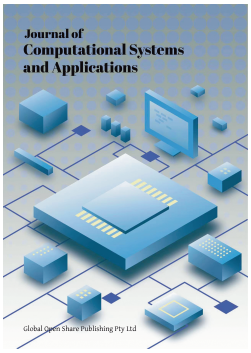
GOSP

Journal of Computational Systems and Applications

https://jcsa.gospub.com/index.php/jcsa

Global Open Share Publishing



Article

A Hybrid Approach Combining Ant Colony Optimization and Simulated Annealing for Cloud Resource Scheduling

Saurabh Singhal*

Department of CSE, Greater Noida Institute of Technology (Engg. Institute), Greater Noida, Uttar Pradesh, India.

*Corresponding author: Saurabh Singhal, me.ssaurabh@gmail.com

Abstract

Cloud computing is imperative to schedule efficiently for tasks and resources to assure performance, reduction of costs, and service-level agreement. Traditional methods cannot balance this complexity, resulting in the conception of a hybrid model that will be based on the integration of ACO with SA. Here, the former algorithm applies the collective intelligence of ACO, coupled with positive feedback, to achieve better quality solutions and escapes the local optimum of the latter. This algorithm runs in two phases: The initial solution is generated using the ACO, and the solution is refined using SA. Simulated experiments within a simulated cloud environment of CloudSim showed that this hybrid approach succeeds in minimizing makespan, reducing energy consumption, and optimizing the cost for different workloads. The ACO-SA algorithm heralds a promising direction toward highly efficient management of cloud resources and opens up further research directions in hybridizing other complementary algorithms.

Keywords

Cloud computing, Resource scheduling, Ant colony, Simulated annealing, Cost

Article History

Received: 26 January 2025 Revised: 16 April 2025

Accepted: 12 May 2025 Available Online: 9 June 2025

Copyright

© 2025 by the authors. This article is published by the Global Open Share Publishing Pty Ltd under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0): https://creativecommons.org/licenses/by/4.0/

1. Introduction

Cloud computing has transformed how organizations manage and scale their computing resources in an adaptive, efficient, and scalable paradigm [1]. There are a variety of cloud service providers, and more clients demand high-performance services that drive this shift. However, identifying the right method to share resources is paramount so that all different kinds of consumers can reap benefits from them. Resource scheduling should be carried out in a proper manner so that Quality of Service (QoS), pricing, and performance are improved in cloud computing [2].

Resource scheduling in the cloud refers to the allocation of jobs or workloads to computing resources like virtual machines, servers, or storage units for optimal performance regarding time, energy usage, and cost while also satisfying constraints, such as adhering to deadlines and ensuring resources are available [3,4]. This is a hard problem with no easy solution because cloud resources are not all the same and change all the time, and users' needs are hard to predict. Different types of computer methods have been studied, such as heuristic approaches, advanced metaheuristics, and mixed algorithms. Traditional scheduling methods, such as First Come, First Serve (FCFS), Shortest Job First (SJF), and Round Robin (RR), are simple. However, they don't always work well in cloud systems because they can't to adapt to dynamic changes.

Metaheuristics such Genetic Algorithms (GA), Particle Swarm Optimization (a), and Ant Colony Optimization (ACO) could simplify the planning of how to exploit cloud resources. These methods find almost flawless results in a reasonable time span using random and iterative search. Examining how ants locate food and developing a fast and effective mechanism led to an algorithm known as ACO [5]. The ACO algorithm might either get caught in local optima or converge too quickly. A statistical optimization method called Simulated Annealing (SA) helps to get out of local optima by accepting worse solutions with a certain chance [6]. This aspect of SA enables algorithms such as ACO to perform better searches in the world.

The study provides a hybrid algorithm that combines ACO and SA to solve scheduling problems with cloud resources. The method uses both SA's fine-tuning and global search tools, as well as ACO's ability to come up with great first answers. To make sure the algorithm can adapt to the changing and unique nature of cloud environments, the integration finds a good mix between exploring and using. Adding the probabilistic acceptance criterion from SA to the update method of pheromones in ACO assists the search. The mixed approach, thus, would be able to work for cloud resource planning. It aims at improving the quality of solutions found with respect to trade-offs such as discovery-exploitation and speed of convergence.

This work will propose a hybrid algorithm of ACO-SA for effective resource scheduling within the cloud with considerations of the standards of QoS factors such as response time, cost, and energy efficiency. Benchmarks will use datasets with realistic scenarios in comparing the performances against ACO, SA, and other current top methods. Some important parameters are going to be analysed for real-world usage in cloud computer systems. Hybrid ACO-SA algorithm is one of the resource division strategies used in complex problems, and this shows the great potential that could be gained if two successful algorithms were put together. This work opens the door for further progress in mixed optimization methods and their use in managing cloud resources.

The following are main contribution of the work:

A hybrid algorithm improves algorithmic design by making the best use of cloud resources by using probabilistic tuning and pheromone-guided search to schedule them. The suggested approach works better than tried-and-true methods in terms of solution quality, rate of convergence, and ability to change to different workloads. This work therefore allows mixed algorithms to be used in real cloud platforms by providing useful information about how these algorithms are made and how to make them work best for allocating cloud resources.

2. Related Work

Task scheduling in cloud computing involves binding user tasks to connected resources based on task schedulers' decisions based on metrics. Several algorithms have been proposed for scheduling tasks in grid and cloud computing systems. Previous research indicates that heuristics-based algorithms are more suitable than traditional algorithms for task scheduling in cloud computing, as it is an NP-complete problem [7].

Authors in [8] have proposed task scheduling using genetic algorithms where the motive had been to reduce the response time without having much success due to dynamic environment. The authors in [9] presented a parallel particle swarm optimization (PPSO) algorithm in order to reduce average execution time. In [10], the authors have presented an ACO based task scheduling technique with few improvements with respect to FCFS and Round Robin. Pradhan et al. [11] introduced the Modified Round Robin Algorithm for the resource allocation strategy in cloud computing, which aims to reduce turnaround and waiting time but exhibits low throughput.

The genetic-based task scheduling approach was presented in [12] by authors with the objective of minimizing makespan and cost of task execution with maximum resource utilization. The idea of IDEA in [13] was introduced, which is the enhanced differential evolution algorithm for task scheduling and resource allocation in the cloud, by the

combination of DEA and Taguchi methods to explore and exploit solutions. In [14], the authors have proposed a super genetic algorithm, named N-GA for decreasing makespan and execution time through evolutionary genetic algorithms and behavioural modelling.

