**JISKA (Jurnal Informatika Sunan Kalijaga)**
**Vol. 11, No. 3, SEPTEMBER, 2026, Pp. 366 – 379**
**ISSN: 2527 – 5836 (print) | 2528 – 0074 (online)**

# Enhanced Ant Colony Optimization for Cloud Scheduling with Local Search and Elitist

**I Made Alit Darma Putra**
Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
e-mail : alitdarmaputra@gmail.com.
This article was submitted on 5 February 2026, revised on 27 August 2026, accepted on 28 August 2026, and published on 25 September 2026.

**Abstract**

Cloud computing allows the execution of various types of jobs with diverse resource requirements, so an efficient scheduling mechanism is needed to minimize processing time and improve resource utilization. The Ant Colony Optimization (ACO) algorithm is one of the metaheuristic methods widely used for job scheduling optimization. However, the ACO algorithm still has limitations in slow convergence and a tendency to get stuck in local solutions. This study proposes Enhanced Ant Colony Optimization with the Swap-Based Local Search and Elitist Strategy approaches to improve the quality of scheduling decisions. Evaluations were conducted in a heterogeneous emulated cloud environment with a focus on scheduling performance, convergence behaviour, and resource utilization metrics. The experimental results show that the proposed method outperforms FCFS, ACO, and ACO with Local Search for both normal and heavy scenarios. The study also demonstrates that the integration of local search and elitist strategy effectively improves the convergence process toward the global optimal solution while promoting a fairer utilization of resources. This paper may serve as a basis for further development on larger job scales and the addition of other optimization techniques.

**Keywords: Ant Colony Optimization, Cloud Computing, Elitist Strategy, Job Scheduling, Local Search**

**Abstrak**

*Komputasi awan memungkinkan eksekusi berbagai jenis tugas dengan beragam kebutuhan sumber daya, sehingga diperlukan mekanisme penjadwalan yang efisien untuk meminimalkan waktu pemrosesan dan meningkatkan pemanfaatan sumber daya. Algoritma Ant Colony Optimization (ACO) adalah salah satu metode metaheuristik yang banyak digunakan untuk optimasi penjadwalan tugas. Namun, algoritma ACO masih memiliki keterbatasan dalam konvergensi yang lambat dan kecenderungan untuk terjebak dalam solusi lokal. Studi ini mengusulkan Enhanced Ant Colony Optimization yang dioptimasi dengan Swap-Based Local Search dan Strategi Elitis untuk meningkatkan kualitas penjadwalan. Evaluasi dilakukan dalam lingkungan cloud yang disimulasikan dan bersifat heterogen dengan fokus pada metrik performa penjadwalan, perilaku konvergensi, dan pemanfaatan sumber daya. Hasil eksperimen menunjukkan bahwa performa dari metode yang diusulkan mengungguli FCFS, ACO, serta ACO dengan Local Search untuk skenario normal dan berat. Studi ini juga menunjukkan bahwa integrasi local search dan strategi elitis secara efektif meningkatkan proses konvergensi menuju solusi optimal global sekaligus mendorong pemanfaatan sumber daya yang lebih adil. Makalah ini dapat menjadi dasar pengembangan lebih lanjut pada skala tugas yang lebih besar dan penambahan teknik optimasi lainnya.*

***Kata Kunci: Ant Colony Optimization, Komputasi Awan, Local Search, Penjadwalan Tugas, Strategi Elitis***

## 1. INTRODUCTION

In recent years, the development of cloud computing has rapidly increased. This development is in line with the need for cloud services to be able to serve a variety of jobs that require resources in the cloud. The variety of jobs arises because nowadays different types of applications can run on the cloud. This cloud capability makes jobs dynamic and diverse in terms of size, execution time, and resource usage (Bashaireh, 2023).

Job scheduling is a mechanism for allocating certain jobs to specific resources at specific times (Murad et al., 2022). When the resources in the cloud are heterogeneous, job scheduling becomes a challenge to ensure optimal job completion. Mistakes in adopting the right scheduling algorithm can result in slow execution, wasted resources, and increased SLA violations (Samriya et al., 2021).

One of the classic scheduling algorithms is First Come, First Serve (FCFS). FCFS is an algorithm that schedules jobs that arrive first (Tarigan et al., 2021). This algorithm schedules jobs based solely on their order of arrival. Due to its simplicity, low computational overhead, and ease of implementation, FCFS is commonly used as a baseline for evaluating cloud job scheduling algorithms (Kumar & Shankar, 2026; Talaat & Hamza, 2025). However, this algorithm does not consider the resources required or the workload characteristics of each job. Furthermore, it can also lead to a problem of task queue congestion because different resources are assumed to have equal capabilities (Pemasinghe & Rajapaksha, 2022).

More adaptive scheduling algorithms continue to be developed. The Ant Colony Optimization (ACO) algorithm is a metaheuristic inspired by the foraging behaviour of ants which leave pheromone trails. ACO is capable of incorporating heterogeneous resource capabilities in a cloud environment, resulting in better scheduling (Jeyaraj & Paul, 2022). However, ACO typically requires a large number of iterations and may fail to obtain the global optimum solution (Shi & Jiang, 2025).

Prior studies have implemented ACO in job scheduling. For instance, Vidhya & Devi (2023) implemented a cloud job scheduling algorithm based on ACO and compared it with the Particle Swarm Optimization (PSO) algorithm using the CloudSim toolkit. The result showed that the ACO approach outperformed PSO, achieving a lower makespan of 254.52 ms compared to 317.76 ms for PSO. Additionally, Buulolo & Wau (2024) compared ACO with Genetic Algorithm (GA) and PSO for the production scheduling problem, showing that ACO resulted in a lower makespan and more efficient resource utilization. Furthermore, Kushwah et al. (2020) used ACO to balance the resource utilization in a cloud environment, resulting in the method outperforming other existing algorithms, such as FCFS, Round Robin, Min-min, and Max-min.

