

# Optimizing job scheduling on cloud resources using the first-come, first-served-SlotFree method

Ardi Pujiyanta<sup>1</sup>, Fiftin Noviyanto<sup>2</sup>, Taufiq Ismail<sup>2</sup>

<sup>1</sup>Master of Electrical Engineering Study Program, Faculty of Industrial Technology, Universitas Ahmad Dahlan, Yogyakarta, Indonesia

<sup>2</sup>Department of Informatics, Faculty of Industrial Technology, Universitas Ahmad Dahlan, Yogyakarta, Indonesia

## Article Info

### Article history:

Received Dec 9, 2024

Revised Aug 8, 2025

Accepted Sep 1, 2025

### Keywords:

Cloud computing

First-come, first-served-

SlotFree

Scheduling

Utilization

Waiting time

## ABSTRACT

Cloud computing environments encounter significant challenges in job scheduling, particularly due to excessive waiting times and inefficient resource utilization associated with conventional algorithms such as first-come, first-served (FCFS) and backfilling. This study introduces FCFS-SlotFree, a novel scheduling algorithm that enhances resource allocation efficiency by dynamically sorting jobs based on their arrival times and workloads, and subsequently assigning them to a fixed set of virtual machines (VMs) without relying on rigid time-slot constraints. This flexible scheduling approach facilitates better adaptation to heterogeneous workloads. Extensive experiments conducted under realistic cloud scenarios demonstrate that FCFS-SlotFree significantly reduces average waiting time (AWT) by approximately 32.78% compared to FCFS and by 9.68% compared to backfilling, while concurrently improving resource utilization by 3.58% and 1.27%, respectively. The results substantiate the algorithm's effectiveness in optimizing scheduling performance and resource efficiency within complex cloud environments.

*This is an open access article under the [CC BY-SA](#) license.*



## Corresponding Author:

Ardi Pujiyanta

Master of Electrical Engineering Study Program, Faculty of Industrial Technology

Universitas Ahmad Dahlan

Jl. Ringroad Selatan, Tamanan, Banguntapan, Bantul, Daerah Istimewa Yogyakarta, 55191 Indonesia

Email: ardipujiyanta@tif.uad.ac.id

## 1. INTRODUCTION

Cloud computing revolutionizes information technology by providing on-demand resources through a network service model characterized by self-service, dynamic elasticity, scalability, pooled resource management, and extensive network access, thereby enhancing cost efficiency for both users and providers [1]. Task scheduling techniques, including branch-and-bound algorithms optimized for scalability, have been demonstrated through CloudSim to minimize makespan in cloud environments [2]. Efficient task allocation is crucial in cloud systems to optimize performance by distributing workloads, reducing makespan, and ensuring compliance with service level agreements (SLAs) [3]. Key challenges include mapping virtual machines (VMs) to user requests to balance throughput and resource utilization [4]. Heuristic batch-mode scheduling strategies that combine static and dynamic data have been shown to successfully allocate heterogeneous VM tasks, minimizing both makespan and variability [5]. Additionally, considerations of network bandwidth during allocation further enhance resource usage and reduce execution time during peak periods [6].

Optimized scheduling methods address workload allocation with an emphasis on reliability, energy efficiency, and task completion speed. Algorithms such as the grasshopper optimization algorithm (GOA) and dynamic multi-level task scheduling (DMLTS) tackle the complexities of large-scale scheduling,

achieving significant reductions in makespan and effective load balancing [7], [8]. VM management, essential for resource optimization, employs scheduling algorithms that minimize makespan and balance load by factoring in total VM execution time and workload [9], [10]. Hybrid threshold-based and heuristic scheduling approaches further enhance system stability in dynamic, heterogeneous VM environments [11]. Workload management aims to improve system throughput and resource utilization through adaptive scheduling techniques that address imbalances, employing methods such as improved threshold-based task scheduling (ITBTS) and multi-objective allocation that consider VM capacity and execution time [12]. Quality of service (QoS) factors, including response time, throughput, availability, and latency, shape scheduling decisions to meet deadlines and reduce costs, while heuristics are utilized to optimize provider profits and user satisfaction [13], [14]. Recent advancements in smart cloud management frameworks leverage knowledge bases to dynamically optimize resource configuration and SLA monitoring, thereby enhancing resource availability and autonomy [15]. Advanced scheduling algorithms, including multi-priority workflow grouping and adaptive round Robin modifications, effectively reduce waiting times while improving throughput and load distribution [16].

