Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

June 2026. Vol. 5. No. 3; e-ISSN: 2808-4519

Load Balancing Optimization in Cloud Task Scheduling Using Genetic Algorithm

Fellycia Caroline[1]*, Yohannes[2]

[1,2] *Informatics, Faculty of Computer Science and Engineering, Universitas Multi Data Palembang*
fellyciacaroline_2327250010@mhs.mdp.ac.id[1] *, yohannesmasterous@mdp.ac.id [2]

Abstract

Cloud computing environments face significant challenges in task scheduling and load balancing due to the increasing scale and complexity of computing workloads. Inefficient task scheduling leads to uneven resource utilization, increased makespan, and higher operational costs. This research proposes load balancing optimization in cloud task scheduling using a *Genetic Algorithm* applied to the Cloud Task Scheduling Dataset. The dataset underwent preprocessing including categorical encoding, data cleaning, and Min-Max Normalization prior to the optimization process. The *Genetic Algorithm* was implemented using Tournament Selection, Two-Point Crossover, and Uniform Integer Mutation, with the fitness function formulated based on makespan and degree of load imbalance minimization. The performance of the proposed approach was evaluated against a Random Assignment baseline across five metrics: makespan, degree of load imbalance, load distribution efficiency, average per-task completion time, and computational cost. The results demonstrated that the *Genetic Algorithm* significantly outperformed, achieving a makespan reduction of 51.33%, a load imbalance reduction of 43.81%, a load distribution efficiency improvement of 11.24%, an average per-task completion time reduction of 26.37%, and a computational cost reduction of 19.70%. These findings confirm that the *Genetic Algorithm* is an effective approach for optimizing task scheduling and load balancing in cloud computing environments.

Keywords: *Cloud Computing; Genetic Algorithm; Load Balancing; Task Scheduling; Virtual Machine*

1. Introduction

The development of cloud computing technology has driven the increased use of computing services that utilize resources via the internet to flexibly meet user needs. As the scale of the cloud increases and the variety of applications and tasks being executed increases, the demand for system performance has also increased significantly [1]. Cloud computing is a distributed system widely used for data processing and storage. As the amount of data being processed increases, managing and scheduling tasks in cloud environments becomes increasingly complex [2].

One of the essential components of a cloud computing environment is task scheduling, which is the process of allocating tasks to available computing resources so that execution can run efficiently. Although cloud computing offers various advantages, the task scheduling process remains a significant challenge because it impacts the efficiency and effectiveness of the cloud system [3]. In addition, the task scheduling process in a cloud environment is a complex multi-objective optimization problem because it involves various performance parameters that need to be considered, such as execution time, cost, and energy consumption [4].

A suboptimal scheduling process can impact the distribution of resource usage in a cloud environment. Inefficient task scheduling algorithms can lead to excessive or suboptimal resource usage, which can degrade the quality of system services [5]. In addition, the scheduling process in a cloud environment needs to pay attention to several performance parameters such as makespan, resource utilization rate, execution cost, and response time so that the system can run efficiently [6].

Various optimization methods have been developed to address task scheduling and load balancing issues in cloud environments, one of which uses a metaheuristic approach. *Genetic Algorithms* (GAs) are widely used in task scheduling problems due to their excellent solution-finding capabilities in large search spaces [7]. In addition, the application of the *Genetic Algorithm* approach combined with other optimization methods can increase resource utilization and reduce the level of load imbalance in cloud environments [8].

Previous studies have demonstrated the importance of optimization techniques in improving scheduling performance in cloud environments. Previous studies showed that *Genetic Algorithm*-based task scheduling approaches were capable of improving scheduling performance by optimizing completion time, execution cost, and resource utilization in cloud environments [9]. Furthermore, load balancing has been identified as an important factor affecting workload distribution and overall cloud system performance [10]. In addition, previous studies indicated that scheduling optimization techniques can improve service performance and increase resource allocation efficiency in cloud environments [11].

Although previous studies have shown promising results in improving scheduling performance, most existing studies mainly focus on multiple scheduling objectives without specifically emphasizing workload distribution among available computing resources. Uneven workload distribution may reduce resource utilization efficiency and create bottlenecks that affect overall cloud performance. Therefore, load balancing optimization remains an important issue that requires further investigation.
Based on these challenges, this research proposes load balancing optimization in cloud task scheduling using a *Genetic Algorithm* by utilizing the Cloud Task Scheduling Dataset obtained from Kaggle. The dataset contains information related to task and resource characteristics that can be used to simulate scheduling scenarios in cloud environments. This research aims to achieve a more balanced workload distribution among *Virtual Machine*s (VMs) to improve resource utilization and overall system performance based on the selected evaluation parameters.