Despite its effectiveness, classical ACO still suffers from slow convergence and the risk of failing to obtain globally optimal solutions (Bei et al., 2024). To achieve better results, the ACO algorithm was optimized with a local search algorithm. This optimization aims to enable the algorithm to better explore local neighborhoods within the search space by refining the solution at each iteration, thus enabling the algorithm to achieve a more optimal solution (Guo et al., 2022). Bestari et al. (2024) adopted local search optimization using the swap operation approach for job scheduling, which led to a reduction in total makespan by 5.08% compared to the FCFS method. Further, Gabhane et al. (2023) performed local search optimization for ACO in cloud job scheduling. However, the evaluation was limited to a simulated environment.

In addition to optimizing the search process, optimization is also developed regarding how pheromones are updated. One strategy that can be implemented is the elitist strategy (Abd Alhussain & Hassan, 2024). This elitist strategy helps find optimal solutions quickly by strengthening the pheromone trails of the best solutions and reducing unnecessary exploration of low-quality solutions (Abd Alhussain & Hassan, 2024). Djebbar & Belalem (2025) conducted research to compare the elitist strategy with the classical ACO algorithm in the CloudSim environment. The result showed that the optimization reduces the total response time, the average response time, and the total execution cost. Similarly, Kansara et al. (2023) applied the elitist strategy to improve performance in cloud job scheduling with ACO. Still, the implementation did not incorporate any local search mechanism and was also limited to CloudSim-based simulations.

As presented in Table 1, prior studies have employed various ACO-based approaches for cloud task scheduling, incorporating different enhancement strategies such as heuristic mechanisms,

hybrid optimization, local search, and elitist pheromone updating. However, few have discussed optimizing ACO that combines local search and the elitist strategy specifically in cloud job scheduling. By integrating these two mechanisms, the quality of solutions constructed by ants at each iteration is expected to be progressively refined, potentially leading to more effective pheromone update and discovery of the global best solution.

**Table 1 State-of-the-Art ACO-Based Cloud Scheduling Approaches**

| Ref | Method | Local Search | Elitist | Metrics | Environment |
|---|---|---|---|---|---|
| Pirozmand et al. (2023) | Hybrid (PSO + ACO) | - | - | Execution Time | Simulation |
| Gabhane et al. (2023) | ACO | Yes | No | Makespan, throughput, and total cost | Simulation |
| Abdulghani (2024) | Hybrid (ACO + GA) | No | Yes | Makespan | Simulation |
| **Proposed** | **ACO** | **Yes** | **Yes** | **Makespan, response time, throughput, SLA violation, convergence behaviour, utilization** | **Emulation** |

To address the gap in prior studies, this study aims to (1) develop an enhanced ACO algorithm with swap-based local search and elitist strategy optimization for cloud job scheduling and (2) compare the developed algorithm with other algorithms such as FCFS, ACO, and ACO with Swap-Based Local Search in an emulation environment. The structure of this paper is as follows: The Methods section describes the proposed enhanced Ant Colony Optimization algorithm; the Results and Discussion section presents the experimental results and their analysis; finally, the Conclusion section summarizes the overall study.

## 2. METHODS

This section describes the methods used in developing the proposed scheduling approach. This study focuses on improving the Ant Colony Optimization algorithm through the application of Swap-Based Local Search and Elitist Strategy to overcome the problems of slow convergence and the tendency to get stuck in local solutions. This section explains the experiment flow, algorithm improvement mechanisms, and experimental configurations used as the basis for performance testing. Flowcharts, system models, and algorithmic stages are also presented to clarify the implementation process and the overall algorithm execution flow.

### 2.1 Experimental Flow

Figure 1 briefly illustrates the experimental flow, which consists of three stages: System Preparation, Job Execution, and Evaluation. These stages represent the sequence of processes performed from preparing the emulation environment to evaluating the experimental results. A comparative evaluation is conducted among FCFS, ACO, ACO with Local Search (ACO-LS), and ACO with Local Search and Elitist (ACO-LSE).

In the System Preparation stage, the emulation environment is prepared using technologies such as VirtualBox with Ubuntu Server and Docker based on predetermined specifications. This stage also includes the job initialization process to determine the characteristics and distribution of the jobs to be executed. The prepared environment and initialized jobs are then used as the basis for the subsequent execution stage.

In the Job Execution stage, the prepared jobs are executed using several different scheduling algorithms in the emulation environment. During the execution process, each job parameter, such

as the selected VM and makespan, is recorded for documentation and evaluation purposes. The recorded results provide the data required to compare the performance of the scheduling algorithms.

The Evaluation stage is the final stage of the experimental flow and includes an analysis and discussion of the results. Each observed job execution metric is compared across the scheduling algorithms to identify differences in their performance. The comparison is then used to draw conclusions and determine the algorithm with the best performance.

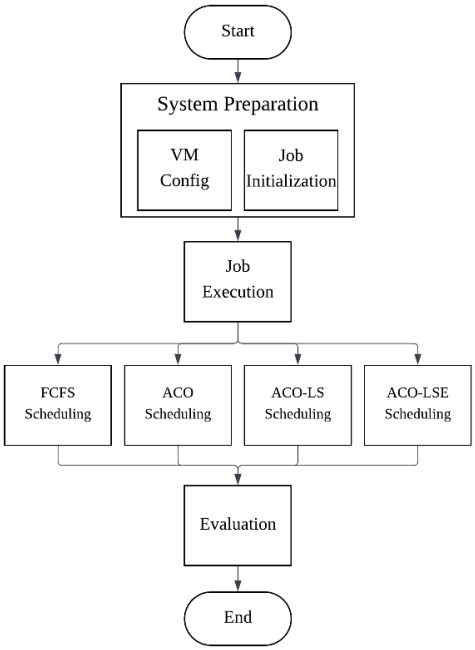


**Figure 1 Experimental Flow**

### 2.2 Ant Colony Optimization Algorithm

Ant Colony Optimization is an optimization algorithm that solves problems using probabilistic techniques on a graph model (Hussain et al., 2022). This algorithm is inspired by how ants communicate in nature, finding the best route from the nest to a food source by leaving pheromone trails for other ants to follow. In the implementation of the ACO algorithm, there are two main mechanisms. The first mechanism is solution space exploration, and the second is pheromone update (Tian et al., 2024). Each ant constructs a solution by probabilistically selecting a path based on pheromone values and local heuristics as defined in Eq. (1).

$$P_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l \in N_i^k}(\tau_{il})^{\alpha}\,(\eta_{il})^{\beta}} \tag{1}$$

The probability of ant-k moving from node i to node j will increase if the pheromone value $\tau_{ij}$ is high or the heuristic value $\eta_{ij} = \frac{1}{d_{ij}}$ (e.g., distance, time, or cost) is small. The parameters $\alpha$ and $\beta$ control the influence of pheromone and heuristic. After all ants have completed solution construction, pheromone values are updated to strengthen the best path and evaporate suboptimal ones following Eq. (2). This process is carried out iteratively until convergence is reached.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{best\_iter}(t) \tag{2}$$

The $\rho$ variable determines the level of evaporation that occurs. The pheromone contribution from the k-th ant is calculated through Eq. (3). This contribution is used in the pheromone update process to influence path selection in the subsequent iteration.