Backfilling scheduling techniques, which are evaluated based on prediction accuracy and performance, segment tasks into queues to enhance real-time processing without delaying earlier tasks, proving particularly effective under moderate to balanced workload conditions [17], [18]. Queueing models, such as M/G/1, analyze customer choices in resource pre-ordering to maximize revenues, addressing the challenges faced by Cloud Providers in VM placement to meet user requirements while considering scalability [19]. Static independent task scheduling assigns tasks based on VM availability, taking into account processing power, cost, and volume for effective grouping [20]. Research by Pujiyanta and Noviyanto [21] proposes a centralized scheduling model framework in which a single scheduler entity manages the entire process. Jobs submitted by users are first sequenced and then forwarded to the scheduler, which interacts with the logical view module to execute the tasks on VMs registered in the cloud information service (CIS). This approach provides more efficient and streamlined control, as a singular scheduling component oversees the distribution and execution of jobs. The centralized scheduler emphasizes simplicity and centralized management, with the aim of optimizing resource utilization and job throughput. Ultimately, this framework ensures efficient cloud job allocation and timely feedback to users.

The overall design of the scheduling architecture and its algorithms are influenced by several factors, including idle time, waiting time, resource availability, and the time period. These elements represent the primary challenges and constraints faced by existing resource scheduling algorithms. This study focuses on addressing these critical issues, with particular emphasis on the factors that impact job-to-VM mapping to achieve optimal scheduling outcomes.

## 2. METHOD

In the existing methods, jobs are scheduled to available processors based on job parameters using scheduling algorithms. Meanwhile, the proposed method schedules the given jobs to available processors using dynamic cloud scheduling and first-come, first-served (FCFS)-SlotFree algorithms. This approach is expected to provide significant improvement in execution time compared to the existing results. Figure 1 illustrates the proposed Cloud Job Scheduling system, which functions through a series of interconnected steps.

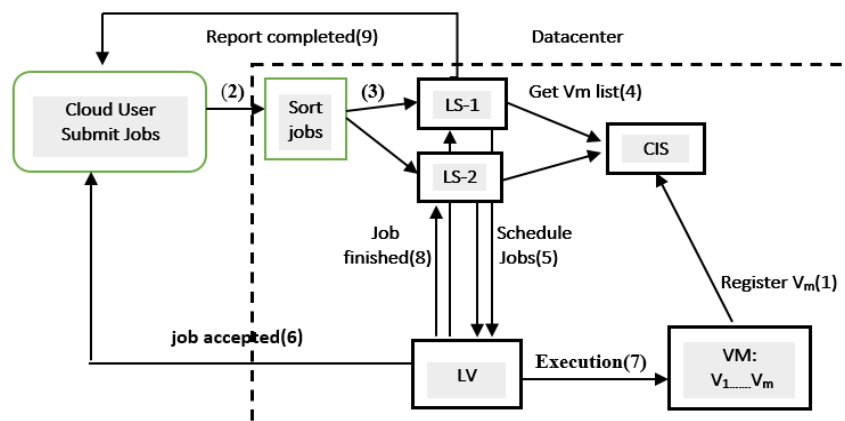


Figure 1. Job scheduling system in the cloud with dual LS

In Step 1, the available VMs within the data center are registered with the CIS, which maintains information about the VMs that are ready for deployment. Next, in Step 2, cloud users submit their jobs to the system, which then directs them to the subsequent sorting phase. Step 3 involves categorizing the submitted jobs based on specific criteria such as arrival time, priority, or job size, after which the jobs are divided and allocated to two load schedulers (LS-1 and LS-2). In Step 4, both LS request a list of available VMs from the CIS, which provides them with VMs prepared to execute the jobs. Step 5 elaborates on how LS-1 and LS-2 schedule the jobs to the appropriate VMs based on the available VM list, while also considering the respective availability and capacity of each VM. Once the scheduling is finalized, Step 6 indicates that the system confirms the acceptance of the scheduled jobs for execution, sending this confirmation back to the cloud user. In Step 7, the load validator (LV) executes the jobs on the designated VMs according to the established schedule. Following the execution in Step 8, the LV notifies LS-1 and LS-2 that the jobs have been completed. Finally, in Step 9, LS-1 and LS-2 deliver a completion report of the jobs to the cloud user, who subsequently receives the results of the executed jobs.

In this section, we present a novel scheduling algorithm designed to optimize resource utilization, minimize makespan, and reduce delay time within cloud environments.