2. Research Methods

This research uses an experimental method by applying a *Genetic Algorithm* (GA) to optimize load balancing in the cloud task scheduling process. A *Genetic Algorithm* is an algorithm that utilizes the process of natural selection, also known as the evolutionary process, as proposed by Charles Darwin. In the evolutionary process, individuals continuously change their genes to adapt to their environment [12]. The research was conducted using the Cloud Task Scheduling Dataset obtained from Kaggle as simulation data for the task scheduling process in a cloud computing environment. The goal of this research is to obtain a more even load distribution on *Virtual Machines* (VMs) to improve resource efficiency and system performance. The stages of research carried out are shown in Figure 1.

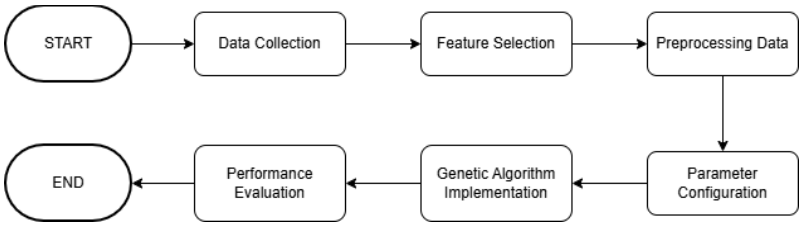

Fig. 1: Research Flow

2.1. Research Datasets

The dataset used in this research is the Cloud Task Scheduling Dataset obtained from Kaggle, available at https://www.kaggle.com/datasets/programmer3/cloud-task-scheduling-dataset. The dataset was selected because it provides comprehensive information related to task scheduling, *Virtual Machine* resource allocation, and system performance in a cloud computing environment. It is considered suitable for this study because the variables contained in the dataset can represent task distribution and workload balancing conditions required for evaluating *Genetic Algorithm*-based optimization.
The dataset consists of 6,000 records and 22 attributes, where each record represents a task processed in a distributed cloud computing environment. The attributes describe several aspects of cloud task scheduling, including task characteristics, *Virtual Machine* specifications, execution performance, resource consumption, and scheduling outcomes.

Table 1: Dataset Attribute Description

Attribute Name	Description
Task_ID	Unique identifier assigned to each task in the dataset.
Task_Length_MIPS	Represents the computational workload required by a task.
Task_Deadline	Defines the maximum allowable completion time for a task.
Data_Upload_Size_MB	Indicates the size of input data uploaded before task execution.
Data_Download_Size_MB	Indicates the size of output data downloaded after task completion.
VM_ID	Unique identifier assigned to the *Virtual Machine* executing the task.
VM_MIPS	Represents the processing capability of the *Virtual Machine*.
VM_Memory_GB	Describes the amount of memory allocated to the *Virtual Machine*.
VM_Bandwidth_MBps	Represents the available network bandwidth of the *Virtual Machine*.
Resource_Type	Indicates the type or category of resource assigned for task execution.
Task_Priority	Represents the priority level assigned to the task in the scheduling process.
Execution_Time_S	Indicates the actual execution time required to complete the task.
Waiting_Time_S	Represents the waiting time before task execution begins.
Completion_Time_S	Indicates the total completion time from submission until execution finishes.
Energy_Consumption_J	Represents the amount of energy consumed during task execution.
Scheduling_Algorithm	Indicates the scheduling algorithm associated with task processing.
Makespan_S	Represents the total completion time of all scheduled tasks.
Response_Time_S	Indicates the elapsed time between task submission and system response.
Execution_Cost_$	Represents the estimated execution cost incurred during task processing.
Degree_of_Imbalance	Measures how evenly workloads are distributed among *Virtual Machine*s.
Storage_Utilization	Represents the amount of storage resource used during task execution.
Path_Load	Indicates the workload level on the communication or network path during execution.

In cloud computing, task scheduling plays an important role in assigning tasks to available resources efficiently. The dataset reflects this scheduling process by providing information such as computational workload, execution time, waiting time, resource utilization, and system performance metrics. These variables are useful in analyzing how tasks are distributed among *Virtual Machine*s and how workload balancing can be improved. The attributes contained in the dataset are presented in Table 1.
Based on Table 1, the dataset contains both task-related attributes and resource-related attributes. Task-related attributes include task workload, deadline, priority, execution time, and completion time, while resource-related attributes describe *Virtual Machine* capabilities such as processing power, memory, bandwidth, and storage utilization. In addition, the dataset also includes performance evaluation metrics

such as makespan, execution cost, energy consumption, and degree of imbalance. Overall, this dataset provides a representative simulation of distributed task scheduling in cloud computing environments. Therefore, it serves as an appropriate data source for analyzing task allocation and evaluating load balancing optimization using the *Genetic Algorithm* in this research.

2.2. Feature Selection

The feature selection stage is performed to select attributes that are relevant to the task scheduling and load balancing processes. Feature selection aims to reduce the use of irrelevant attributes, allowing the optimization process to run more effectively. The features used in this research can be seen in Table 2.