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{1}{L_k}, if\ ant\ k\ passes\ path\ i \to j \\ 0, otherwise \end{cases} \quad (3)$$

### 2.3 Enhanced Ant Colony Optimization

To address the limitations of basic ACO, including premature convergence and stagnation in local optima, this study proposes an Enhanced Ant Colony Optimization (ACO) algorithm incorporating two key enhancements: Swap-Based Local Search and Elitist Strategy. Swap-Based Local Search improves the solutions generated by the ants through local modifications of job-to-VM assignments, while the Elitist Strategy reinforces the best solution through additional pheromone deposition. The complete pseudocode of the proposed Enhanced ACO scheduling algorithm is provided in Appendix A.

#### 2.3.1 Local Search

Local search is an optimization technique that uses neighborhood search to improve the solutions produced by ACO (Muklason & Agung Premananda, 2023). Swap-Based Local Search generates neighboring solutions by swapping two elements in the current solution, as defined in Eq. (4). In this study, the swap operation exchanges the VM allocations of two randomly selected jobs.

$$S' = \mathrm{swap}(S, j_1, j_2) \quad (4)$$

The variables $j_1$ and $j_2$ represent two randomly selected jobs whose VM allocations are exchanged. If the new solution produces a better objective value or lower execution time, the current solution is updated. Swap-Based Local Search is applied at each ACO iteration to refine the constructed solutions before the pheromone update is performed, as shown in lines 15–26 of Algorithm 1 in Appendix A.

#### 2.3.2 Elitist Strategy

The elitist strategy extends the ACO algorithm by assigning a greater pheromone weight to the best solution, thereby encouraging quicker convergence (Abd Alhussain & Hassan, 2024). Mathematically, the pheromone update follows Eq. (5). This mechanism strengthens the influence of the global best solution on subsequent iterations.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{global\_best}(t)\omega \quad (5)$$

The variable $\omega$ represents the elitist weight used to reinforce the best solution. A larger value of $\omega$ increases the influence of the global best path on the next iteration. The elitist strategy is applied during the pheromone update phase, as shown in lines 30–41 of Algorithm 1 in Appendix A, by assigning additional pheromone to the global best solution.

### 2.4 Evaluation Metrics

The performance of the proposed Enhanced Ant Colony Optimization algorithm is evaluated using six key metrics that assess different aspects of scheduling effectiveness. These metrics include makespan, average response time, throughput, SLA violation, convergence behaviour, and resource utilization. Together, these metrics provide an evaluation of scheduling performance in terms of efficiency, responsiveness, service quality, convergence, and resource usage.

**Makespan.** Makespan represents the total time required to complete all jobs, measured from the start of the first job to the completion of the last job. The makespan calculation is defined in Eq.

(6), where $C_i$ represents the completion time of job $i$ and $n$ is the total number of jobs. It is the primary metric used in this study to evaluate scheduling efficiency.

$$Makespan = \max\{C_i\} = max\{C_1, C_2, \ldots, C_n\} \quad (6)$$

**Average Response Time.** Average Response Time (RT) measures the mean time from job submission to job completion, reflecting system responsiveness from the user's perspective. It is calculated as shown in Eq. (7), where $RT_i$ denotes the response time for job *i* and *n* represents the total number of jobs.

$$Average\ RT\ = \frac{1}{n}\sum_{i}^{n} RT_i \quad (7)$$

**Throughput.** Throughput measures the number of jobs completed per unit time, indicating the productivity and processing capacity of the scheduling system. It is calculated using Eq. (8), where $n$ represents the number of completed jobs and $T_{total}$ denotes the total time span from start to finish.

$$Throughput = \frac{n}{T_{total}} \quad (8)$$

Average response time and throughput provide complementary perspectives on scheduling performance. Average response time evaluates how quickly individual jobs are completed, whereas throughput measures the number of jobs that can be completed within a given period. These metrics are therefore used alongside makespan to provide a broader assessment of scheduling efficiency.

**SLA Violation.** Service Level Agreement (SLA) Violation measures the percentage of jobs that fail to meet predefined performance constraints, particularly deadline requirements. The calculation is formulated in Eq. (9), where $N_{violated}$ represents the number of jobs that violate the SLA and $N_{total}$ represents the total number of submitted jobs. This metric is critical for maintaining Quality of Service (QoS). In this study, a job is considered to have violated the SLA if the response time is higher than 0.05 s.

$$SLA\ Violation\ Rate = \frac{N_{violated}}{N_{total}} \times 100\% \quad (9)$$

**Convergence Behaviour.** Convergence behaviour is used to analyze the progression of the ACO-based algorithm toward convergence. Convergence refers to the condition in which the global solution no longer shows significant improvement. In this study, convergence is evaluated by recording the global best makespan at each iteration, which represents the best scheduling solution obtained up to that iteration. By observing changes in the global best makespan across iterations, the convergence pattern of the algorithm can be identified.

**Resource Utilization.** Each algorithm can produce different scheduling patterns that affect VM resource usage. Therefore, resource utilization is evaluated to examine the impact of each scheduling algorithm on the available computing resources. In this study, the evaluation focuses on CPU utilization by monitoring the percentage of CPU usage for each VM during the experiment.