Notation and preliminary definitions:

Set of VMs:  $V = \{ VM_1, VM_2, \dots, VM_m \}$

Set of jobs submitted by users:  $J = \{ J_1, J_2, \dots, J_n \}$

Function for registering VMs with the CIS: RegisterVM:  $V \rightarrow CIS$

Function for sorting jobs: SortJobs:  $J \rightarrow (J_1, J_2)$  where  $J_1, J_2 \subseteq J$  and  $J_1 \cup J_2 = J$

Function for retrieving the list of VMs from CIS: GetVMList:  $CIS \rightarrow V$

Function for scheduling jobs to VMs: ScheduleJobs:  $(J_i, V) \rightarrow Schedule_i$ , where  $i \in \{1, 2\}$

Function for executing jobs by the LV: Execute:  $Schedule_i \rightarrow ExecutionResult_i$

Function for reporting job results: ReportCompleted:  $ExecutionResult_i \rightarrow User$

where;  $A_i$ : arrival time of job  $i$  (arrival time),  $S_i$ : start time of job  $i$  execution (start time),  $W_i = S_i - A_i$ : waiting time of job  $i$ ,  $t_i$ : duration of job  $i$  execution,  $U_j$ : total time of VM  $j$  used for job execution,  $T$ : total observation time (from start to all jobs completed), and  $m$ : number of VMs.

To optimize job scheduling in a cloud computing environment, a FCFS-SlotFree is employed. This algorithm aims to minimize average waiting time (AWT) and enhance resource utilization. The details of the algorithmic steps can be found in Algorithm 1. Table 1 shows description of the steps in the proposed algorithm FCFS-SlotFree.

Algorithm 1. FCFS-SlotFree

1. Start
2. RegisterVM( $V$ )  $\rightarrow$  CIS
3. User submits job  $J$  with arrival times  $A_i$  and execution times  $t_i$
4.  $(J_1, J_2) = \text{SortJobs}(J)$
5.  $V \leftarrow \text{GetVMList}(CIS)$
6.  $Schedule1 = \text{ScheduleJobs}(J_1, V)$ 
  - For each job in  $Schedule1$ , record start time  $S_i$
7.  $Schedule2 = \text{ScheduleJobs}(J_2, V)$ 
  - For each job in  $Schedule2$ , record start time  $S_i$
8. Confirmation of job received to User
9.  $ExecutionResult1 = \text{Execute}(Schedule1)$
10.  $ExecutionResult2 = \text{Execute}(Schedule2)$
11. Calculate waiting time  $W_i$  for each job:
  - $W_i = S_i - A_i$
12. Calculate average waiting time:
  - $AverageWaitingTime = (\text{Sum of all } W_i) / (\text{Total number of jobs})$
13. Calculate resource utilization:
  - For each VM  $j$ :
    - $U_j = \text{Sum of execution times of jobs assigned to VM } j$
    - $T = \text{Total time from start of first job to end of last job}$
    - $ResourceUtilization = (\text{Sum of all } U_j) / (m * T) * 100\%$
14.  $\text{ReportCompleted}(ExecutionResult1, AverageWaitingTime, ResourceUtilization) \rightarrow User$
15.  $\text{ReportCompleted}(ExecutionResult2, AverageWaitingTime, ResourceUtilization) \rightarrow User$
16. End

Table 1. Brief explanation

Step	Symbols and functions	Short description
1	Start	
2	RegisterVM(V)	Register VM to CIS
3	J	User submit Jobs
4	SortJobs	Sort and assign jobs to J <sub>1</sub> , J <sub>2</sub>
5	GetVMList(CIS)	Get list of available VMs
6-7	ScheduleJobs(J <sub>i</sub> ,V)	Schedule jobs to VMs
8	Confirmation of job received to User	Job accepted
9-10	Execute(Schedule <sub>i</sub> )	Execute jobs by LV
11	For each job i: W <sub>i</sub> = S <sub>i</sub> – A <sub>i</sub>	Calculate waiting time W <sub>i</sub> for each job
12	AverageWaitingTime	Calculate AWT
13	ResourceUtilization	Calculate resource utilization
14-15	ReportCompleted(ExecutionResult <sub>i</sub> )	Send result report to user
16	End	

## 2.1. Performance metrics

Scheduling is the process of organizing a collection of tasks to determine the manner in which multiple tasks compete for access to one or more shared and reusable resources. These resources can encompass hardware components, including processors, communication channels, and storage devices, as well as software assets. Task scheduling further involves allocating specific resources to tasks by establishing their respective start and end times within defined constraints. As a fundamental aspect of cloud computing, the primary goal of scheduling is to ensure efficient allocation of resources for task execution. This allocation process, often referred to as task assignment, aims to optimize the distribution of tasks across available resources and is managed by the scheduler. To assess the efficiency and effectiveness of a scheduling algorithm, various performance metrics are utilized.