In [15], the authors proposed a modified PSO algorithm with fuzzy theory for task scheduling to minimize makespan. Jobs are assigned using one-to-one mapping and transferring the shortest job to the fastest processor, thereby improving PSO behaviour. In [16], the authors proposed an enhanced version of Multi-Verse Optimizer (EMVO) for better scheduling of tasks. The authors of [17] proposed Hybrid Electro Search with a Genetic Algorithm (HESGA) as an approach to make task scheduling better in terms of performance, load balancing, resource utilization, and multi-cloud cost. A modified Henry Gas Solubility Optimization (HGSO) based on Whale Optimization Algorithm (WOA) and comprehensive opposition-based learning (COBL) for optimal task scheduling was discussed in [18]. In [19], the authors proposed efficient task scheduling in cloud computing using a hybrid machine learning (RATS-HM) for resource allocation security and efficient task scheduling in cloud computing. The authors, in [20], presented a Dynamic Weighted Round-Robin (DWRR) algorithm for improving the performance of task scheduling considering resource competencies, task priorities, and duration.

They further proposed Hybrid Particle Swarm Parallel Ant Colony Optimization for task execution delay. The authors in [21] applied Non-Dominated Classification Genetic Algorithm, named NSGA-III, to cloud task scheduling with objectives such as execution time, power consumption, energy, and cost. In [22], the authors combined Enhanced Scheduling Efficiency Algorithm with Levy Flight Algorithm. In [23], a hybrid weighted ant colony optimization (HWACO) was proposed and in [24] authors tried to overcome task scheduling problems of cloud computing by a combination of GA and gravitational emulation local search algorithm (GELS). The authors in [25] proposed an improved Dynamic Johnson Sequencing Algorithm for efficient resource scheduling for distributed overload. In [26], the authors proposed a dynamic weighted round-robin strategy, focusing on even more efficient cloud-based task allocation using Particle swarm and Ant colony.

A summary of the work presented in the related work is presented in Table 1.

Table 1. Literature survey for resource scheduling

Ref. No.	Algorithm used	QoS Parameters	Advantages	Limitations
[8]	Genetic Algorithm	Response time	Minimizing response time	Comparison with static algorithm
[9]	Parallel Particle Swarm Optimisation	Execution time	Better outcomes for medical queries	Considered medical queries only as jobs
[12]	Genetic Algorithm	Makespan, throughput and load balancing.	Improved performance	Does not consider service level agreement
[15]	Pparticle swarm optimization and fuzzy theory	Makespan, improvement ratio, imbalance degree, efficiency, and total execution time	Faster and high convergent	Fault tolerance environment not considered
[16]	Multi-Verse Optimizer	Makespan time and resource utilization.	Improved performance	Comparison with two algorithm
[20]	Ant Colony Optimization and PSO	Execution and waiting time, system throughput, and resource utilization.	Improved performance	Similar kind of metaheuristic algorithm used
[21]	NSGA-III	Cost, Execution time, Energy Consumption	Improved performance	Relatively high runtime compared to the NSGAII
[22]	Wild Horse Optimization with Levy Flight Algorithm	Task scheduling time, scheduling cost, and resource utilization	Improved performance	Only jobs are varied, VM are fixed
[23]	Hybrid Weighted Ant Colony Optimization	Makespan and cost	Improved performance	Two QoS parameters considered
[24]	Genetic algorithm and gravitational emulation local search	Execution Time and Resource Utilization	Improved performance	High run time

3. Proposed Methodology

The proposed methodology is about the designing of a hybrid algorithm using ACO and SA by solving the cloud resource scheduling problem. This section discusses problem formulations, an in-depth explanation of ACO and SA, their integration, and structures of the hybrid algorithm as well as basic key equations and mechanisms for the hybrid.

3.1 Problem Formulation

The cloud resource scheduling problem involves assigning tasks $T = \{T_1, T_2, \cdots, T_n\}$ to a set of resources $R = \{R_1, R_2, \cdots, R_m\}$, aiming to optimize certain objectives such as execution time, cost, and energy consumption, while satisfying constraints like deadlines and resource capacities. Each task Ti is characterized by the following attributes:

- Task Length (L_i): Represents the total number of instructions in millions (Million Instructions - MI) that must be executed.
- Required CPU Capacity (C_i): Defines the amount of processing power needed, typically measured in MIPS (Million Instructions Per Second).
- Storage Requirement (S_i): Represents the amount of storage (in GB) required for execution.
- Memory Requirement (M_i): Specifies the RAM needed to execute the task.
- Bandwidth Requirement (B_i): Indicates the network bandwidth (in Mbps) required for communication and data transfer.
- Deadline (D_i): Specifies the maximum allowable completion time for the task.

These characteristics directly influence the scheduling process, as tasks with higher CPU, memory, or storage demands require more efficient resource allocation to minimize delays and SLA violations.

Cloud resources $R = \{R_1, R_2, \cdots, R_m\}$ consist of a set of heterogeneous virtual machines (VMs) running on physical hardware. Let $V = \{V_1, V_2, \cdots, V_m\}$ represent the available virtual machines, each defined by the following parameters:

- Processing Capacity (PC_j): The computational power of the VM in MIPS.
- Memory (RAM_j): The amount of RAM available for task execution (GB).
- Storage (ST_j): The total disk space provided by the VM (GB).
- Bandwidth (BW_j): The network bandwidth capacity for data transfer (Mbps).
- Energy Consumption Rate (E_j): The energy usage of the VM in watts per second, considering both active and idle states.

The allocation of tasks to VMs must consider these constraints to ensure efficient scheduling and avoid resource overloading.

Objectives:

The resource scheduling in cloud by the proposed algorithm is done considering the following objectives:

(1) Minimize Total Execution Time:

$$C_{\max} = \max_{i \in \{1,2,\cdots,n\}} \{C_i\} \tag{1}$$

Where C_i is the completion time of Task T_i

(2) Minimize Total Cost

$$Cost = \sum_{i=j}^{n} \sum_{j=1}^{m} x_{ij} * c_j * t_{ij} \tag{2}$$

Where,

x_{ij} : Binary variable indicating if Task T_i is assigned to Resource R_j (1 if assigned, else 0)

c_j : Cost per unit time for resource R_j

t_{ij} : Execution time of T_i on R_j

(3) Maximize Resource Utilization

$$Res_{\text{util}} = \frac{\sum_{j=1}^{m} Usage_j}{\sum_{j=1}^{m} Capacity_j} \tag{3}$$

Subjected to the following constraints:

(1) Each task must be assigned to only one resource

$$\sum_{j=1}^{m} x_{ij} = 1, \ \forall i \in \{1, 2, \cdots, n\} \tag{4}$$

(2) Resource Capacity should not exceed.

$$\sum_{j=1}^{n} x_{ij} * d \le Capacity, \ \forall i \in \{1, 2, \cdots, m\} \tag{5}$$

3.2 Ant Colony Optimization (ACO)

ACO is a bio-inspired algorithm that mimics the foraging behaviour of ants. In this algorithm, artificial ants construct solutions by probabilistically selecting resources based on pheromone levels and heuristic information.

Key Steps in ACO

(1) Pheromone Representation: The pheromone matrix $\tau = \tau_{ij}$ represents the desirability of assigning task T_i to resource R_j. Higher pheromone values indicate stronger desirability.

(2) Heuristic Information: The heuristic information η_{ij} is defined as:

$$\eta_{ij} = \frac{1}{\tau_{ij} + \varepsilon} \tag{6}$$

where ε is a small constant to avoid division by zero. It inversely relates to the execution time of task T_i on resource R_j.