Table 2: Research Features

Feature	Explanation
Task_Length_MIPS	Task computing load
Task_Deadline	Task completion deadline
VM_MIPS	VM processing capacity
VM_Memory_GB	VM memory capacity
VM_Bandwidth_MBps	VM data transfer capacity
Execution_Time_S	Execution time
Makespan_S	Total completion time
Degree_of_Imbalance	Degree of load imbalance
Resource_Type	Resource type

2.3. Preprocessing Data

The preprocessing stage is performed to prepare the data before it is used in the optimization process using a *Genetic Algorithm*. This stage aims to ensure the data is in the appropriate format and can be processed properly during the optimization process. The preprocessing stages carried out in this research include:

1. Categorical Data Transformation
   The Resource_Type variable is categorical data so it needs to be converted into numeric form using encoding techniques so that it can be processed by the algorithm. The transformation is carried out with the following rules at Table 3.

   Table 3: Resource Type Variable Encoding Results

Resource type	Value
Cloud	0
Fog	1
Edge	2

   The transformation results are stored in a new attribute, namely Resource_Type_Encoded.
2. Data Cleaning
   Some numeric attributes in the dataset are still stored in string format and have a dot (.) character as a number separator, which can interfere with the calculation process. Attributes that are subject to cleaning include: Task_Length_MIPS, Task_Deadline, VM_MIPS, VM_Memory_GB, VM_Bandwidth_MBps, Execution_Time_S, Makespan_S, and Degree_of_Imbalance. The cleaning process is carried out by removing the dot (.) character and changing the data format to numeric (float).
3. Data Normalization
   Normalization is carried out using the Min-Max Scaling method to equalize the range of values between variables so that there is no dominance of attributes with a larger scale in the optimization process. Normalized attributes include: Task_Length_MIPS, Task_Deadline, VM_MIPS, VM_Memory_GB, and VM_Bandwidth_MBps. Normalization is done using the Equation 1 [13]:

   $$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1)$$

   Description:
   x = initial data value
   x_{min} = minimum data value
   x_{max} = maximum data value
   x' = normalized value

2.4. Parameter Configuration

The parameter configuration used in the implementation of the *Genetic Algorithm* in this research is presented in Table 4.

Table 4: *Genetic Algorithm* Parameter Configuration

Parameter	Value
Population Size	200
Number of Generations	100
Crossover Probability	0.7
Mutation Probability	0.2
Individual Gene Mutation Probability	0.1
Number of Tasks	6000
Number of VMs	20
Selection Method	Tournament Selection (tournsize = 3)
Crossover Method	Two-Point Crossover
Mutation Method	Uniform Integer Mutation
Fitness Weight w1 (Makespan)	0.5
Fitness Weight w2 (Load Imbalance)	0.5

2.5. Implementation of *Genetic Algorithm*

In this research, *Genetic Algorithm* is used to determine the optimal task scheduling solution. The overall flow of the GA implementation is illustrated in Figure 2.

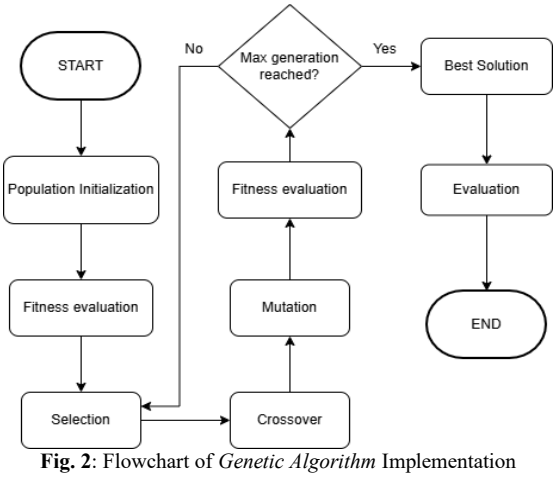

Fig. 2: Flowchart of *Genetic Algorithm* Implementation

The stages of the *Genetic Algorithm* include [14]:

1. Population Initialization
   The initial stage of generating parent individuals based on a predetermined population size through a random process is referred to as population initialization. Population Initialization represents the first phase in the *Genetic Algorithm* process, where a set of chromosomes is randomly generated to produce candidate solutions for subsequent optimization processes.
2. Selection
   The selection process is a stage aimed at generating a new population in each generation by evaluating and ranking all individuals along with their offspring based on their fitness values. Individuals with superior fitness values are then selected to survive and proceed to the next generation, ensuring that higher-quality solutions are retained throughout the evolutionary process.
3. Crossover
   Crossover is one of the reproduction mechanisms in *Genetic Algorithm* that can be implemented using the *one-cut point* method. This method operates by randomly selecting two chromosomes as parent individuals. A single crossover point is then determined, dividing each parent chromosome into two segments. Subsequently, the chromosome segments are exchanged between the parent individuals to generate new offspring as candidate solutions for the next generation.
4. Mutation
   Mutation is a reproduction process in *Genetic Algorithm* that aims to introduce random changes within a chromosome structure. The mutation mechanism is performed by modifying specific genes to generate variations in the population, where a single mutation process can produce one offspring from one parent individual. In this study, mutation is implemented using the *Uniform Integer Mutation* approach, which randomly alters selected genes within a chromosome by replacing their values with new integer values within a predefined range.
5. Fitness Evaluation
   Evaluation is a process used to assess and determine the quality of individuals based on the calculated *fitness* values. This stage aims to measure the performance of each chromosome in representing a potential solution. The *fitness* value serves as an indicator for evaluating chromosome quality. In this research, the *fitness* function is formulated based on *makespan* and load imbalance criteria; therefore, the optimization process follows a minimization approach, where chromosomes with lower *fitness* values are considered more optimal. Lower *fitness* values indicate better scheduling performance due to reduced *makespan* and a lower degree of load imbalance. The fitness function used in this study is presented in Equation (2), while the *makespan* and *load imbalance* formulations are shown in Equations (3) and (4), respectively [6, 15].

$$Fitness = w_1(Makespan) + w_2(LI) \tag{2}$$

With

$$Makespan = \max(VM_{load}) \tag{3}$$

$$LI = \frac{\max(VM_{load}) - \min(VM_{load})}{mean(VM_{load})} \tag{4}$$

Description:
Fitness = an evaluation value used to determine the quality of scheduling solutions generated by the *Genetic Algorithm*.
w_1= weight coefficient for the Makespan parameter.
w_2= weight coefficient for the Load Imbalance parameter.
Makespan = the maximum completion time of all tasks executed on *Virtual Machine*s.
LI = *Load Imbalance*, used to measure the level of workload imbalance among *Virtual Machine*s.
VM_{load}= the total workload or execution time assigned to each *Virtual Machine*.
$\max(VM_{load})$= the maximum workload among all *Virtual Machine*s.
$\min(VM_{load})$= the minimum workload among all *Virtual Machine*s.
$mean(VM_{load})$= the average workload of all *Virtual Machine*s.
$w_1 + w_2 = 1$= the total sum of weights used in the fitness function.

2.6. Evaluation of Results

The evaluation stage is performed to assess the effectiveness of the *Genetic Algorithm* in improving load balancing for cloud task scheduling. The assessment is conducted by comparing the scheduling outcomes generated by the GA-based optimization method with a baseline Random Assignment method, where tasks are allocated randomly across available VMs without applying any optimization technique. The performance indicators utilized in this study are described as follows:

1. Makespan
   Represents the highest load among all VMs after task allocation, as expressed in Equation (3). A smaller makespan value reflects better scheduling efficiency.
2. Degree of Load Imbalance
   Determined using Equation (4), which quantifies the proportion between the difference between maximum and minimum VM loads and the average VM load. A lower LI value indicates a more balanced workload distribution across VMs.
3. Load Distribution Efficiency
   Load Distribution Efficiency evaluates the extent to which workloads are evenly allocated across all VMs and is expressed as a percentage. This metric is directly calculated from the load imbalance value based on Equation (5). A higher efficiency value reflects a more balanced task distribution among VMs.

$$\text{Efficiency} = \frac{100}{1 + LI} \tag{5}$$

   Description:
   Eficiency = load distribution efficiency expressed as a percentage
   LI= degree of load imbalance as defined in Equation (4)
4. Average Per-Task Completion Time
   Average Per-Task Completion Time measures the average time required to complete each individual task. The execution time for each task is initially determined based on the task length and the processing capability of the assigned VM, as defined in Equation (6). Subsequently, the completion time of each task is calculated cumulatively by considering the execution times of previous tasks assigned to the same VM, as presented in Equation (7). The average value is then computed across all Ntasks using Equation (8). A lower value indicates better responsiveness of the scheduling process toward individual tasks.

$$ET_i = \frac{\text{Task\_Length}_i}{\text{VM\_MIPS}_{v(i)}} \tag{6}$$

$$LCT_i = \sum_{j \in \Omega_{v(i),<i}} ET_j + ET_i \tag{7}$$

$$\overline{CT} = \frac{1}{N} \sum_{i=1}^{N} CT_i \tag{8}$$

   Description:
   ET_i = execution time of task $i$in seconds
   Task_Length_i= computational length of task $i$in MIPS
   $\text{VM\_MIPS}_{v(i)}$= processing capacity of the VM assigned to task $i$in MIPS
   $v(i)$= index of the VM assigned to task i
   CT_i= completion time of task $i$in seconds
   $\Omega_{v(i),<i}$= set of tasks assigned to the same VM as task $i$and scheduled before it
   $\overline{CT}$= average completion time across all tasks in seconds
   N= total number of tasks
5. Computational Cost
   Computational Cost measures the overall monetary expense generated by the scheduling solution. It is determined by multiplying the total execution time of all tasks by a predefined cost rate per second, as expressed in Equation (9). A lower computational cost value indicates a more cost-efficient scheduling outcome.