### – Utilization

In a cloud computing system, the nodes involved in service execution also experience certain levels of service utilization and satisfaction. During the execution of random tasks, not only is the load on the cloud system affected, but there is also a loss of waiting time. If the system operates for an extended period, the waiting time loss can become significant. Conversely, the longer a random task waits, the lower the service quality and satisfaction of the system [22]. The issue addressed in this paper is how to minimize the waiting time in the cloud computing system, improve resource utilization, and ensure that service requirements are met without task loss [23], [24].

### – Waiting time

There are instances when a resource is not available at the requested reservation time, but it can be reserved for a different time. The difference between the expected start time and the actual start time is known as the waiting time, represented in (1) and (2) [25], [26]:

Waiting time formula for job i:

$$W_i = S_i - A_i \quad (1)$$

AWT:

$$\bar{W} = \frac{1}{n} \sum_{i=1}^n W_i = \frac{1}{n} \sum_{i=1}^n (S_i - A_i) \quad (2)$$

where; n is total number of jobs, A<sub>i</sub> is arrival time of job I, S<sub>i</sub> is start time of job I, and W<sub>i</sub> is waiting time of job i.

## 2.2. Illustration of method

Table 2 displays the jobs submitted by the user, where *twet* represents the execution start time, *ts* is the flexible time, *te* indicates the execution duration, and *CN* denotes the number of resources needed. Before being assigned to a logical node, tasks are first sorted in ascending order. The results of this sorting are presented in Table 3. All tasks submitted by users are assigned to logical nodes. If there is available space, the task is allocated to that node. If no space is available, the task will be adjusted to accommodate the necessary flexible time. For instance, as illustrated in Figure 2, job number 2 has an execution time from t=0 to t=12. If there is space within this time slot, the task will be assigned to the logical view. Conversely, if no space is available, the task will be shifted to start at slot 12. Similarly, task number 5 is placed in slot number 6 because slots 3 to 6 are occupied by task number 3. After all tasks have been processed, the user is notified that their job has been accepted and is currently being processed, providing them with certainty regarding the acceptance or rejection of their task.

Table 2. Presents the jobs submitted by users

No	$t_{wet}$	$t_{pa}$	$t_e$	CN
1	0	23	6	2
2	0	0	7	2
3	0	0	5	1
4	0	0	5	5
5	0	12	6	1
6	2	8	7	1
7	5	5	5	2
8	5	5	6	4
9	6	18	5	2

Table 3. Jobs after sorting

No	$t_{wet}$	$t_{pa}$	$t_e$	CN
1	0	0	5	1
2	0	0	5	5
3	0	12	6	1
4	0	23	6	2
5	0	0	7	2
6	2	8	7	1
7	5	5	5	2
8	5	5	6	4
9	6	18	5	2

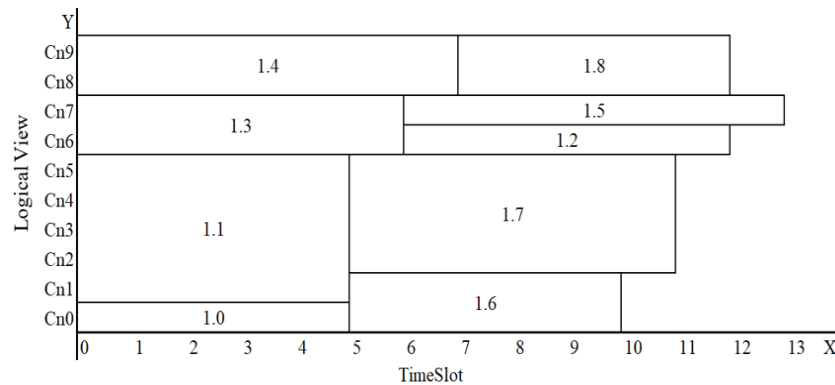


Figure 2. The placement of tasks for their execution

Figure 3 illustrates the placement of tasks scheduled for execution using the backfilling method. Task number 7 will be rejected because its initial execution time, starting from slot number 5 until the completion of task number 5, does not have the required available slot space. As shown in Figure 3, the job utilization level is lower compared to that of the proposed SlotFree algorithm. Additionally, the backfilling algorithm demonstrates uncertainty regarding the exact execution time of tasks; tasks are placed on hold until sufficient free space becomes available.