(3) Solution Construction: An ant k selects resource R_j for task T_i using a probabilistic rule:

$$P_{ij}^{k} = \frac{\left(\tau_{ij}\right)^{\alpha} \left(\eta_{ij}\right)^{\beta}}{\sum_{j=1}^{m} \left(\tau_{ij}\right)^{\alpha} \left(\eta_{ij}\right)^{\beta}} \tag{7}$$

where:

- α : Pheromone importance factor,
- β : Heuristic importance factor.

(4) Pheromone Update: After all ants construct solutions, pheromones are updated to reinforce promising assignments:

$$\tau_{ij} \leftarrow (1 - \rho) * \tau_{ij} + \Delta\tau_{ij} \tag{8}$$

where:

- ρ : Evaporation rate,
- $\Delta\tau_{ij} = \sum_{k=1}^{K} \Delta\tau_{ij}^{k}$
- $\Delta\tau_{ij}^{k} = \frac{Q}{Cost^{k}}$ if ant k used (i, j), otherwise 0.

3.3 Simulated Annealing (SA)

SA is a probabilistic optimization algorithm inspired by the annealing process in metallurgy. It refines solutions by allowing controlled acceptance of worse solutions to escape local optima.

Key Steps in SA

(1) Objective Evaluation: Each solution is evaluated using the total cost and execution time. The objective function is:

$$f(solution) = w_1 \cdot C_{\max} + w_2 \cdot Cost \tag{9}$$

where w_1 and w_2 are weights balancing the objectives.

(2) Neighbourhood Search: A new solution solution'\text{solution}' is generated by making a small modification (e.g., reassigning a task).

(3) Acceptance Criterion: The acceptance of solution'\text{solution}' is based on:

$$P(accept) = \begin{cases} 1, \ if \ f(solution') < f(solution) \\ e^{\frac{\Delta f}{T}}, \ if \ f(solution') \geq f(solution) \end{cases} \quad (10)$$

where:

- $\Delta f = f(solution') - f(solution)$
- T : Current temperature.

(4) Cooling Schedule: The temperature T is gradually reduced:

$$T \leftarrow \alpha T \quad (11)$$

where $\alpha \in (0,1)$ is the cooling rate.

3.4 Hybrid ACO-SA Algorithm

Ant Colony Optimization (ACO) and Simulated Annealing (SA) are two optimization algorithms applied in solving intricate optimization problems such as cloud resource scheduling. ACO is well-suited for building initial solutions because it is probabilistic and has a positive feedback mechanism, hence the ability to explore multiple scheduling paths at once. It also provides rapid generation of feasible solutions through pheromone updates, making it appropriate for coping with the intricacy of cloud resources. ACO's dynamic flexibility enables it to change with evolving conditions, making it appropriate for kick-starting solutions where resource demand and task specifications change. Simulated Annealing is incorporated to optimize and identify the optimal solution, enabling it to avoid being trapped in local optima, apply gradient-free optimization, and optimize scheduling decisions. SA's temperature-controlled search enables improved makespan, cost, and energy efficiency beyond ACO-generated solutions.

The algorithm presented optimizes exploration and exploitation through a merge of ACO for building initial solutions and SA for solution improvement. ACO offers a diverse solution, thereby minimizing SA's time to optimal solutions. SA improves and handles local optima traps, making ACO-SA always perform better than individual techniques in makespan, cost, use of resources, and energy consumption.

The hybrid approach combines ACO's solution construction capabilities with SA's refinement process to balance exploration and exploitation. The proposed algorithm is discussed in Algorithm 1 and its process flow is shown in Figure 1.

Algorithm 1

Input:

- $\boldsymbol{T} = \{\boldsymbol{T_1}, \boldsymbol{T_2}, \cdots, \boldsymbol{T_n}\}$: Set of tasks with defined characteristics (length, CPU, memory, storage, bandwidth, deadlines).
- $\boldsymbol{V} = \{\boldsymbol{V_1}, \boldsymbol{V_2}, \cdots, \boldsymbol{V_m}\}$: Set of virtual machines (VMs) with defined resources (CPU, memory, storage, bandwidth, energy consumption).
- Parameters: Number of ants ($\boldsymbol{k}$), pheromone evaporation rate ($\boldsymbol{\rho}$), initial temperature ($\boldsymbol{T_0}$), cooling rate ($\boldsymbol{\alpha}$), and stopping criteria.

Output:

- Optimized task-to-VM assignment minimizing makespan, cost, and energy consumption, while maximizing resource utilization.

(1) Initialization:

Input task ($\boldsymbol{T}$) and VM ($\boldsymbol{V}$) characteristics.

Initialize pheromone matrix τ\tau uniformly.

Set ACO parameters: Number of ants ($\boldsymbol{k}$), Initial pheromone level, and Evaporation rate ($\boldsymbol{\rho}$).

Set SA parameters: Initial temperature ($\boldsymbol{T_0}$), Cooling rate ($\boldsymbol{\alpha}$).

Initialize adaptive controls for $\boldsymbol{\rho}$ and $\boldsymbol{\alpha}$ to adjust dynamically.

(2) ACO Solution Construction:

Task Prioritization: Ranking of tasks by urgency based on deadlines and resource demand.

Ant Solution Construction:

For each ant a = 1 to k :

For each task T_i :

Select VM V_j using a probability-based selection:

$$P\left(T_i \to V_j\right) = \frac{\tau\left(T_i, V_j\right)^{\alpha} * \eta\left(T_i, V_j\right)^{\beta}}{\sum_{V_k} \tau\left(T_i, V_j\right)^{\alpha} * \eta\left(T_i, V_j\right)^{\beta}} \tag{12}$$

τ (T_i , V_j): Pheromone level.

η (T_i , V_j): Heuristic value.

Update Initial Solution using greedy load-balancing heuristics.

Energy-Aware Adjustment: Identification of underloaded VMs for consolidation and then reassigning tasks to minimize energy consumption.

(3) Pheromone Update:

Global Update:

- For the best solution S_{best} :

$$\tau\left(T_i, V_j\right) = \left(1 - \rho\right) * \tau\left(T_i, V_j\right) + \Delta\tau \tag{13}$$

- $\Delta\tau$ **:** Inversely proportional to the objective function.

Adaptive Tuning:

- If solution diversity decreases, increase ρ for more exploration.
- If convergence slows, reduce ρ for deeper exploitation.

(4) Apply SA to Refine Solutions:

Input: Best solution from ACO.

Repeat Until Stopping Criteria Met:

- Neighbour Generation:
  - Perform intelligent neighbourhood search focusing on bottleneck tasks.
- Energy-Aware Reassignment:
  - Identify high-energy VMs and migrate tasks if energy exceeds a threshold.
- Solution Acceptance:
  - Accept a new solution if it improves the objective function.
  - If not, accept with a probability:

$$P\left(accept\right) = \mathrm{e}^{\frac{-\Delta f}{T}} \tag{14}$$

  - Δf : Change in the objective function.
- Temperature Update:

$$T = T \times \alpha \tag{15}$$

- Adjust α dynamically based on solution improvement.
- Evaluate solutions based on the objective functions

(5) Termination:

• Repeat steps 2–5 until the maximum number of iterations or a convergence criterion is met or maximum iterations are reached or there is no Improvement in the solution number of iterations.