$$\text{Cost} = \left(\sum_{i=1}^{N} ET_i \right) \times c \tag{9}$$

   Description:
   Cost = total computational cost in USD
   N= total number of tasks
   ET_i= execution time of task $i$as defined in Equation (6)
   c = cost rate per second ($c = \$0.01$/second)
6. Improvement Percentage
   The improvement percentage for each metric is calculated to measure the performance enhancement achieved by the GA-based optimization approach compared to the Random Assignment baseline, as presented in Equation (10). For metrics where lower values indicate better performance (Makespan, Load Imbalance, Average Per-Task Completion Time, and Computational Cost), a

positive improvement percentage signifies that the GA approach provides superior results. In contrast, for Load Distribution Efficiency, where higher values indicate better performance, the numerator is adjusted accordingly by reversing its order.

$$\text{Improvement} = \frac{\text{Metric}_{\text{Random}} - \text{Metric}_{\text{GA}}}{\text{Metric}_{\text{Random}}} \times 100\% \quad (10)$$

Description:
$\text{Metric}_{\text{Random}}$= metric value of the Random Assignment baseline
$\text{Metric}_{\text{GA}}$= metric value of the GA-optimized solution

3. Results & Discussion

3.1. Preprocessing Results

The preprocessing stage was carried out on the Cloud Task Scheduling Dataset consisting of 6,000 records and 22 attributes. After the feature selection stage, 9 relevant attributes were retained for the optimization process. The preprocessing results are presented in Table 5.

Table 5: Descriptive Statistics of Dataset After Preprocessing

Attribute	Count	Mean	Std	Min	25%	50%	75%	Max
Task_Length_MIPS	6000	0.4986	0.2877	0.0000	0.2495	0.5007	0.7457	1.0000
Task_Deadline	6000	0.1846	0.1677	0.0000	0.0706	0.1584	0.2476	1.0000
VM_MIPS	6000	0.5002	0.2885	0.0000	0.2502	0.4993	0.7464	1.0000
VM_Memory_GB	6000	0.3703	0.3509	0.0000	0.1429	0.1429	0.4286	1.0000
VM_Bandwidth_MBps	6000	0.3669	0.3926	0.0000	0.0000	0.3333	1.0000	1.0000
Execution_Time_S	6000	1.161e+16	1.151e+16	5.880e+12	4.956e+15	9.382e+15	1.307e+16	6.391e+16
Makespan_S	6000	1.290e+16	1.369e+16	5.374e+12	4.180e+15	6.261e+15	1.861e+16	6.376e+16
Degree_of_Imbalance	6000	1.262e+16	6.466e+15	1.874e+12	1.219e+16	1.478e+16	1.739e+16	1.999e+16
Resource_Type_Encoded	6000	0.6967	0.7864	0.0000	0.0000	0.0000	1.0000	2.0000

.

The Resource_Type attribute was encoded into numeric values, where Cloud, Fog, and Edge were mapped to 0, 1, and 2 respectively. Data cleaning was then performed on numeric attributes that were stored in string format with dot (.) separators, converting them to float format. Finally, Min-Max Normalization was applied to 5 attributes, namely Task_Length_MIPS, Task_Deadline, VM_MIPS, VM_Memory_GB, and VM_Bandwidth_MBps, resulting in all normalized values ranging between 0 and 1. The normalized attributes show a mean value of 0.4986 for Task_Length_MIPS and 0.5002 for VM_MIPS, indicating a relatively uniform distribution of task and VM characteristics across the dataset. No missing values were found in the dataset, so no imputation was required. The final dataset used in the optimization process consists of 6,000 records and 10 attributes including the newly added Resource_Type_Encoded attribute.

3.2. *Genetic Algorithm* Convergence

The *Genetic Algorithm* was executed over 100 generations with a population size of 200 individuals, completed in 94.15 seconds. Table 6 presents the progression of the minimum fitness, average fitness, and standard deviation values across selected generations, while Figure 2 illustrates the convergence curve of the GA.