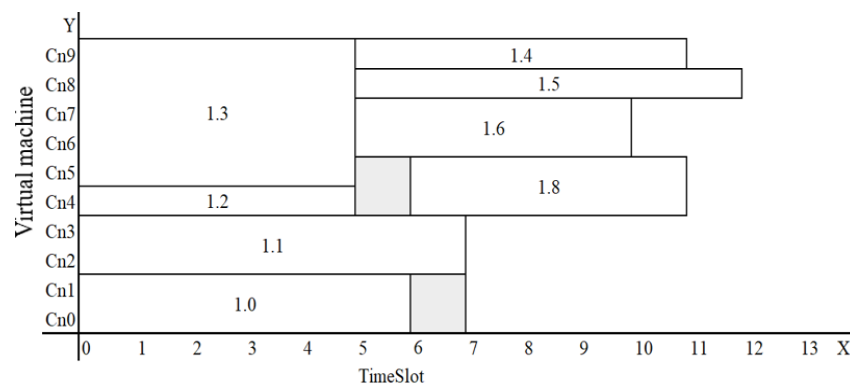


Figure 3. The placement of tasks for execution during the backfilling process

### 3. RESULTS AND DISCUSSION

The backfilling technique is employed as a basis for comparison because it advances reservations earlier to create space for incoming reservations. However, the subsequent job is required to stay in the waiting queue until the preceding job is fully executed, without assurance regarding its exact execution time. This limitation can cause inefficient resource usage since jobs might experience prolonged waiting periods. When a leading job demands more computing time than the currently available resources, it remains queued. The backfilling method permits jobs with shorter execution durations to move forward and execute on idle resources. Performance evaluation uses metrics such as delay time, waiting time, and job waiting time, aiming to optimize resource utilization.

The experimental setup for evaluating the proposed FCFS-SlotFree scheduling algorithm was meticulously designed to simulate a realistic cloud computing environment. A fixed number of 50 VMs, registered within the CIS, served as resources for job execution. These VMs were supported by one data center and 50 hosts, operating in space-sharing mode, which allowed only one job to run at a time on any given resource. Jobs, with sizes ranging from 300 to 700 units to reflect diverse workload complexities, were submitted by users and subsequently sorted based on their arrival times and workloads. The scheduler dynamically allocated these jobs to the available VMs without enforcing fixed time slots, thereby facilitating flexible task execution. Various computing scenarios were considered to observe key parameters such as resource utilization and job waiting time, as outlined in Table 4. To evaluate delay times, the FCFS-SlotFree method was compared against traditional FCFS and backfilling approaches. Performance metrics, including AWT and resource utilization rates, were collected by monitoring job start times, execution durations, and completion reports, which were then communicated to the users. Each set of executed jobs operated independently, ensuring an isolated assessment of scheduling efficiency. This comprehensive setup effectively validates the algorithm's capability to reduce scheduling delays and optimize resource utilization under heterogeneous task conditions typical of cloud systems.

Table 4. Experiment parameters

Name	Value
Execution time	Fixed
Number of resources	Fixed
Execution start time	Changed
Execution end time	Changed

Figure 4 depicts the AWT generated by each scheduling algorithm when applied to workloads of different sizes, illustrating that the proposed algorithm achieves a notable improvement in AWT. The results demonstrate that this algorithm surpasses other existing methods, with an average reduction in AWT of 32.78% compared to FCFS and 9.68% relative to backfilling. Overall, the proposed algorithm significantly decreases waiting times. In situations where the job's required execution time exceeds the available compute node time, the leading job in the queue must wait. The backfilling technique permits jobs with shorter durations to proceed and execute on idle compute nodes. However, in the backfilling algorithm, the subsequent job remains in the waiting queue until the previous job finishes execution, causing uncertainty about the actual start time of that job.



Figure 4. Displays the outcomes of AWT under various workloads

Figure 5 demonstrates the resource utilization attained by each algorithm across workloads of different sizes, indicating that the proposed algorithm significantly enhances resource utilization. The data reveal that the proposed approach outperforms other algorithms, achieving an average increase in resource utilization of 3.58% compared to FCFS and 1.27% relative to the backfilling method. Overall, the algorithm presents a marked improvement in resource usage. In scenarios where the required execution time surpasses the available compute node time, the foremost job in the queue must wait. The backfilling strategy allows jobs with shorter execution durations to proceed and execute on idle compute nodes. However, within the backfilling algorithm, subsequent jobs remain in the waiting queue until preceding jobs complete execution, resulting in uncertainty concerning the actual start times of these jobs.