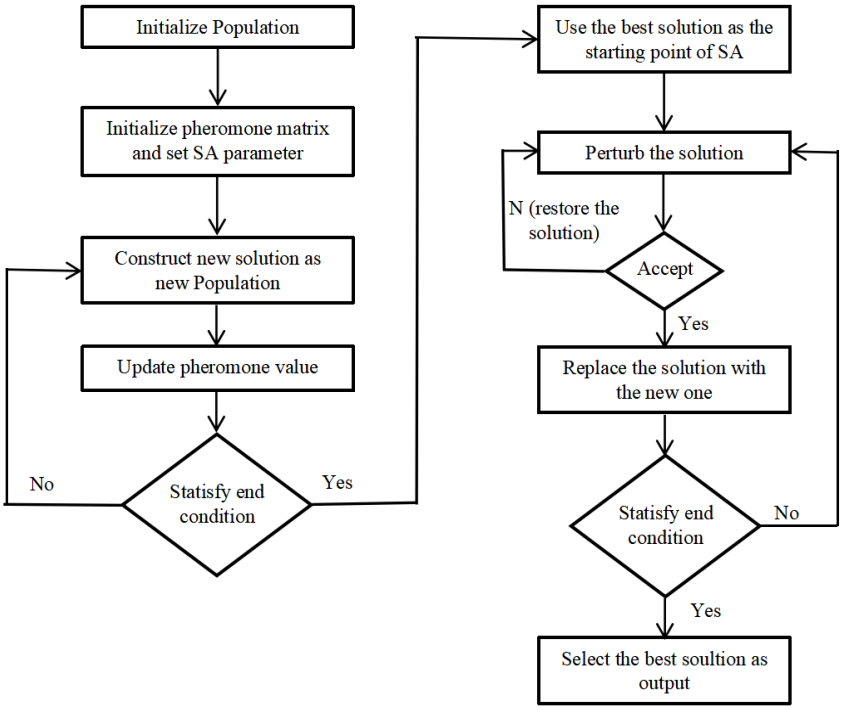


Figure 1. Process flow of the proposed algorithm

3.5 Parameter Settings and Optimization

Key parameters influencing the hybrid algorithm include:

(1) ACO Parameters:

- α, β, ρ, Q: Affect the trade-off between exploration and exploitation.

(2) SA Parameters:

- T_{init} , α , max_iterations: Influence the convergence rate.

Fine-tune these parameters to get the best performance, and there may be a grid search or some optimization technique to use for that purpose.

This methodology integrates ACO and SA to create a synergistic approach capable of efficiently solving the complex cloud resource scheduling problem. It uses ACO for the initial construction of solutions and SA for refining those solutions, ensuring robust exploration and effective convergence.

This methodology integrates ACO and SA to create a synergistic approach capable of efficiently solving the complex cloud resource scheduling problem. It uses ACO for the construction of initial solutions and SA for the refinement of these solutions, thereby ensuring robust exploration and effective convergence.

4. Experimental Setup

This section describes the experimental setup used for evaluating the proposed hybrid ACO-SA algorithm for cloud resource scheduling. It covers details on datasets, the simulation environment, evaluation metrics, and the experimental methodology, which includes key equations and configurations applied to analyse performance

4.1 Dataset Description

A synthetic benchmark datasets of cloud workload dataset were used for the experiments.

- Randomly generated task sets with varying characteristics, such as:

- Number of tasks (n): 50,100,200,50050, 100, 200, 500.
- Number of resources (m): 5,10,15, 20.
- Task execution times (t_{ij}): Uniformly distributed between [1,100] (time units).

- Resource costs (c_j): Random values between [1,10] (cost units per time unit).

The diversity in datasets ensures that the proposed algorithm is evaluated under varying workload intensities and resource heterogeneity.

4.2 Simulation Environment

The experiments were conducted in a simulated cloud environment using a resource scheduling simulator developed for this purpose. The specifications of the environment are:

The experiments were conducted on an Intel Core i7 processor with 3.6 GHz clock speed, 16 GB of RAM, and a Windows 10 operating system. This hardware configuration ensures high computational performance and efficient handling of complex scheduling tasks. The experiments allowed for accurate assessment of the hybrid ACO-SA algorithm's effectiveness across various cloud scheduling scenarios, ensuring reliable performance and compatibility. The result for the proposed algorithm are validated through two scenarios involving two data centres with 10 physical machines. In scenario I, the number of jobs is fixed at 50, 100, 200 and 500, while the number of VMs were varied. In scenario II, the number of VMs is fixed at 5, 10, 15 and 20, while the number of jobs was fixed. The details for the experiment setup of CloudSim is shown in Table 2.

Table 2. Experimental setup of CloudSim

Parameters	**Value**
Simulation Software Tool	CloudSim Software Version 3.0
Host Machine	i7-9750HF CPU @ 2.60GHz 2.59 GHz
Host Machine Memory Capacity	8-32 GB
Host Machine Operating System	Windows10, 11 and Ubuntu 22.04
Number of Virtual Machine	5-20
Virtual Machine Memory Capacity	2-8 GB
Virtual Machine Storage Capacity	100-512 GB
Number of cloudlets	100-500

VMs were categorized into multiple types (e.g., Small, Medium, Large), each varying in vCPU, RAM, storage, and bandwidth, to emulate real-world cloud instance diversity as seen in public cloud platforms. The scheduling policy used for VM allocation is time-shared, allowing multiple cloudlets to be executed concurrently on the same VM, which better reflects modern multi-tenant cloud environments. Table 3 and Table 4 provide the configuration details of Host machines and virtual machines used in the simulation.

Table 3. Configuration details of Host Machines

Host Name	**CPU Speed (GHz)**	**RAM (GB)**	**Storage (GB)**	**Bandwidth (Mbps)**	**Operating System**
Host_1	3.6	32	1000	1000	Windows 10
Host_2	3.2	8	2000	1000	Windows 11
Host_3	2.9	16	4000	1000	Ubuntu 22.04
Host_4	3	32	3000	1000	Ubuntu 22.04

Table 4. Configuration details of Host Machines

VM Type	**RAM (GB)**	**Storage (GB)**	**Bandwidth (Mbps)**	**Price ($/Hour)**
VM_Small	2	50	100	0.02
VM_Medium	4	100	200	0.05
VM_Large	8	200	400	0.1
VM_XLarge	8	400	800	0.2

4.3 Evaluation Metrics

To evaluate the performance of the hybrid ACO-SA algorithm, the following metrics are considered:

(1) Makespan (Execution Time): Makespan is the sum of time to finish all the tasks scheduled. Lower makespan means quicker execution, which results in improved system responsiveness [27].

$$C_{\max} = \max_{i \in \{1,2,\cdots,n\}} \{C_i\} \tag{16}$$

where C_i is the completion time of task T_i. Lower makespan indicates better scheduling efficiency.