Table 6: GA Fitness Progression Across Selected Generations

Generation	Nevals	Min Fitness	Avg Fitness	Std
0	200	1179.85	1348.42	76.4651
10	161	1012.44	1100.44	33.5974
20	156	921.169	979.881	29.1413
30	159	860.984	908.221	27.1673
40	147	804.442	839.117	28.7207
50	161	755.536	796.497	29.6807
60	147	731.336	766.316	31.9425
70	157	712.278	742.554	32.9148
80	148	703.938	725.785	32.0152
90	156	697.748	713.596	30.4284
100	151	697.012	712.609	32.5899

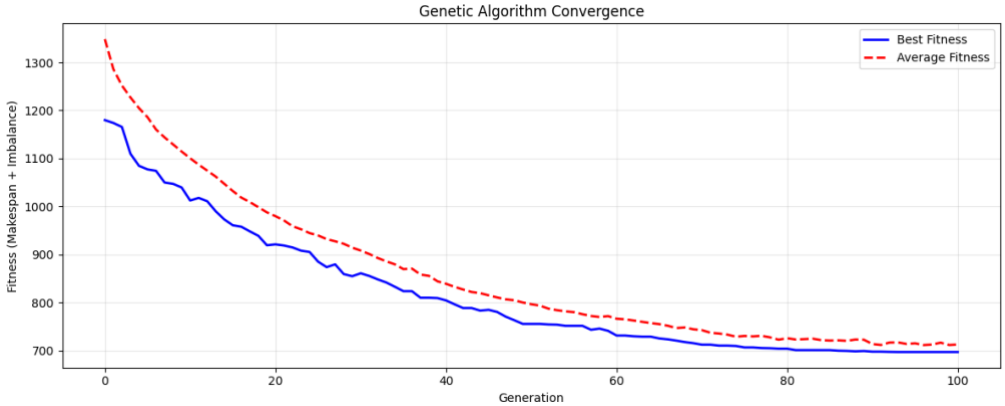

Fig. 3: *Genetic Algorithm* Convergence

As shown in Figure 3, both the minimum and average fitness values consistently decreased throughout the evolutionary process. The fitness value dropped significantly in the early generations, from 1179.85 at generation 0 to 873.751 at generation 26, demonstrating rapid exploration of the solution space in the initial phase. The rate of improvement gradually slowed in the later generations, with the minimum fitness value stabilizing around 697.012 from generation 93 onward, indicating convergence toward a near-optimal solution. The gap between the best fitness and average fitness also narrowed progressively across generations, reflecting the increasing quality and consistency of the population as the algorithm evolved.

3.3. *Genetic Algorithm* Results

The best scheduling solution produced by the GA achieved a final best fitness value of 697.0118. The final load distribution across the 20 VMs under the GA-optimized solution is presented in Table 7.

Table 7: Final VM Load Distribution (GA Optimized)

VM ID	Load (seconds)
0	302.5012
1	1391.0902
2	1385.1545
3	177.5855
4	298.8169
5	692.9185
6	1133.7754
7	196.2672
8	719.0048
9	340.0155
10	687.6143
11	264.2797
12	1391.9607
13	421.3771
14	164.0657
15	1217.9319
16	198.8221
17	196.6415
18	452.6508
19	271.7228

The results show that VM 12 received the highest load at 1391.9607 seconds, while VM 14 carried the lowest load at 164.0657 seconds. The final makespan achieved was 1391.9607 seconds with a load imbalance of 2.0630.

3.4. Performance Comparison

The GA-optimized solution was compared against the Random Assignment baseline in terms of makespan and load imbalance. The load distribution comparison between the two approaches is illustrated in Figure 3, while the performance improvement summary is presented in Table 8.

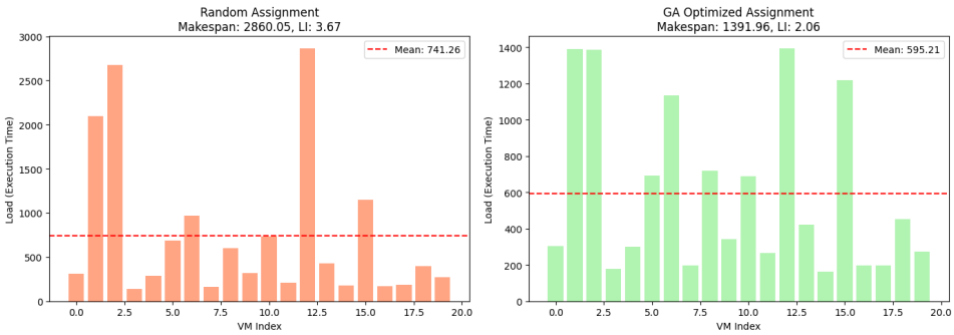

Fig. 4: VM Load Distribution Comparison Between Random Assignment and GA Optimized Assignment

As shown in Figure 4, the Random Assignment baseline produced a highly uneven load distribution across VMs, with some VMs receiving loads exceeding 2500 seconds while others carried less than 200 seconds, resulting in a makespan of 2860.05 and a load imbalance of 3.67 with a mean load of 741.26 seconds. In contrast, the GA-optimized solution produced a more balanced distribution with a mean load of 595.21 seconds, significantly reducing the peak load and narrowing the gap between the most and least loaded VMs, achieving a makespan of 1391.96 and a load imbalance of 2.06.