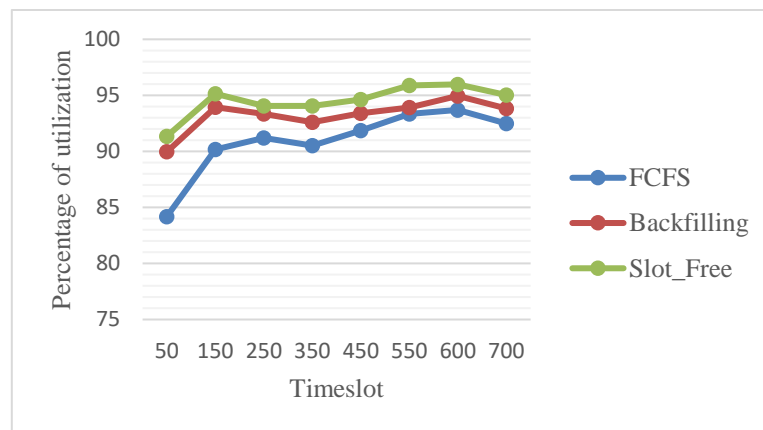


Figure 5. Resource utilization based on number of slots

#### 4. CONCLUSION

This research introduces a new scheduling algorithm called FCFS-SlotFree, designed to optimize task scheduling in cloud computing environments. Simulation results indicate that the algorithm is capable of reducing the average task waiting time by 32.78% compared to the conventional FCFS algorithm and by 9.68% compared to the backfilling method. In addition to reducing waiting times, FCFS-SlotFree also improves resource utilization by 3.58% compared to FCFS and by 1.27% compared to backfilling, thereby enhancing the overall system efficiency. This slot-free scheduling approach has proven effective in addressing the limitations of conventional FCFS algorithms and backfilling methods while improving scheduling performance in cloud computing. However, the focus of this research is limited to efforts aimed at reducing waiting times and increasing resource utilization, leaving other aspects such as energy consumption and system security unexplored in depth.

Future research could be directed toward the development of hybrid scheduling algorithms that integrate FCFS-SlotFree with optimization methods such as genetic algorithms or machine learning to enhance adaptability and performance under complex workloads. Additionally, evaluations in a broader and more diverse cloud computing environment are necessary to test the reliability and scalability of the algorithm, as well as to integrate scheduling with energy-saving strategies to promote more environmentally friendly cloud computing practices. Thus, the FCFS-SlotFree algorithm demonstrates a significant contribution to the efficiency of cloud computing scheduling and has the potential for widespread implementation to enhance the performance of cloud computing systems.

#### FUNDING INFORMATION

This study was funded by a research grant from the Ministry of Research, Technology, and Higher Education (Ristekdikti) of the Republic of Indonesia, under contract number 016/PFR/LPPM-UAD/VI/2024.

#### AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.



Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Ardi Pujiyanta	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Fiftin Noviyanto			✓	✓		✓		✓		✓	✓	✓		
Taufiq Ismail			✓	✓	✓		✓			✓	✓		✓	✓

C : Conceptualization

M : Methodology

So : Software

Va : Validation

Fo : Formal analysis

I : Investigation

R : Resources

D : Data Curation

O : Writing - Original Draft

E : Writing - Review &amp; Editing

Vi : Visualization

Su : Supervision

P : Project administration

Fu : Funding acquisition

## CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

## DATA AVAILABILITY

The simulation datasets generated and analyzed during the current study are available from the corresponding author upon reasonable request. The simulation datasets generated and analyzed during the current study are available from the corresponding author upon reasonable request.