(2) Total Cost [28]: Cost is the overall computational cost incurred while scheduling tasks. It depends on resource cost, and execution time. The cost of executing a task T_i on a VM V_j depends on the processing time and the cost per unit CPU cycle. Cloud providers charge based on CPU-hour or CPU-second, and the cost varies between different instance types. The cost function also includes energy cost and penalty cost for missing out the deadlines. Mathematically, it can be expressed as

$$Cost = C_{\mathrm{R}} + C_{\mathrm{E}} + C_{\mathrm{pen}} \tag{17}$$

Where,

C_{R} is the cost of VM usage

C_{E} is the Energy cost based on the power consumed by VM

C_{pen} is the penalty cost incurred if the deadline of a job is missed

$$C_{\mathrm{R}} = \sum_{i=j}^{n} \sum_{j=1}^{m} x_{ij} * c_j * t_{ij} \tag{18}$$

Equation (14) gives the cost associated with VM usage. This include processing power, RAM, storage and bandwidth of a particular VM. If the job is mapped to the resource it is calculated otherwise it is Zero.

$$C_{\mathrm{R}} = \begin{cases} x_{ij} * c_j * t_{ij} \\ 0, \ If \ Job \ is \ not \ mapped \ to \ resource \end{cases} \tag{19}$$

Where, x_{ij} is the mapping of job_i to $resource_j$, c_j is the per unit cost of $resource_j$ and t_{ij} is the execution time of job_i on $resource_j$.

For Energy Cost, Equation (15) is used. It is the summation of time period when the VM is busy or idle.

$$C_{\mathrm{E}} = \sum_{j=1}^{m} \left(P_{\mathrm{Active}_j} * t_{\mathrm{exec}_j} + P_{\mathrm{idle}_j} * t_{\mathrm{idle}_j} \right) \tag{20}$$

Where,

P_{Active} is total power consumed by a VM while executing the job

P_{idle} is total power consumed by a VM while it is idle

t_{exec} is the execution time of the job mapped on the VM

t_{idle} is the idle time of the VM

The SLA (Service Level Agreement) cost is introduced to penalize scheduling decisions that violate deadline constraints. Mathematically it is expressed as:

$$C_{\mathrm{open}} = \sum_{k}^{n} P_{\mathrm{SLA}} * t_{\mathrm{delay}_k} \tag{21}$$

Where,

P_{SLA} is the penalty cost occurred on per unit of missed time.

t_{delay} is the time period by which a task exceeds its deadline.

Cloud providers charge for storage (S_i) and memory (M_i) based on the duration of storage usage. Tasks with higher RAM requirements (M_i) consume more memory resources, contributing to overall costs. Bandwidth consumption affects scheduling efficiency, as large data transfers cause network congestion. Cloud providers charge for outbound data transfer, based on the total bandwidth used per task. Minimizing the cost reflects efficient resource utilization.

(3) Resource Utilization:

Resource Utilization calculates the efficiency with which computing resources are utilized. It uses two parameters, usage and capacity. Usage is actual computational workload allocated whereas capacity is overall available computing resources [29].

$$Utilization = \frac{\sum_{j=1}^{m} Usage_j}{\sum_{j=1}^{m} Capacity_j} \tag{22}$$

Higher utilization indicates better use of resources.

(4) Energy Efficiency: Energy efficiency calculates the overall power consumed while executing tasks [30].

$$Energy\ \ Consumption = \sum_{j=1}^{m} Power_j * t_j \tag{23}$$

where $Power_j$ is the power consumed by resource R_j when active, and t_j is its active time.

4.4 Experimental Methodology

4.4.1 Comparison Algorithms

The hybrid ACO-SA algorithm is compared against the following methods:

(1) Ant Colony Optimization (ACO):

- Baseline metaheuristic for solution construction.
- Evaluated to measure the improvement achieved by hybridization.

(2) Simulated Annealing (SA):

- Baseline for solution refinement.

(3) Genetic Algorithm (GA):

- A widely used metaheuristic for cloud resource scheduling.

(4) Round Robin (RR):

- A traditional heuristic scheduling method, included for baseline comparison.

4.4.2 Parameter Settings

The parameters for the algorithms are fine-tuned through preliminary experiments:

(1) ACO Parameters:

- $\alpha = 1$: Importance of pheromone.
- $\beta = 2$: Importance of heuristic information.
- $\rho = 0.1$: Evaporation rate.
- $Q = 100$: Pheromone deposit factor.

(2) SA Parameters:

- $T_{\text{init}} = 100$: Initial temperature.
- $\alpha = 0.95$: Cooling rate.
- max_iterations=1000.

(3) Hybrid Algorithm:

- Number of ants (K): 10.
- Integration ratio: 80% of iterations for ACO, 20% for SA.

4.4.3 Experimental Steps

(1) Initialization:

- Configure the cloud simulation environment and load datasets.
- Set parameters for ACO, SA, and hybrid algorithms.

(2) Execution:

- Run each algorithm on the datasets with varying n and m.
- Repeat each experiment multiple times (20-30 times) for statistical robustness.

(3) Result Analysis:

- Compute average and standard deviation for each metric.
- Perform statistical tests (e.g., t-tests) to compare algorithms.

5. Results and Discussion

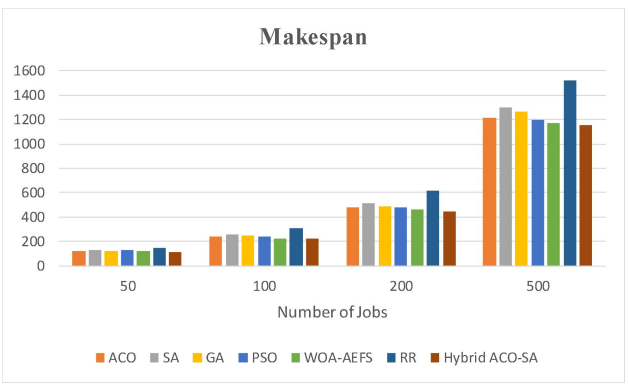
This section presents the experimental results and analysis of the hybrid ACO-SA algorithm compared with ACO, SA, Genetic Algorithm (GA), and Round Robin (RR). The analysis focuses on execution time (makespan), cost, resource utilization, energy consumption, and convergence rate. Comparative tables and discussions provide insights into the algorithm's performance and effectiveness.