### 2.5 Experimental Setup

In this study, emulation was performed on a personal computer equipped with a 3.1 GHz Intel Core i5 processor, 16 GB DDR4 RAM, and a 500 GB SSD disk. Available resources were divided into three virtual machines via VirtualBox, as shown in Figure 2, each with different computing capabilities, reflecting the heterogeneous nature of cloud resources. The host acted as a

scheduler, sending jobs to each VM and recording evaluation metrics. Each VM instance ran a worker that received jobs from the scheduler. Here, the job in the VM was emulated as a Docker container with limited resources according to the job's characteristics, running a stressful application that would consume all allocated resources.

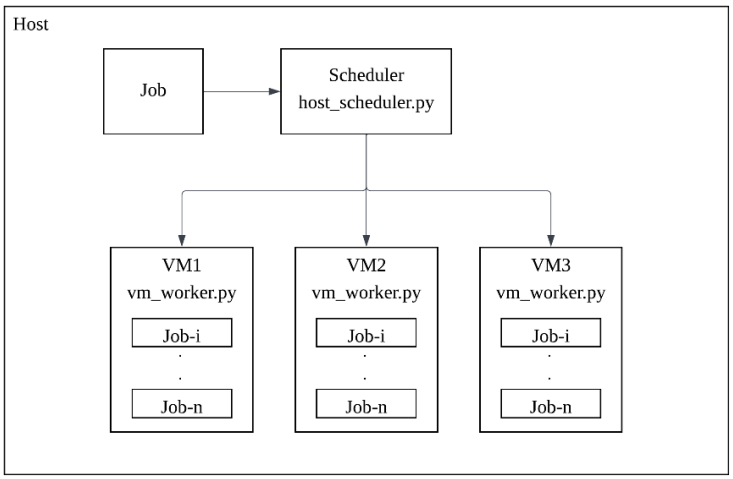


**Figure 2 Experimental Flow**

In this study, three virtual machines (VMs) were used as the execution environment for job scheduling. Each VM is configured with different computational resources to simulate a heterogeneous cloud environment. The specifications of each VM, including the number of virtual CPUs (vCPUs) and memory capacity, are summarized in Table 2. These variations allow the evaluation of the scheduling algorithms under diverse resource constraints, reflecting realistic cloud computing scenarios.

**Table 2 VM Configurations**

| VM ID | IP Address | Port | vCPU | Memory (MB) |
|---|---|---|---|---|
| 1 | 192.168.56.101 | 5001 | 1 | 1024 |
| 2 | 192.168.56.102 | 5002 | 2 | 1024 |
| 3 | 192.168.56.103 | 5003 | 4 | 1024 |

### 2.6 Job Scenarios

The specifications of jobs are made following a certain distribution pattern to simulate realistic cloud workload scenarios (Mangalampalli et al., 2023). In this study, two evaluation scenarios with different job specifications are considered: normal and heavy workload scenarios. These scenarios are designed to evaluate the scheduling algorithms under different workload conditions.

#### 2.6.1 Normal Workload Scenario

In this scenario, the specifications of the jobs were generated randomly following a normal distribution with a mean value of 1200 and a standard deviation of 600. The total number of jobs generated for this scenario is 20 jobs, as presented in Figure 3. The aim of this scenario is to see the performance of each algorithm in a normal workload scenario as the comparison baseline.

#### 2.6.2 Heavy Workload Scenario

Some algorithms may perform best in normal scenarios but suffer when the workload is extreme. Therefore, evaluations in heavy workload scenarios were conducted. In this scenario, the CPU requirement for jobs was determined randomly following a normal distribution with a mean value of 2000 and a standard deviation of 1000. The total number of jobs also increased to 100 jobs to

emulate a server that needs to serve many requests simultaneously. This scenario aims to see the behaviour of each algorithm when the workload increases. Figure 4 depicts the job distribution under a heavy workload scenario.

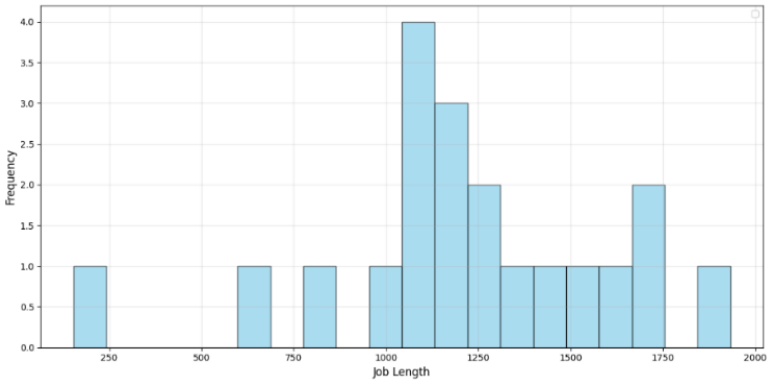


**Figure 3 Job Distribution Under Normal Workload**

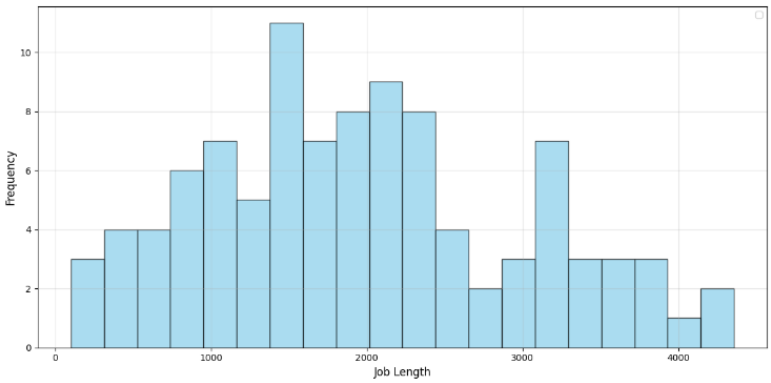


**Figure 4 Job Distribution Under Heavy Workload**

### 2.7 ACO Parameters

In ACO, several parameters must be set before the solution search process is carried out. The value of each parameter is described in Table 3. In this study, the heuristic parameter value is set higher than the pheromone parameter value, following a similar treatment adopted in other related studies (Sanaj et al., 2026; Singhal, 2025). This setting is further motivated by the dynamic nature of heuristic values, which reflect current resource and system utilization conditions (Lilhore et al., 2025).