Table 8: Performance Improvement Summary

Metric	Random Assignment	GA Optimized	Improvement (%)
Makespan (seconds)	2860.0474	1391.9607	51.33%
Load Imbalance	3.6712	2.0630	43.81%

As presented in Table 8, the GA-optimized solution achieved a makespan of 1391.9607 seconds compared to 2860.0474 seconds under Random Assignment, representing an improvement of 51.33%. The degree of load imbalance was also reduced from 3.6712 to 2.0630, an improvement of 43.81%, confirming that the GA produced a considerably more balanced workload distribution across the 20 VMs compared to random task allocation.

3.5. Additional Metrics Evaluation

The evaluation was further extended using three additional metrics to provide a more comprehensive assessment of the scheduling performance. The summary of the three additional metrics is presented in Table 9, while the visualization of each metric is shown in Figure 5.

Table 9: Summary of Additional Evaluation Metrics

Metric	Random Assignment	GA Optimized	Improvement (%)
Load Distribution Efficiency (%)	21.41%	32.65%	11.24%
Avg Per-Task Completion Time (seconds)	378.06	278.37	26.37%
Computational Cost (USD)	$148.2521	$119.0420	19.70%

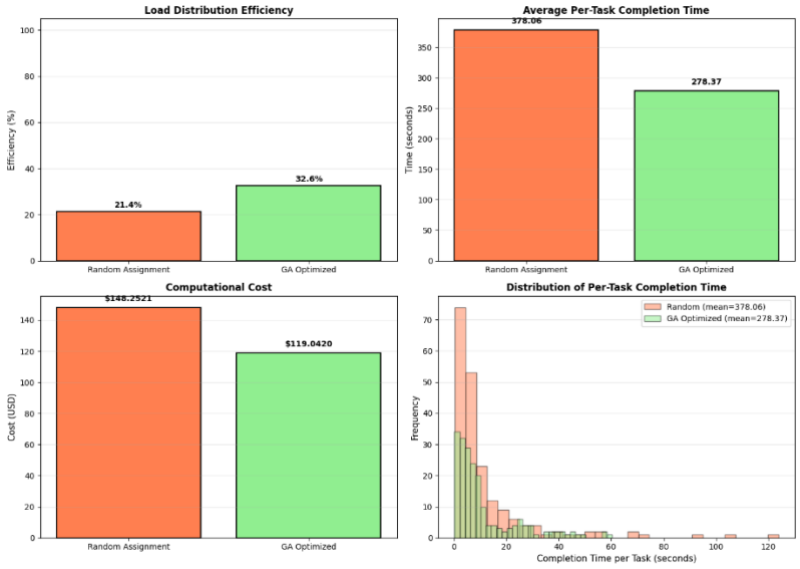

Fig. 5: Visualization of Additional Evaluation Metrics

As presented in Table 9 and Figure 5, the load distribution efficiency increased from 21.41% to 32.65%, an improvement of 11.24%. Although the absolute efficiency values are relatively low for both approaches, the improvement reflects the positive impact of GA optimization in reducing load imbalance. The relatively low efficiency values in both cases are attributable to the inherent heterogeneity of VM processing capacities, which naturally creates load variation even under an optimized scheduling solution.

The average per-task completion time was reduced from 378.06 seconds to 278.37 seconds, an improvement of 26.37%. As shown in the distribution histogram in Figure 4, the GA-optimized scheduling produced a distribution with a lower mean and a more concentrated spread of completion times, indicating that more tasks were completed faster under the optimized schedule compared to random assignment.

The computational cost was reduced from $148.2521 to $119.0420, a reduction of 19.70%, reflecting the lower total execution time achieved by the GA-optimized scheduling solution. This result demonstrates that optimizing task scheduling not only improves system performance but also leads to a direct reduction in operational costs.

4. Conclusion

This research demonstrated that the *Genetic Algorithm* is an effective approach for optimizing load balancing in cloud task scheduling, producing significant improvements across all evaluation metrics compared to the Random Assignment baseline. The GA achieved a makespan reduction of 51.33% and a load imbalance reduction of 43.81%, confirming that the evolutionary optimization process successfully distributed tasks more evenly across the available *Virtual Machine*s. Additional evaluation metrics further validated the superiority of the GA-optimized solution, with load distribution efficiency improving by 11.24%, average per-task completion time reduced by 26.37%, and computational cost reduced by 19.70%. These results highlight that optimizing task scheduling in cloud environments not only improves system performance but also leads to measurable reductions in operational costs.

For future research, the integration of the *Genetic Algorithm* with other metaheuristic methods such as Particle Swarm Optimization or Ant Colony Optimization may produce a more robust hybrid optimization approach. Incorporating additional objectives such as energy consumption and response time into the fitness function could further enhance the multi-objective scheduling capability. Additionally, testing on larger-scale datasets with a greater number of tasks and VMs would provide a more comprehensive evaluation of the scalability and practical applicability of the proposed approach in real-world cloud computing environments.