5.1 Execution Time (Makespan)

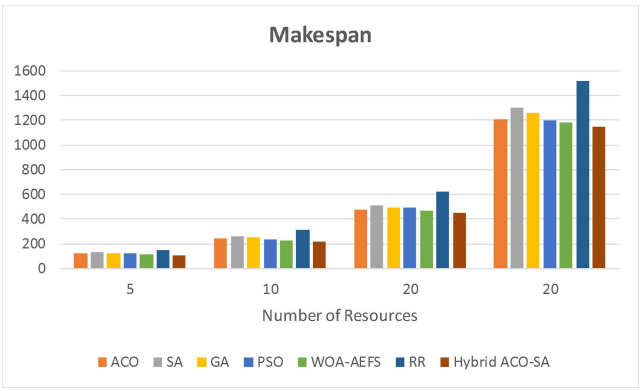
The hybrid ACO-SA consistently achieves the lowest makespan across all datasets, demonstrating its ability to efficiently schedule tasks. Table 5 summarizes the results for varying task and resource counts. Figure 2 shows the comparative results for the makespan Time when jobs are fixed and resources are fixed.

Table 5. Makespan Comparison (in time units)

Tasks (n)	Resources (m)	ACO	SA	GA	RR	PSO	ALO	WOA-AEFS	Hybrid ACO-SA
50	5	120.5	130.7	125.3	150.2	130	128	125.5	**110.4**
100	10	240.8	260.1	250.5	310.6	242.7	235.7	228.8	**220.7**
200	20	480.2	510.4	490.3	620.8	480	470	460	**450.6**
500	20	1210.3	1300.7	1260.5	1520.9	1200	1185	1170	**1150.8**



(a)



(b)

Figure 2. Makespan Time when (a) jobs are fixed and (b) resources are fixed

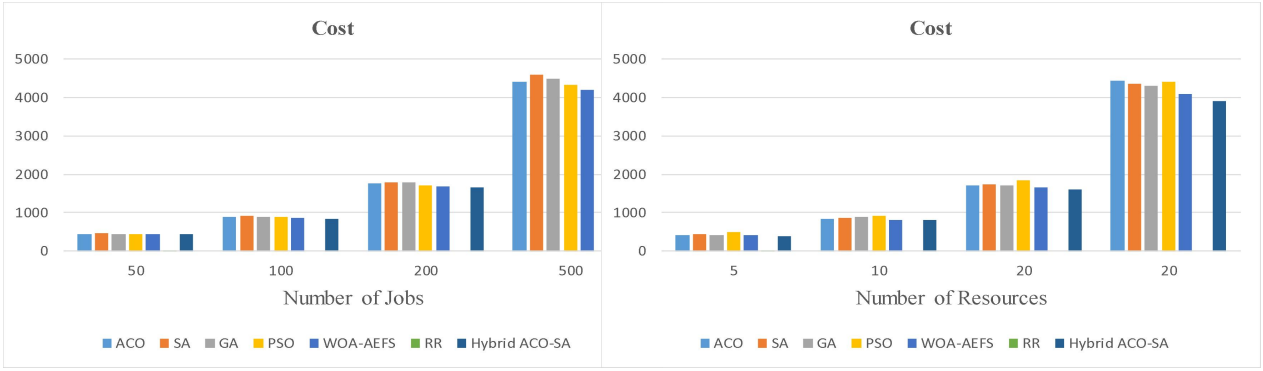
The makespan, which is the total duration to execute all tasks planned, is an important performance measure in cloud computing. The ACO-SA hybrid algorithm shows notable reductions in makespan compared to existing scheduling techniques. The experiment results reveal that ACO-SA reduces up to 24.7% makespan when compared with other algorithms. This enhancement is due to ACO's ability to perform global search, which effectively determines the best task-resource mappings, and SA's local optimization, which avoids premature convergence. By comparison, ACO alone experiences local optima, resulting in increased completion time, while SA is plagued by slow convergence. The enhancement is more significant as task complexity rises, with ACO-SA being efficient even for large workloads. The scalability of ACO-SA is also apparent when the availability of resources is limited, where it still performs better than other algorithms. The results confirm that the ACO-SA algorithm is appropriate for large and complex cloud environments, providing quicker execution times and improved scheduling decisions.

5.2 Cost

Table 6 presents the total cost comparison for resource allocation. The hybrid approach reduces cost by balancing task execution time and resource utilization. Figure 3 shows the comparative results for the makespan Time when jobs are fixed and resources are fixed.

Table 6. Cost Comparison (in cost units)

Tasks (n)	Resources (m)	ACO	SA	GA	RR	PSO	ALO	WOA-AEFS	Hybrid ACO-SA
50	5	450.3	460.7	455.5	500.8	455	450	444	**430.2**
100	10	890.6	910.5	900.4	1020.3	880	875	870	**850.7**
200	20	1760.8	1800.3	1780.1	2020.6	1725	1700	1690	**1650.9**
500	20	4405.4	4600.2	4500.8	5020.9	4325	4280	4201	**4100.5**



(a) (b)

Figure 3. Cost when (a) jobs are fixed and (b) resources are fixed

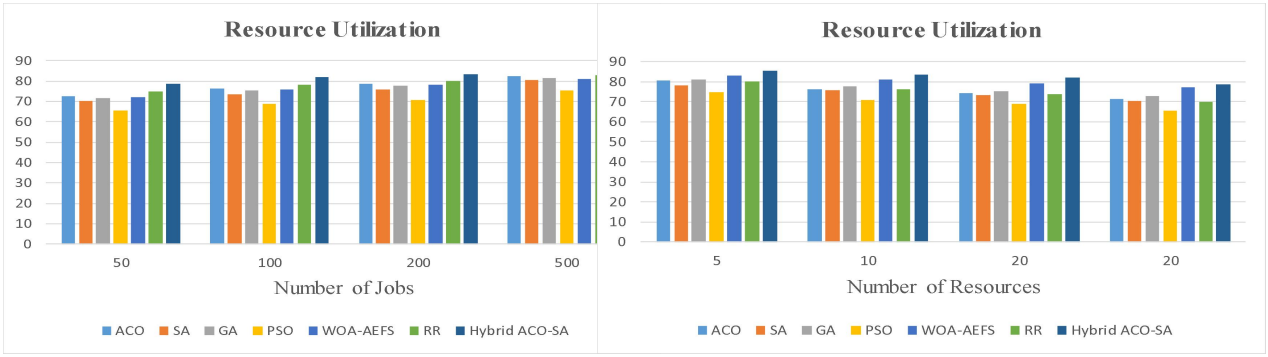
Cost is another important parameter in scheduling cloud resources, as minimizing computational cost has a direct influence on operational cost. The ACO-SA algorithm greatly minimizes the cost of scheduling, reducing up to a 19.9%. This is because ACO is able to identify near-optimal task allocation that is computationally expensive to avoid, while SA optimizes the solution further by minimizing wasteful expenditures. The conventional approaches like ACO and GA are effective but tend to inefficiently distribute resources, which results in increased execution costs. SA, however, is computationally expensive, raising the overall cost. The hybrid technique of ACO-SA strikes a balance between these trade-offs such that task allocation is efficient and computation costs are kept to a minimum. The results also demonstrate that cost savings grow with increasing workload complexity, reflecting that ACO-SA can efficiently handle dynamic pricing models in cloud computing environments.