**Table 3 ACO Parameters**

| Name | Description | Value |
|---|---|---|
| Number of Ants | Number of artificial ants constructing solutions in each iteration | 5 |
| Number of Iterations | Total number of optimization cycles performed | 20 |
| Pheromone Evaporation Rate | Rate at which pheromone trails evaporate | 0.1 |
| $\alpha$ | Relative importance of pheromone trails in the solution construction process | 1.0 |
| $\beta$ | Relative importance of heuristic information in guiding solution selection | 2.0 |
| $\omega$ | Weight factor used to reinforce pheromone levels of the global best solution | 5.0 |

## 3. RESULTS AND DISCUSSION

This section presents the results of tests conducted to evaluate the performance of the Enhanced Ant Colony Optimization (ACO) algorithm, namely ACO with Local Search and Elitist (ACO-LSE), compared to three benchmark methods: FCFS, standard ACO, and ACO with Local Search (ACO-LS). The experiment was done in normal and heavy workload scenarios with three heterogeneous VM configurations and different job characteristics.

Experiments using a normal workload scenario were conducted with 20 jobs. Jobs had varying CPU requirements, with an average of 1200 millicores. The results of the evaluation metrics for each algorithm in the normal workload scenario are summarized in Table 4. In a normal workload scenario, ACO-LSE achieves the lowest makespan (0.905s) and average response time (0.044s), representing a roughly 38% improvement in speed compared to the baseline FCFS algorithm. This computational efficiency directly correlates with the highest throughput of 22.09 jobs per second, significantly outperforming the standard ACO and ACO-LS variants. In this scenario, based on the SLA violation rate, the proposed optimization significantly reduces the number of SLA violations. The ACO algorithm appears unable to produce the best solution, resulting in more SLA violations than the FCFS baseline.

**Table 4 Experimental Results Under Normal Workload**

| Algorithm | Makespan (s) | Average Response Time (s) | Throughput (jobs/s) | SLA Violation (%) |
|---|---|---|---|---|
| FCFS | 1.476359 | 0.073230 | 13.546842 | 20.00 |
| ACO | 1.387929 | 0.067616 | 14.409957 | 25.00 |
| ACO-LS | 1.300568 | 0.063803 | 15.377892 | 25.00 |
| **ACO-LSE** | **0.905222** | **0.044480** | **22.094018** | **10.00** |

Based on the convergence comparison under normal workload depicted in Figure 5, the implementation of local search and elitist optimization accelerates convergence and guides the ACO algorithm toward global optimal solutions. In the normal scenario, the ACO-LSE is able to produce a better makespan value as the best solution compared to other ACO-based methods, which tend to stagnate. The results of resource usage on each virtual machine (VM) under normal workload are visualized graphically in Figure 6. The comparison shows that ACO-based scheduling appears to be fairer and better considers the capabilities of each VM compared to FCFS.

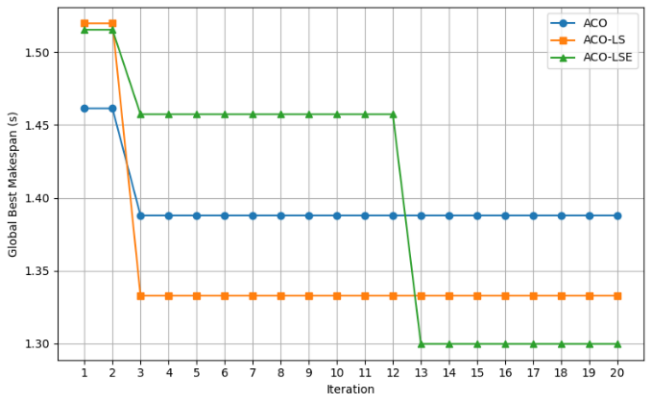


**Figure 5 Convergence Comparison Under Normal Workload**

In the experiment using a heavy workload scenario, the number of jobs executed was increased to 100. The CPU requirement for each job also increased to an average of 2000 millicores. Table 5 shows the evaluation results for each monitored metric. In this scenario, the FCFS algorithm struggles to compete with the ACO-based algorithm. The ACO-based algorithm outperforms the

FCFS baseline, with the best results achieved in ACO-LSE with makespan (9.410 s), average response time (0.093 s), throughput (10.627 job/s), and SLA violation (49%). As the workloads become heavier, the performance of all tested algorithms decreases in this scenario.

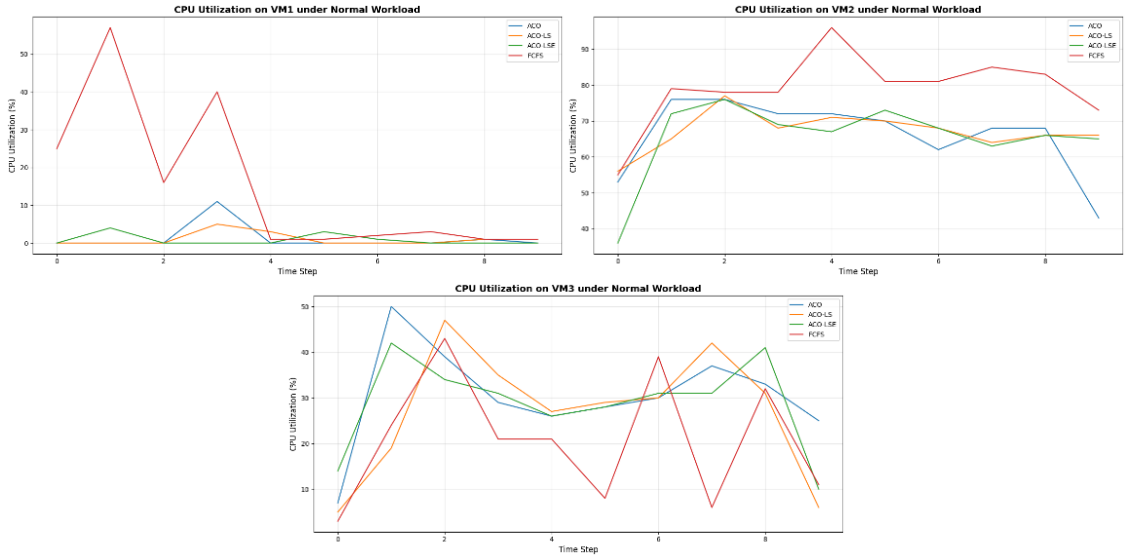


**Figure 6 CPU Utilization Comparison Under Normal Workload**

**Table 5 Experimental Results Under Heavy Workload**

| Algorithm | Makespan (s) | Average Response Time (s) | Throughput (jobs/s) | SLA Violation (%) |
|---|---|---|---|---|
| FCFS | 14.773962 | 0.146793 | 6.768665 | 66.00 |
| ACO | 10.946791 | 0.108620 | 9.135097 | 55.00 |
| ACO-LS | 10.410109 | 0.103168 | 9.606047 | 50.00 |
| **ACO-LSE** | **9.410198** | **0.093133** | **10.626769** | **49.00** |

