, doi: 10.1109/ACCESS.2021.3074833.
13. M. Z. Nayyer et al., "LBRO: Load Balancing for Resource Optimization in Edge Computing," in IEEE Access, vol. 10, pp. 97439-97449, 2022, doi: 10.1109/ACCESS.2022.3205741.
14. H. Shen and L. Chen, "A Resource Usage Intensity Aware Load Balancing Method for Virtual Machine Migration in Cloud Datacenters," in IEEE Transactions on Cloud Computing, vol. 8, no. 1, pp. 17-31, 1 Jan.-March 2020, doi: 10.1109/TCC.2017.2737628.
15. Neetu Sharma, Sonal, Puneet Garg, " Ant colony-based optimization model for QoS-based task scheduling in cloud computing environment," Measurement: Sensors, Volume 24, 2022, 100531, ISSN 2665-9174, https://doi.org/10.1016/j.measen.2022.100531.
16. Saurabh Singhal; Ashish Sharma; Anushree; Pawan Kumar Verma; Mohit Ku, "Energy Efficient Load Balancing Algorithm for Cloud Computing Using Rock Hyrax Optimization," Published in: IEEE Access (Volume: 12), Page(s): 48737 - 48749, Date of Publication: 21 March 2024, DOI: 10.1109/ACCESS.2024.3380159.
17. A. Kishor, R. Niyogi, A. T. Chronopoulos and A. Y. Zomaya, "Latency and Energy-Aware Load Balancing in Cloud Data Centers: A Bargaining Game Based Approach," in IEEE Transactions on Cloud Computing, vol. 11, no. 1, pp. 927-941, 1 Jan.-March 2023, doi: 10.1109/TCC.2021.3121481.
18. S. Souravlas, S. D. Anastasiadou, N. Tantalaki and S. Katsavounis, "A Fair, Dynamic Load Balanced Task Distribution Strategy for Heterogeneous Cloud Platforms Based on Markov Process Modeling," in IEEE Access, vol. 10, pp. 26149-26162, 2022, doi: 10.1109/ACCESS.2022.3157435.
19. B. Kruekaew and W. Kimpan, "Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm with Reinforcement Learning," in IEEE Access, vol. 10, pp. 17803-17818, 2022, doi: 10.1109/ACCESS.2022.3149955.
20. Aggarwal, A., Dimri, P., Agarwal, A. and Bhatt, A. (2021), "Self-adaptive fruit fly algorithm for multiple workflow scheduling in the cloud computing environment," Kybernetes, Vol. 50 No. 6, pp. 1704-1730. https://doi.org/10.1108/K-11-2019-0757.
21. Jafarnejad Ghomi, E., Masoud Rahmani, A., & Nasih Qader, N. (2019). Service load balancing, task scheduling, and transportation optimization in cloud manufacturing by applying a queuing system. Enterprise Information Systems, 13(6), 865–894. https://doi.org/10.1080/17517575.2019.1599448.
22. M. Menaka, K.S. Sendhil Kumar, "Supportive particle swarm optimization with time-conscious scheduling (SPSO-TCS) algorithm in cloud computing for optimized load balancing," International Journal of Cognitive Computing in Engineering, Volume 5, 2024, Pages 192-198, ISSN 2666-3074, https://doi.org/10.1016/j.ijcce.2024.05.002.
23. S.Rekha, C.Kalaiselvi, "Multi Level Queue Scheduling With Particle Swarm Optimization (Mlqs-Pso) Of Vms in Queueing Heterogeneous Cloud Computing Systems," International

Received: August 08,2025

Volume 38 No. 10s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Journal of Recent Technology and Engineering (IJRTE), ISSN: 2277-3878, Volume-8 Issue-5, January 2020, DOI:10.35940/ijrte.E6080.018520.

24. Zi-Ren Wang, Xiao-Xiang Hu, Peng Wei, Bo Yuan, "An improved particle swarm optimization algorithm for scheduling tasks in cloud environment," First published: 12 March 2024, https://doi.org/10.1111/exsy.13529.
25. Kumar, M., Sharma, S.C. PSO-based novel resource scheduling technique to improve QoS parameters in cloud computing. Neural Comput & Applic 32, 12103–12126 (2020). https://doi.org/10.1007/s00521-019-04266-x
26. Shikha Chaudhary, Vijay Kumar Sharma, R. N. Thakur, Amit Rathi, Pramendra Kumar, Sachin Sharma, "Modified Particle Swarm Optimization Based on Aging Leaders and Challengers Model for Task Scheduling in Cloud Computing," First published: 23 June 2023 https://doi.org/10.1155/2023/3916735.
27. M. Menaka, K.S. Sendhil Kumar, "Supportive particle swarm optimization with time-conscious scheduling (SPSO-TCS) algorithm in cloud computing for optimized load balancing," International Journal of Cognitive Computing in Engineering, Volume 5, 2024, Pages 192-198, ISSN 2666-3074, https://doi.org/10.1016/j.ijcce.2024.05.002.
28. Sirisha Potluri, Katta Subba Rao, "Optimization model for QoS based task scheduling in cloud computing environment," Indonesian Journal of Electrical Engineering and Computer Science Vol. 18, No. 2, May 2020, pp. 1081~1088, ISSN: 2502-4752, DOI: 10.11591/ijeecs. v18.i2. pp1081-1088.
29. Farid, M.; Latip, R.; Hussin, M.; Abdul Hamid, N.A.W. A Survey on QoS Requirements Based on Particle Swarm Optimization Scheduling Techniques for Workflow Scheduling in Cloud Computing. Symmetry 2020, 12, 551. https://doi.org/10.3390/sym12040551
30. Khaledian, N., Khamforoosh, K., Akraminejad, R. et al. An energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment. Computing 106, 109–137 (2024). https://doi.org/10.1007/s00607-023-01215-4
31. Sharma, G., Miglani, N. & Kumar, A. PLB: a resilient and adaptive task scheduling scheme based on multi-queues for cloud environment. Cluster Comput 24, 2615–2637 (2021). https://doi.org/10.1007/s10586-021-03280-w
32. Ali, A.; Iqbal, M.M.; Jamil, H.; Qayyum, F.; Jabbar, S.; Cheikhrouhou, O.; Baz, M.; Jamil, F. An Efficient Dynamic-Decision Based Task Scheduler for Task Offloading Optimization and Energy Management in Mobile Cloud Computing. Sensors 2021, 21, 4527. https://doi.org/10.3390/s21134527
33. Z. Deng, Z. Yan, H. Huang and H. Shen, "Energy-Aware Task Scheduling on Heterogeneous Computing Systems with Time Constraint," in IEEE Access, vol. 8, pp. 23936-23950, 2020, doi: 10.1109/ACCESS.2020.2970166.
34. Xu, H., Xu, S., Wei, W. et al. Fault tolerance and quality of service aware virtual machine scheduling algorithm in cloud data centers. J Supercomput 79, 2603–2625 (2023). https://doi.org/10.1007/s11227-022-04760-5.
35. Wu, S.; Wu, Z.; Wu, X.; Tao, J.; Gu, Y. Queuing-Based Federation and Optimization for Cloud Resource Sharing. Information 2022, 13, 361. https://doi.org/10.3390/info13080361.

Received: August 08,2025