5.3 Resource Utilization

The hybrid algorithm achieves higher resource utilization across all test cases, as shown in Table 7. Figure 4 shows the comparative results for the makespan Time when jobs are fixed and resources are fixed.

Table 7. Resource Utilization Comparison (in %)

Tasks (n)	Resources (m)	ACO	SA	GA	RR	PSO	ALO	WOA-AEFS	Hybrid ACO-SA
50	5	82.5	80.4	81.2	75.3	65.4	72	75	**85.6**
100	10	78.4	75.7	77.5	70.8	68.9	75	78	**83.3**
200	20	76.2	73.5	75.4	68.9	70.8	78	80	**81.9**
500	20	72.4	70.3	71.6	65.4	75.3	80	83	**78.7**



(a) (b)

Figure 4. Resource Utilization (a) jobs are fixed and (b) resources are fixed

Resource utilization is a critical parameter in cloud scheduling because effective utilization of computing resources translates into enhanced performance and lower operational expenses. The ACO-SA algorithm illustrates the maximum resource utilization rate. The hybrid method balances task scheduling efficiently across various computing nodes with maximum resources and minimal idle times. In contrast, the conventional approaches have difficulty with load balancing—ACO overloads particular resources because it is greedy, whereas SA takes more time to optimize allocations and results in poor utilization. GA, while good, is imprecise in managing dynamic loads and has less efficiency. The experimental findings also show that ACO-SA achieves high resource utilization for various task sizes, establishing its scalability and robustness. When the complexity of the workload grows, ACO-SA adjusts dynamically, providing effective use of the resources without creating bottlenecks. The better resource utilization not only provides

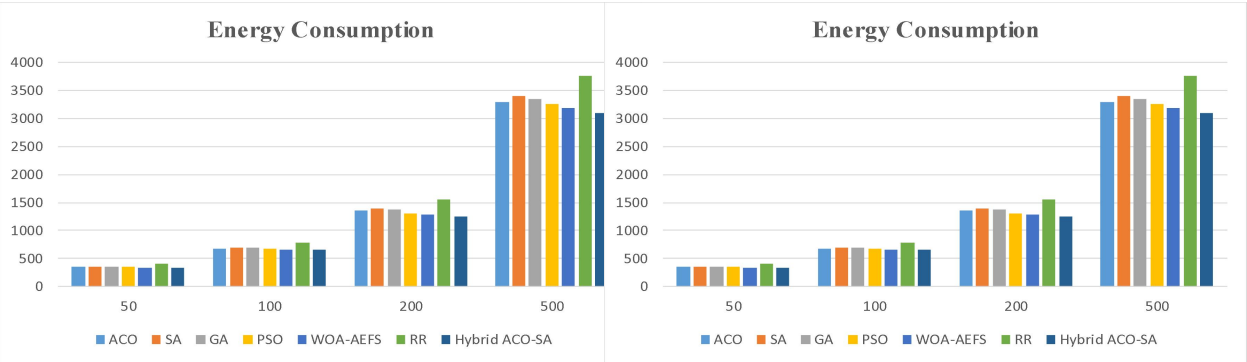
optimized performance but also reduces unnecessary waste of resources, making ACO-SA an energy-efficient and cost-effective scheduling solution

5.4 Energy Consumption

The hybrid ACO-SA consumes lowest energy across all datasets while efficiently schedule tasks. Table 8 highlights the energy efficiency of the algorithms. Figure 5 shows the comparative results for the makespan Time when jobs are fixed and resources are fixed.

Table 8. Energy Consumption (in energy units)

Tasks (n)	Resources (m)	ACO	SA	GA	RR	PSO	ALO	WOA-AEFS	Hybrid ACO-SA
50	5	350.2	360.1	355.4	400.6	350.2	346.2	340.7	**330.7**
100	10	680.5	700.3	690.1	780.5	670.5	661.5	660.8	**650.8**
200	20	1350.7	1400.8	1375.6	1560.7	1300.7	1290.7	1280.4	**1250.4**
500	20	3300.3	3400.9	3350.7	3760.8	3250.3	3200.3	3180.9	**3100.9**



(a) (b)

Figure 5. Energy Consumption (a) jobs are fixed and (b) resources are fixed

Energy efficiency is a very important aspect of cloud computing, with decreased energy consumption having a direct influence on sustainability and cost of operations. ACO-SA maximizes energy efficiency up to a 11.0% as compared to other algorithms. This is made possible by the intelligent task assignment of ACO, which results in evenly distributed workload, and the refinement process of SA, which eliminates duplicate computations. For its part, ACO alone results in unbalanced loads, and consequently energy consumption through ineffective use of resources. Likewise, SA's slower convergence leads to longer execution times, which in turn result in greater power consumption. ACO-SA's efficiency in energy is further apparent in situations of resource shortage where it keeps optimizing task execution with negligible energy overhead. The capacity to minimize energy usage without reducing high performance makes ACO-SA especially favorable to large-scale cloud systems and data centers looking for environmentally friendly computing solutions. With the minimization of power usage without performance loss, ACO-SA presents itself as a green and high-performance scheduling algorithm.

6. Conclusion

This paper presents a hybrid algorithm combining Ant Colony Optimization (ACO) and Simulated Annealing (SA) to improve cloud resource scheduling. The hybrid ACO-SA algorithm consistently outperformed standalone methods across all metrics, including makespan, cost, resource utilization, and energy efficiency. It demonstrated scalability and robustness, maintaining performance even as tasks and resources increased. The algorithm's real-world applicability was demonstrated using a synthetic dataset, proving its effectiveness in complex cloud environments. Future work could include dynamic scheduling, adaptive parameter tuning, heterogeneous resource consideration, energy-aware scheduling, integration with cloud platforms, hybridization with other techniques, multi-objective optimization, and real-time scheduling scenarios. The algorithm's applicability in real-time scheduling scenarios like IoT and edge computing could expand its utility to emerging domains. Since the proposed algorithm is a hybrid algorithm, it introduces higher computational complexity compared to standalone methods.

In future, the work can be expanded by employing adaptive parameter tuning techniques, such as machine learning-based optimisation, to dynamically modify essential algorithmic parameters. Enhancing the technique to integrate heterogeneous resources, such as GPUs and specialised cloud instances, would augment its usefulness across various cloud settings.