The convergence comparison graph under heavy workload is shown in Figure 7. The comparison shows that the proposed method is able to reach convergence conditions faster and produce an optimal solution with only 3 iterations compared to ACO and ACO-LS. Figure 8 depicts the behaviour of the ACO-based algorithm under heavy workload. Similarly to the normal scenario, the ACO-based algorithm tends to select VMs with higher specifications when processing heavy workloads. Meanwhile, the FCFS algorithm still assigns lower-spec VMs to process heavy jobs, resulting in slower execution times for some workloads.

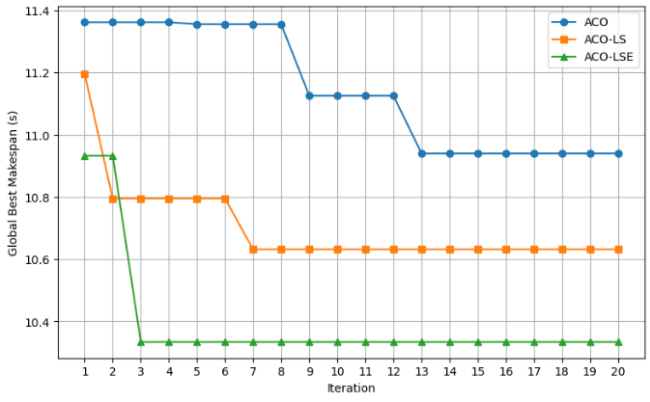


**Figure 7 Convergence Comparison Under Heavy Workload**

The comparative analysis of scheduling algorithms shows that the ACO-LSE method delivers superior performance across all evaluated metrics, establishing it as the most efficient approach among those tested. By taking resource capabilities more effectively than FCFS, it achieves better scheduling performance. Furthermore, the proposed method is also quicker in generating an optimal solution and reaching a convergence state compared to other ACO-based methods. This superiority is seen in both normal and heavy workload scenarios.

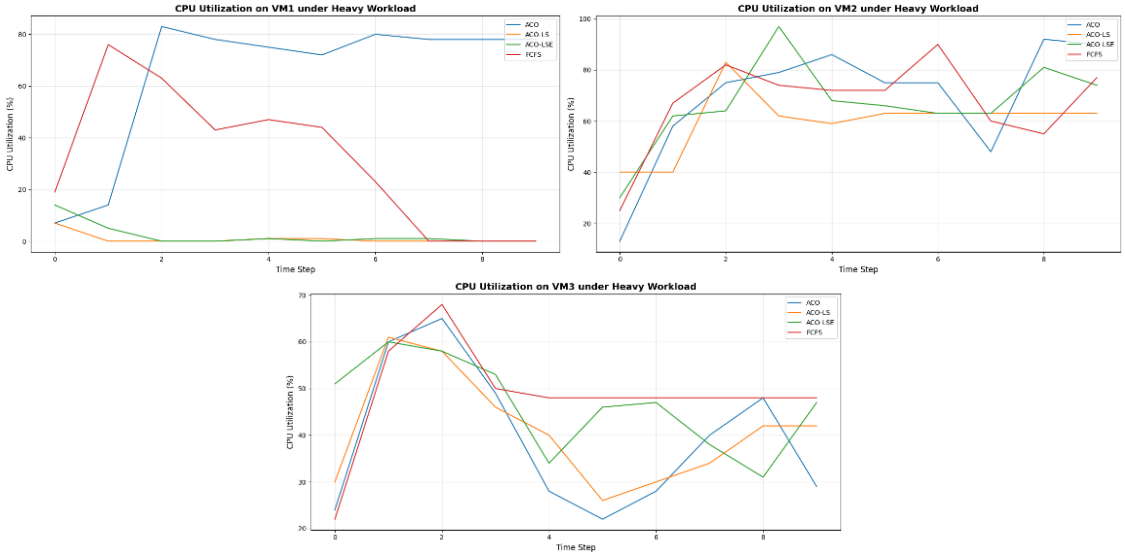


**Figure 8 CPU Utilization Comparison Under Heavy Workload**

## 4. CONCLUSIONS

This paper has implemented optimization on Ant Colony Optimization (ACO) with Swap-Based Local Search and an Elitist Strategy in the cloud job scheduling process. Implementation of the algorithm in a heterogeneous local VM environment demonstrated that the proposed method provides significant performance improvements compared to FCFS, ACO, and ACO-LS. ACO-LSE achieved the lowest makespan and response time, the highest throughput, and the lowest SLA violation rate for both normal and heavy workload scenarios. Thus, the addition of a local search mechanism and an elitist strategy proved effective in accelerating solution convergence, improving scheduling efficiency, and achieving better CPU utilization through more balanced workload distribution. This study is still limited to a relatively small number of jobs and resource configurations. Further study could focus on large-scale testing, the implementation of adaptive parameters, and integration with real-world cloud systems to more closely reflect production conditions. Further development could also include multi-objective optimization to improve other aspects such as energy consumption or fairness of load distribution.