References

[1] H. Zhang, "A Cloud Computing Task Scheduling Method Based on *Genetic Algorithm*," in *Proceedings of the 2nd International Conference on Information Economy, Data Modeling and Cloud Computing, ICIDC 2023*, 2023. doi: 10.4108/eai.2-6-2023.2334608.

[2] M. Manavi, Y. Zhang, and G. Chen, "Resource Allocation in Cloud Computing Using *Genetic Algorithm* and Neural Network," in *2023 IEEE 8th International Conference on Smart Cloud (SmartCloud*, 2023, pp. 1–8.

[3] T. Hai *et al.*, "Task scheduling in cloud environment : optimization , security prioritization and processor selection schemes," *J. Cloud Comput.*, vol. 12, no. 15, pp. 1–12, 2023, doi: 10.1186/s13677-022-00374-7.

[4] L. Imene, S. Sihem, K. Okba, and B. Mohamed, "A third generation *Genetic Algorithm* NSGAIII for task scheduling in cloud computing," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 9, pp. 7515–7529, 2022, doi: 10.1016/j.jksuci.2022.03.017.

[5] M. Agarwal and S. Gupta, "An Adaptive *Genetic Algorithm*-Based Load Balancing-Aware Task Scheduling Technique for Cloud Computing," *Comput. Mater. Contin.*, vol. 73, no. 3, pp. 6103–6119, 2022, doi: 10.32604/cmc.2022.030778.

[6] M. I. Alghamdi, "Optimization of Load Balancing and Task Scheduling in Cloud Computing Environments Using Artificial Neural Networks-Based Binary Particle Swarm Optimization (BPSO)," *Sustainability*, vol. 14, no. 19, pp. 1–20, 2022, doi: 10.3390/su141911982.

[7] R. Gulbaz, A. B. Siddiqui, N. Anjum, A. A. Alotaibi, T. Althobaiti, and N. Ramzan, "Balancer *Genetic Algorithm* — A Novel Task Scheduling Optimization Approach in Cloud Computing," *Appl. Sci.*, vol. 11, no. 14, pp. 1–24, 2021, doi: 10.3390/app11146244.

[8] N. R. Sabat, R. R. Sahoo, B. Acharya, and R. Kumar, "Hybrid *Genetic Algorithm* and Water Wave Optimization Approach for QoS Aware Multi Objective Task Scheduling and Load Balancing in Cloud Environments," *EAI Endorsed Trans. Internet Things*, vol. 11, pp. 1–16, 2025, doi: 10.4108/eetiot.12172.

[9] A. Y. Hamed and M. H. Alkinani, "Task Scheduling Optimization in Cloud Computing Based on *Genetic Algorithm*s," *Comput. Mater. Contin.*, vol. 69, no. 3, pp. 3289–3301, 2021, doi: 10.32604/cmc.2021.018658.

[10] I. Naz, S. Naaz, P. Agarwal, B. Alankar, F. Siddiqui, and J. Ali, "A *Genetic Algorithm*-Based *Virtual Machine* Allocation Policy for Load Balancing Using Actual Asymmetric Workload Traces," *symmetry S Artic.*, vol. 15, no. 5, pp. 1–22, 2023, doi: 10.3390/sym15051025.

[11] L. Yin, J. Liu, F. Zhou, M. Gao, and M. Li, "Cost - based hierarchy *Genetic Algorithm* for service scheduling in robot cloud platform," *J. Cloud Comput.*, vol. 12, no. 35, pp. 1–16, 2023, doi: 10.1186/s13677-023-00395-w.

[12] Y. Setiawati *et al.*, "Penentuan Rute Optimal Wisata di Kota dan Kabupaten Madiun Menggunakan Algoritma Genetika," *J. Keilmuan dan Keislam.*, vol. 3, no. 1, pp. 49–56, 2024, doi: 10.23917/jkk.v3i1.223.

[13] K. F. G, E. Rosman, M. I. Nasution, Y. K. Febrina, and R. S. Hasibuan, "Penerapan Data Mining untuk Pemetaan Kinerja Akademik Mahasiswa dengan Metode K-Means," *J. Sci. Soc. Res.*, vol. 8, no. 2, pp. 3137–3143, 2025, doi: 10.54314/jssr.v8i2.3179.

[14] R. Naufal and M. S. Hasibuan, "Optimization of Distribution Routes Using the *Genetic Algorithm* in the Traveling Salesman Problem," *J. Appl. Informatics Comput.*, vol. 9, no. 1, pp. 211–220, 2025, doi: 10.30871/jaic.v9i1.8864.

[15] Y. Li, S. Wang, X. Hong, and Y. Li, "Multi-objective Task Scheduling Optimization in Cloud Computing based on *Genetic Algorithm* and Differential Evolution Algorithm," in *2018 37th Chinese Control Conference (CCC)*, Technical Committee on Control Theory, Chinese Association of Automation, 2018, pp. 4489–4494.