## REFERENCES




- [1] R. Aron and A. Abraham, "Resource scheduling methods for cloud computing environment: The role of meta-heuristics and artificial intelligence," *Eng. Appl. Artif. Intell.*, vol. 116, p. 105345, 2022, doi: 10.1016/j.engappai.2022.105345.
- [2] Y. Benmouna and B. Benmammar, "Efficient Branch-and-Bound Algorithm for Task Scheduling in Cloud Computing," in *Proc. - Int. Conf. Adv. Comput. Inf. Technol. ACIT*, 2023, pp. 440–443, doi: 10.1109/ACIT58437.2023.10275480.
- [3] S. Nabi, M. Ibrahim, and J. M. Jimenez, "DRALBA: Dynamic and Resource Aware Load Balanced Scheduling Approach for Cloud Computing," *IEEE Access*, vol. 9, pp. 61283–61297, 2021, doi: 10.1109/ACCESS.2021.3074145.
- [4] S. K. Vishnoi and S. Patel, "Comparison of Average Completion Time and Makespan for Various Task Scheduling Techniques," in *2023 4th Int. Conf. Comput. Commun. Syst. I3CS 2023*, 2023, pp. 1–6, doi: 10.1109/I3CS58314.2023.10127383.
- [5] P. Banga and S. Rana, "Efficient Task Scheduling Technique under Batch Mode Heuristic for Cloud Environment," in *Proc. IEEE Int. Conf. Signal Process. Control*, 2021, pp. 433–437, doi: 10.1109/ISPCCS53510.2021.9609462.
- [6] S. Guruprasad and D. Shetty, "Efficient scheduling of tasks in cloud," in *Proc. 3rd Int. Conf. Intell. Commun. Technol. Virtual Mob. Networks, ICICV 2021*, 2021, pp. 1045–1049, doi: 10.1109/ICICV50876.2021.9388524.
- [7] H. Al-Zoubi, "Efficient Task Scheduling for Applications on Clouds," in *Proc. - 6th IEEE Int. Conf. Cyber Secur. Cloud Comput. CSCloud 2019 5th IEEE Int. Conf. Edge Comput. Scalable Cloud, EdgeCom 2019*, 2019, pp. 10–13, doi: 10.1109/CSCloud/EdgeCom.2019.00012.
- [8] D. Singh and A. Mittal, "Efficient Task Scheduling in a Cloud Environment based on Dynamic Priority and Optimized Technique," in *IEEE Int. Conf. Knowl. Eng. Commun. Syst. ICKES 2022*, 2022, pp. 1–5, doi: 10.1109/ICKECS56523.2022.10060053.
- [9] K. P. N. Jayasena, K. M. S. U. Bandaranayake, and B. T. G. S. Kumara, "TRET - A novel heuristic based efficient task scheduling algorithm in cloud environment," in *IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON*, 2020, pp. 812–817, doi: 10.1109/TENCON50793.2020.9293787.
- [10] K. M. S. U. Bandaranayake, K. P. N. Jayasena, and B. T. G. S. Kumara, "An Efficient Task Scheduling Algorithm using Total Resource Execution Time Aware Algorithm in Cloud Computing," in *Proc. - 2020 IEEE Int. Conf. Smart Cloud, SmartCloud 2020*, 2020, pp. 29–34, doi: 10.1109/SmartCloud49737.2020.00015.
- [11] V. K. Pallavi, R. Trupthi, G. Varnitha, and T. G. K. Kumar, "Improvised Threshold Based Task Scheduling," in *Proc. 2nd Int. Conf. Electron. Sustain. Commun. Syst. ICESC 2021*, 2021, pp. 496–501, doi: 10.1109/ICESC51422.2021.9532666.
- [12] S. Singh, R. Kumar, and D. Singh, "An empirical investigation of task scheduling and VM consolidation schemes in cloud environment," *Comput. Sci. Rev.*, vol. 50, 2023, doi: 10.1016/j.cosrev.2023.100583.
- [13] A. Goyal, R. Garg, and K. K. Bhatia, "Models and Challenges Categorization in Cloud Computing," in *Proc. - 2021 Int. Conf. Comput. Sci. ICCS 2021*, 2021, pp. 34–37, doi: 10.1109/ICCS54944.2021.00015.
- [14] M. S. Qureshi, M. B. Qureshi, M. Fayaz, M. Zakarya, S. Aslam, and A. Shah, "Time and cost efficient cloud resource allocation for real-time data-intensive smart systems," *Energies*, vol. 13, no. 21, pp. 1–26, 2020, doi: 10.3390/en13215706.
- [15] H. S. Chyad, R. A. Mustafa, and D. N. George, "Cloud resources modelling using smart cloud management," *Bull. Electr. Eng. Informatics*, vol. 11, no. 2, pp. 1134–1142, 2022, doi: 10.11591/eei.v11i2.3286.
- [16] N. Ghazy, A. Abdelkader, M. S. Zaki, and K. A. Eldahshan, "An ameliorated Round Robin algorithm in the cloud computing for task scheduling," *Bull. Electr. Eng. Informatics*, vol. 12, no. 2, pp. 1103–1114, 2023, doi: 10.11591/eei.v12i2.4524.
- [17] S. M. F. D. S. Mustapha and P. Gupta, "DBSCAN inspired task scheduling algorithm for cloud infrastructure," *Internet Things Cyber-Physical Syst.*, vol. 4, pp. 32–39, 2024, doi: 10.1016/j.iotcps.2023.07.001.
- [18] M. Ibrahim et al., "A Comparative Analysis of Task Scheduling Approaches in Cloud Computing," in *Proc. - 20th IEEE/ACM Int. Symp. Clust. Cloud Internet Comput. CCGRID 2020*, 2020, pp. 681–684, doi: 10.1109/CCGrid49817.2020.00-23.
- [19] M. Arora, V. Kumar, and M. Dave, "Task scheduling in cloud infrastructure using optimization technique genetic algorithm," in *Proc. World Conf. Smart Trends Syst. Secur. Sustain. WS4 2020*, 2020, pp. 788–793, doi: 10.1109/WorldS450073.2020.9210303.
- [20] V. A. Lepakshi and C. S. R. Prashanth, "Efficient Resource Allocation with Score for Reliable Task Scheduling in Cloud Computing Systems," in *2nd Int. Conf. Innov. Mech. Ind. Appl. ICIMIA 2020 - Conf. Proc.*, 2020, pp. 6–12, doi: 10.1109/ICIMIA48430.2020.9074914.