## REFERENCES

Abd Alhussain, Z. A., & Hassan, L. S. (2024). A Review on Elitist Ant System Algorithm and Applications in Flexible Job Scheduling Problem. *Al-Bahir*, *4*(2), Article ID: 5. https://doi.org/10.55810/2313-0083.1059

Abdulghani, A. M. (2024). Hybrid Task Scheduling Algorithm for Makespan Optimisation in Cloud Computing: A Performance Evaluation. *Journal on Artificial Intelligence*, *6*(1), 241–259. https://doi.org/10.32604/jai.2024.056259

Bashaireh, R. A. L. (2023). Task Scheduling in Cloud Computing: Priority-Based Algorithms and Future Directions in Data Science. *Journal of Theoretical and Applied Information Technology*, *15*(15). https://www.jatit.org/volumes/Vol101No15/3Vol101No15.pdf

Bei, L., Wenlin, L., Xin, S., & Xibin, X. (2024). An Improved ACO Based Service Composition Algorithm in Multi-Cloud Networks. *Journal of Cloud Computing*, *13*(1), Article ID: 17. https://doi.org/10.1186/s13677-024-00588-x

Bestari, H. M. H., Suryadhini, P. P., & Nopendri, N. (2024). Flow Shop Scheduling Using a Combination of Ant Colony Optimization Algorithm and Tabu Search Algorithm to Minimize Total Tardiness. *Jurnal Teknik Industri: Jurnal Hasil Penelitian dan Karya Ilmiah dalam Bidang Teknik Industri*, *10*(2), 383–391. https://doi.org/10.24014/jti.v10i2.32320

Buulolo, K., & Wau, K. (2024). Performance Comparison of Metaheuristic Optimization Algorithms in Solving Production Scheduling Problems. *Jurnal ICT: Information and Communication Technologies*, *15*(2), 122–132. https://doi.org/10.35335/jict.v15i2.197

Djebbar, E. I., & Belalem, G. (2025). Improvement of the ACO Algorithm for Intelligent Task Scheduling in Cloud Systems. *Scalable Computing: Practice and Experience*, *26*(2), 531–539. https://doi.org/10.12694/scpe.v26i2.3946

Gabhane, J. P., Pathak, S., & Thakare, N. M. (2023). A Novel Hybrid Multi-Resource Load Balancing Approach Using Ant Colony Optimization with Tabu Search for Cloud Computing. *Innovations in Systems and Software Engineering*, *19*(1), 81–90. https://doi.org/10.1007/s11334-022-00508-9

Guo, N., Qian, B., Na, J., Hu, R., & Mao, J.-L. (2022). An Enhanced Ant Colony Algorithm with Variable Neighborhood Descent for Multi-compartment Vehicle Routing Problem with Time Limits. *2022 41st Chinese Control Conference (CCC)*, 1961–1966. https://doi.org/10.23919/CCC55666.2022.9901913

Hussain, S. F., Butt, I. A., Hanif, M., & Anwar, S. (2022). Clustering Uncertain Graphs Using Ant Colony Optimization (ACO). *Neural Computing and Applications*, *34*(14), 11721–11738. https://doi.org/10.1007/s00521-022-07063-1

Jeyaraj, R., & Paul, A. (2022). Optimizing MapReduce Task Scheduling on Virtualized Heterogeneous Environments Using Ant Colony Optimization. *IEEE Access*, *10*, 55842–55855. https://doi.org/10.1109/ACCESS.2022.3176729

Kansara, A. S., Gohil, B. N., & Verma, N. (2023). Task Scheduling in Cloud Computing Using Mean Ant Colony Optimization. *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*, 1–8. https://doi.org/10.1109/ISCON57294.2023.10112102

Kumar, B. S., & Shankar, R. (2026). A Constrained Multi-Objective Metaheuristic Reinforcement Learning Framework for Adaptive Load Balancing in Heterogeneous Edge Cloud Environments. *Sustainable Computing: Informatics and Systems*, *51*, Article ID: 101419. https://doi.org/10.1016/j.suscom.2026.101419

Kushwah, V. S., Goyal, S. K., & Sharma, A. (2020). Maximize Resource Utilization Using ACO in Cloud Computing Environment for Load Balancing. In S. Computing (Ed.), *T. K. and V. O. P. and S. R. and S. A. Pant Millie and Sharma* (pp. 583–590). Springer. https://doi.org/10.1007/978-981-15-0751-9_54

Lilhore, U. K., Simaiya, S., Rao, K. B. V. B., Rao, V. V. R. M., Sharma, Y. K., Alroobaea, R., Alsufyani, H., Alsafyani, M., & Khan, M. D. M. (2025). Hybrid DRL-Enhanced ACO-WWO for Efficient Resource Allocation and Load-Balancing in Cloud Computing. *International Journal of Computational Intelligence Systems*, *18*(1), Article ID: 148. https://doi.org/10.1007/s44196-025-00882-9

Mangalampalli, S., Karri, G. R., & Elngar, A. A. (2023). An Efficient Trust-Aware Task Scheduling Algorithm in Cloud Computing Using Firefly Optimization. *Sensors*, *23*(3), Article ID: 1384. https://doi.org/10.3390/s23031384

Muklason, A., & Agung Premananda, I. G. (2023). Hybrid Iterated Local Search Algorithm for Optimization Route of Airplane Travel Plans. *International Journal of Electrical and Computer Engineering (IJECE)*, *13*(4), 4700–4707. https://doi.org/10.11591/ijece.v13i4.pp4700-4707

Murad, S. A., Muzahid, A. J. M., Azmi, Z. R. M., Hoque, M. I., & Kowsher, M. (2022). A Review on Job Scheduling Technique in Cloud Computing and Priority Rule Based Intelligent Framework. *Journal of King Saud University - Computer and Information Sciences*, *34*(6), 2309–2331. https://doi.org/10.1016/j.jksuci.2022.03.027

Pemasinghe, S., & Rajapaksha, S. (2022). Comparison of CPU Scheduling Algorithms: FCFS, SJF, SRTF, Round Robin, Priority Based, and Multilevel Queuing. *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*, 318–323. https://doi.org/10.1109/R10-HTC54060.2022.9929533

Pirozmand, P., Jalalinejad, H., Hosseinabadi, A. A. R., Mirkamali, S., & Li, Y. (2023). An Improved Particle Swarm Optimization Algorithm for Task Scheduling in Cloud Computing. *Journal of Ambient Intelligence and Humanized Computing*, *14*(4), 4313–4327. https://doi.org/10.1007/s12652-023-04541-9