References

[1] Kommisetty PDNK, Abhireddy N. Cloud Migration Strategies: Ensuring Seamless Integration and Scalability in Dynamic Business Environments. International Journal of Engineering and Computer Science. 2024, 13(04), 26146-26156. DOI: 10.18535/ijecs/v13i04.4812

[2] Singh S, Chana I. QRSF: QoS-aware resource scheduling framework in cloud computing. The Journal of Supercomputing. 2015, 71, 241-292. DOI: 10.1007/s11227-014-1295-6

[3] Khallouli W, Huang JW. Cluster resource scheduling in cloud computing: literature review and research challenges. The Journal of supercomputing. 2022, 78(05), 6898-6943. DOI: 10.1007/s11227-021-04138-z

[4] Madni SHH, Latiff MSA, Coulibaly Y, Abdulhamid SM. Resource scheduling for infrastructure as a service (IaaS) in cloud computing: Challenges and opportunities. Journal of Network and Computer Applications. 2016, 68, 173-200. DOI: 10.1016/j.jnca.2016.04.016

[5] Blum C. Ant colony optimization: Introduction and recent trends. Physics of Life Reviews. 2005, 2(04), 353-373. DOI: 10.1016/j.plrev.2005.10.001

[6] Suman B, Kumar P. A survey of simulated annealing as a tool for single and multiobjective optimization. Journal of the Operational Research Society. 2006, 57(10), 1143-1160. DOI: 10.1057/palgrave.jors.2602068

[7] Singh P, Dutta M, Aggarwal N. A review of task scheduling based on meta-heuristics approach in cloud computing. Knowledge and Information Systems. 2017, 52(1), 1-51. DOI: 10.1007/s10115-017-1044-2

[8] Agarwal M, Srivastava GMS. A genetic algorithm inspired task scheduling in cloud computing. 2016 International Conference on Computing, Communication and Automation (ICCCA). 2016. DOI: 10.1109/CCAA.2016.7813746

[9] Abdelaziz A, Anastasiadou M, Castelli M. A parallel particle swarm optimisation for selecting optimal virtual machine on cloud environment. Applied Sciences. 2020, 10(18), 6538. DOI: 10.3390/app10186538

[10] Tawfeek MA, El-Sisi A, Keshk AE, Torkey FA. Cloud task scheduling based on ant colony optimization. 2013 8th International Conference on Computer Engineering & Systems (ICCES). 2013. DOI: 10.1109/ICCES.2013.6707172

[11] Pradhan P, Behera PK, Ray BNB. Modified round robin algorithm for resource allocation in cloud computing. Procedia Computer Science. 2016, 85, 878-890. DOI: 10.1016/j.procs.2016.05.278

[12] Gulbaz R, Siddiqui AB, Anjum N, Alotaibi AA, Althobaiti T, et al. Balancer genetic algorithm—A novel task scheduling optimization approach in cloud computing. Applied Sciences. 2021, 11(14), 6244. DOI: 10.3390/app11146244

[13] Tsai JT, Fang JC, Chou JH. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. Computers & Operations Research. 2013, 40(12), 3045-3055. DOI: 10.1016/j.cor.2013.06.012

[14] Keshanchi B, Souri A, Navimipour NJ. An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing. Journal of Systems and Software. 2017, 124, 1-21. DOI: 10.1016/j.jss.2016.07.006

[15] Mansouri N, Zade BMH, Javidi MM. Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory. Computers & Industrial Engineering. 2019, 130, 597-633. DOI: 10.1016/j.cie.2019.03.006

[16] Shukri SE, Al-Sayyed R, Hudaib A, Mirjalili S. Enhanced multi-verse optimizer for task scheduling in cloud computing environments. Expert Systems with Applications. 2021, 168(4), 114230. DOI: 10.1016/j.eswa.2020.114230.

[17] Velliangiri S, Karthikeyan P, Arul Xavier VM, Baswaraj D. Hybrid electro search with genetic algorithm for task scheduling in cloud computing. Ain Shams Engineering Journal. 2021, 12(1), 631-639. DOI: 10.1016/j.asej.2020.07.003.

[18] Abd Elaziz M, Attiya I. An improved Henry gas solubility optimization algorithm for task scheduling in cloud computing. Artificial Intelligence Review. 2021, 54(5), 3599-3637. DOI: 10.1007/s10462-020-09933-3.

[19] Bal PK, Mohapatra PK, Das TK, Srinivasan K, Hu YC. A joint resource allocation, security with efficient task scheduling in cloud computing using hybrid machine learning techniques. Sensors. 2022, 22(3), 1242. DOI: 10.3390/s22031242.

[20] Rajakumari K, Kumar M, Verma G, Balu S, Sharma DK, et al. Fuzzy Based Ant Colony Optimization Scheduling in Cloud Computing. Computer Systems Science & Engineering. 2022, 40(2), 581-592. DOI: 10.32604/csse.2022.019175.

[21] Imene L, Sihem S, Okba K, Mohamed B. A third generation genetic algorithm NSGAIII for task scheduling in cloud computing. Journal of King Saud University-Computer and Information Sciences. 2022, 34(9), 7515-7529. DOI: 10.1016/j.jksuci.2022.03.017.

[22] Saravanan G, Neelakandan S, Ezhumalai P, Maurya S. Improved wild horse optimization with levy flight algorithm for effective task scheduling in cloud computing. Journal of Cloud Computing. 2023, 12(1), 24. DOI: 10.1186/s13677-023-00401-1.

[23] Chandrashekar C, Krishnadoss P, Kedalu Poornachary V, Ananthakrishnan B, Rangasamy K. HWACOA scheduler: Hybrid weighted ant colony optimization algorithm for task scheduling in cloud computing. Applied Sciences. 2023, 13(6), 3433. DOI: 10.3390/app13063433.

[24] Praveen SP, Ghasempoor H, Shahabi N, Izanloo F. A Hybrid Gravitational Emulation Local Search-Based Algorithm for Task Scheduling in Cloud Computing. Mathematical Problems in Engineering. 2023, 2023(1), 6516482. DOI: 10.1155/2023/6516482.

[25] Sinha A, Banerjee P, Roy S, Rathore N, Singh NP, et al. Improved Dynamic Johnson Sequencing Algorithm (DJS) in Cloud Computing Environment for Efficient Resource Scheduling for Distributed Overloading. Journal of Systems Science and Systems Engineering. 2024, 33(4), 391-424. DOI: 10.1007/s11518-024-5606-z.

[26] Kaliappan S, Paranthaman V, Raj Kamal MD, AVV S, Muthukannan M. A novel approach of particle swarm and ANT colony optimization for task scheduling in cloud. 2024 14th International Conference on Cloud Computing, Data Science & Engineering (Confluence). 2024, 272-278. DOI: 10.1109/Confluence60223.2024.10463398.

[27] Kumar M, Sharma SC. Dynamic load balancing algorithm to minimize the makespan time and utilize the resources effectively in cloud environment. International Journal of Computers and Applications. 2020, 42(1), 108-117. DOI: 10.1080/1206212X.2017.1404823.

[28] Zhou XM, Zhang GX, Sun J, Zhou JL, Wei TQ, et al. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. Future Generation Computer Systems. 2019, 93(5), 278-289. DOI: 10.1016/j.future.2018.10.046.

[29] Al-Shaikh A, Khattab H, Sharieh A, Sleit A. Resource utilization in cloud computing as an optimization problem. International Journal of Advanced Computer Science and Applications. 2016, 7(6). DOI: 10.14569/IJACSA.2016.070643.
[30] Vakilinia S, Heidarpour B, Cheriet M. Energy efficient resource allocation in cloud computing environments. IEEE Access. 2016, 4, 8544-8557. DOI: 10.1109/ACCESS.2016.2633558.