- [21] A. Pujiyanta and F. Noviyanto, "Job scheduling reservations on cloud resources," *Int. J. Adv. Intell. Informatics*, vol. 10, no. 3, pp. 460–470, 2024, doi: 10.26555/ijain.v10i3.1421.
- [22] S. De, "An efficient technique of resource scheduling in cloud using graph coloring algorithm," *Glob. Transitions Proc.*, vol. 3, no. 1, pp. 169–176, 2022, doi: 10.1016/j.gltp.2022.03.005.
- [23] N. C. Brintha, J. T. W. Jappes, M. A. Khan, and A. Ajithram, "Optimal resource scheduling for SMEs using cloud in manufacturing sectors," *Mater. Today Proc.*, vol. 60, pp. 1480–1486, 2022, doi: 10.1016/j.matpr.2021.11.446.
- [24] M. T. A. Siddique, S. Sharmin, and T. Ahammad, "Performance Analysis and Comparison among Different Task Scheduling Algorithms in Cloud Computing," in *2020 2nd Int. Conf. Sustain. Technol. Ind. 4.0, STI 2020*, 2020, pp. 19–20, doi: 10.1109/STI50764.2020.9350466.
- [25] J. Anand and B. Karthikeyan, "EADRL: Efficiency-aware adaptive deep reinforcement learning for dynamic task scheduling in edge-cloud environments," *Results Eng.*, vol. 27, pp. 1-16, 2025, doi: 10.1016/j.rineng.2025.105890.
- [26] D. Dharrao *et al.*, "Multi-component attention graph convolutional neural network for QoS-aware cloud job scheduling and resource management enhancing efficiency and performance in cloud computing," *Results Eng.*, vol. 27, pp. 1-11, 2025, doi: 10.1016/j.rineng.2025.105676.

## BIOGRAPHIES OF AUTHORS






**Ardi Pujiyanta**    received his Engineering degree in Nuclear Engineering from Universitas Gadjah Mada in 1992. He obtained his Master's degree in Informatics Engineering from the Faculty of Electrical Engineering, Department of Electrical and Informatics, in 2002. He was awarded his Doctorate degree in Informatics Engineering from the Faculty of Electrical Engineering, Department of Electrical and Informatics, in 2020 at Universitas Gadjah Mada. He is currently a Lecturer in the Master's program in Electrical Engineering at Universitas Ahmad Dahlan Yogyakarta. He can be contacted at email: ardupujiyanta@tif.uad.ac.id.



**Fiftin Noviyanto**    received a Bachelor's degree in Informatics Engineering from Universitas Ahmad Dahlan in 2003. He obtained a Master's degree in Computer Science from the Faculty of Computer Science at Universitas Gadjah Mada in 2011. He is currently enrolled in a Doctoral program in Informatics at the National University of Malaysia (UKM), Faculty of Science and Information Technology. He is currently a Lecturer in the Informatics program at Universitas Ahmad Dahlan Yogyakarta. He can be contacted at email: fiftin.noviyanto@tif.uad.ac.id.



**Taufiq Ismail**    received a Bachelor's degree in Informatics Engineering from Ahmad Dahlan University in 1998. He obtained a Master's degree in Computer Science from the Faculty of Computer Science at Gadjah Mada University in 2010. He is currently a Lecturer in the Informatics program at Ahmad Dahlan University in Yogyakarta. He can be contacted at email: taufiq@tif.uad.ac.id.