Samriya, J. K., Chandra Patel, S., Khurana, M., Tiwari, P. K., & Cheikhrouhou, O. (2021). Intelligent SLA-Aware VM Allocation and Energy Minimization Approach with EPO Algorithm for Cloud Computing Environment. *Mathematical Problems in Engineering*, *2021*(1), 1–13. https://doi.org/10.1155/2021/9949995

Sanaj, M. S., Horng, M.-F., Shankar, S. S., Lo, C.-C., Sunder, R., Lilhore, U. K., Simaiya, S., Tekeste, L. G., Mohamed, H. G., & Ghith, E. S. (2026). Energy-Efficient Task Scheduling in Cloud Computing with EcoTaskOpt: A Hybrid Ant Colony and Particle Swarm Optimization Approach. *International Journal of Computational Intelligence Systems*, *19*(1), Article ID: 64. https://doi.org/10.1007/s44196-025-01095-w

Shi, G., & Jiang, X. (2025). Improved Ant Colony Algorithm-Based Path Planning for Mobile Robots. In J. Ma, S. Kadry, & R. A. El-Nabulsi (Eds.), *Ninth International Conference on Computing, Control, and Industrial Engineering (CCIE 2025)* (p. 78). SPIE. https://doi.org/10.1117/12.3093516

Singhal, S. (2025). A Hybrid Approach Combining Ant Colony Optimization and Simulated Annealing for Cloud Resource Scheduling. *Journal of Computational Systems and Applications*, *2*(2), 17–32. https://doi.org/10.63623/9rv3x042

Talaat, F. M., & Hamza, A. A. (2025). CloudSched-GA: An Adaptive Genetic Optimizer for Efficient and Balanced Task Scheduling in Cloud Ecosystems. *Neural Computing and Applications*, *37*(33), 28269–28293. https://doi.org/10.1007/s00521-025-11668-7

Tarigan, U., Siregar, I., Sari, R. M., Syahputri, K., & Rizkya, I. (2021). Comparison of First Come First Served and Ant Colony Algorithm Method for Door Leaf Production Scheduling. *IOP Conference Series: Materials Science and Engineering*, *1122*(1), 012057. https://doi.org/10.1088/1757-899X/1122/1/012057

Tian, Y., Zhang, J., Wang, Q., Liu, S., Guo, Z., & Zhang, H. (2024). Application of Hybrid Algorithm Based on Ant Colony Optimization and Sparrow Search in UAV Path Planning. *International Journal of Computational Intelligence Systems*, *17*(1), Article ID: 286. https://doi.org/10.1007/s44196-024-00652-z

Vidhya, M., & Devi, R. (2023). Analysis of Optimization and Conventional Algorithms Using CloudSim in Cloud. *2023 12th International Conference on System Modeling & Advancement in Research Trends (SMART)*, 403–408. https://doi.org/10.1109/SMART59791.2023.10428192

**JISKA (Jurnal Informatika Sunan Kalijaga)**
**Vol. 11, No. 3, SEPTEMBER, 2026, Pp. 366 – 379**
**ISSN: 2527 – 5836 (print) | 2528 – 0074 (online)**

## APPENDIX A

**Algorithm 1: Enhanced ACO Scheduling Algorithm**

**Input:**
- $J = \{j_1, j_2, \ldots, j_n\}$: Set of n jobs
- $V = \{v_1, v_2, \ldots, v_m\}$: Set of m virtual machines
- $\alpha$: Pheromone influence coefficient
- $\beta$: Heuristic influence coefficient
- $\rho$: Evaporation rate ($0 < \rho < 1$)
- $\omega$: Elitist weight factor
- **MaxIter:** Maximum number of iterations
- **NumAnts:** Number of ants
- **MaxSwaps:** Maximum number of swap attempts in local search

**Output:**
- $S_{best}$: **Optimal job-to-VM allocation**
- $M_{best}$: **Minimum makespan value**

**Initialize pheromone matrix τ with small constant;**
**Initialize heuristic matrix η = 1/ETC;**
$M_{best} \leftarrow \infty$;
$S_{best} \leftarrow \emptyset$;
**while iteration < MaxIter do:**
  **for each ant k in NumAnts do:** Ant Colony Phase
    $S_k \leftarrow \emptyset$;
    **for each job i in J do:**
      Calculate probability $P_{ij} = (\tau_{ij}{}^{\alpha} \times \eta_{ij}{}^{\beta}) / \Sigma(\tau_{il}{}^{\alpha} \times \eta_{il}{}^{\beta})$
      Select VM j using roulette wheel selection based on $P_{ij}$;
      Add assignment (job_i, VM_j) to $S_k$;
    **end for**
    // Swap-Based Local Search Optimization
    $S_{local} \leftarrow S_k$;
    $M_{local} \leftarrow$ CalculateMakespan($S_{local}$);
    **for swap in 1 to MaxSwaps do:**
      Randomly select two jobs (job 1, job 2);
      $S_{temp} \leftarrow$ Swap VM assignments of job 1 and job 2 in $S_{local}$;
      $M_{temp} \leftarrow$ CalculateMakespan($S_{temp}$);
      **if $M_{temp} < M_{local}$ then:**
        $S_{local} \leftarrow S_{temp}$;
        $M_{local} \leftarrow M_{temp}$;
      **end if**
    **end for**
    $S_k \leftarrow S_{local}$;
    $M_k \leftarrow M_{local}$;
  **end for**
  **if $M_{iter} < M_{best}$ then:**
    $M_{best} \leftarrow M_{iter}$;
    $S_{best} \leftarrow S_{iter}$;
  **end if**
  // Pheromone Update Phase
  // Evaporation
  $\tau_{ij} \leftarrow (1 - \rho) \times \tau_{ij}$ for all i,j;
  // Elitist Deposit (only global best)
  $\Delta\tau \leftarrow \omega / M_{best}$;
  **for each ($job_i$, $VM_j$) in $S_{best}$ do:**
    $\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau$;
  **end for**
  iteration ← iteration + 1;
**end while**
return $S_{best}$, $M_{best}$;