ESJFM: An Enhanced SJF Scheduling Model to Improve Resource Utilization In Heterogeneous Cloud Environments

Ali Mothana, Hikmat Al-Quhfa, and Jie Song*

***Abstract*—Cloud computing has become essential in today's IT industry by offering flexible, on-demand services like data storage and computing platforms. However, managing limited resources to meet growing user demand remains a challenge for cloud providers, particularly in maintaining Quality of Service (QoS). Task scheduling plays a crucial role in resource management, but traditional algorithms like Shortest Job First (SJF) can lead to resource imbalances and task starvation. This study introduces the Enhanced Shortest Job First Model (ESJFM), which organizes tasks by size and assigns them to Virtual Machines (VMs) with appropriate processing power—short tasks are directed to low-power VMs, while longer tasks are assigned to high-power VMs. Experiments conducted using CloudSim with GoCJ and random datasets show that ESJFM improves resource utilization by over 90%, reducing makespan by approximately 40% compared to SJF. The proposed model effectively eliminates starvation, balances the load, and enhances resource utilization in heterogeneous cloud environments.**

***Index Terms*—Cloud Computing; Task Scheduling; Shortest Job First (SJF); Resource Utilization; Load Balancing; CloudSim.**

I. Introduction

Cloud computing has become a key technology in the IT industry, offering services such as data storage, computing power, and software applications over the internet [1]. Instead of managing physical infrastructure, organizations can use these resources on-demand, making cloud computing cost-effective, flexible, and scalable [2]. However, managing the growing demand from users while maintaining Quality of Service (QoS) remains a significant challenge for cloud providers [3], [4]. One of the most important aspects of cloud computing is task scheduling, which determines how tasks are assigned to available resources called Virtual Machines (VMs) [5]. Proper scheduling helps ensure that resources are used efficiently, ensuring tasks are completed as quickly as possible [6]. However, scheduling in a cloud system is complex due to dynamic workloads and varying resource capabilities [7], [8]. Traditional scheduling methods like First-Come-First-Serve (FCFS), Round Robin (RR), and Shortest Job First (SJF) are commonly used [9]. Among these, SJF is favored because it prioritizes shorter tasks, minimizing waiting times for those tasks. However, in cloud environments, SJF has several critical drawbacks:

Ali Mothana, Hikmat Al-Quhfa and Jie Song are with the College of Software, Northeastern University, Shenyang, China (e-mail: 2328023@stu.neu.edu.cn; 2327005@stu.neu.edu.cn;).
*Corresponding author: Jie Song (e-mail: song.jie@mail.neu.edu.cn).

- **Starvation**, where longer tasks are delayed indefinitely because shorter tasks are always prioritized.
- **Load imbalance**, where powerful VMs are underutilized while others are overloaded, leading to inefficiencies.
- **Performance degradation**, especially in heterogeneous environments where resource capabilities vary significantly.

These issues are particularly important in real cloud environments, such as those represented by the Google Cloud Jobs (GoCJ) dataset, where workloads are highly dynamic and resources are diverse in terms of capacity and capabilities. For example, the GoCJ dataset illustrates real-world workloads, highlighting the inefficiencies caused by traditional scheduling methods in managing diverse cloud resources effectively. Many improvements to SJF have been proposed, such as Modified SJF (MSJF), which attempts to balance the workload across VMs, and Dynamic Heterogeneous SJF (DHSJF), which takes task CPU usage into account [10], [11]. While these models improve performance in certain scenarios, they still do not fully address the challenges of efficient resource usage, fairness, and managing dynamic workloads, especially when resources are heterogeneous [12]. To address these issues, this study introduces a new scheduling model: the Enhanced Shortest Job First Model (ESJFM). This model builds on the SJF algorithm but introduces a key feature: it classifies tasks based on their length and assigns them to virtual machines (VMs) with appropriate processing power. Short tasks are assigned to low-power VMs, while longer tasks are directed to high-power VMs. This approach helps achieve a better balance in task distribution, improving resource usage and reducing delays. The contributions of this study are:

- Proposing a novel Enhanced Shortest Job First Model (ESJFM) that addresses the limitations of traditional SJF in cloud computing environments.
- Introducing a dynamic task classification mechanism that ensures efficient task distribution based on task size and VM processing power.
- Demonstrating significant improvements in resource utilization and task completion times (i.e., makespan and response time) through simulations using the CloudSim platform with random and GoCJ datasets.
- Comparing the performance of ESJFM with traditional SJF and Modified SJF (MSJF) algorithms, showing a 40% reduction in makespan and over 90% improvement in resource utilization.

This paper provides a practical solution for improving task scheduling in cloud environments, leading to better resource management and higher QoS for users and cloud providers, by enhancing task allocation efficiency and minimizing delays in heterogeneous cloud systems.

II. Related work

Task scheduling in cloud computing remains one of the most critical research topics, as it directly impacts resource efficiency and Quality of Service (QoS), especially in heterogeneous cloud environments where resources and tasks vary in their characteristics. The traditional Shortest Job First (SJF) algorithm is simple and effective for minimizing average waiting time but is severely limited by task starvation and an inability to handle heterogeneous Virtual Machines (VMs) efficiently. This section provides a systematic classification and critical comparison of recent SJF extensions, focusing on their approaches to load balancing, starvation prevention, and managing resource heterogeneity.

A. SJF Enhancements Focused on Starvation Mitigation

Classic SJF is vulnerable to starvation, where long tasks may be delayed indefinitely by a continuous influx of short tasks. Addressing this fundamental flaw often involves modifying the task prioritization rule. [13] proposed the Improved Shortest Job First (IMPSJF) algorithm for general CPU scheduling. Their method enhances fairness by factoring in the average of remaining sorted burst times to boost the priority of older, longer processes. While IMPSJF effectively tackles the starvation problem, its scope is limited to general operating system scheduling and does not consider the specialized, highly heterogeneous resource landscape of a cloud environment (e.g., assigning tasks to VMs of varying power).

B. SJF Models for Load Balancing in Heterogeneous Clouds

To maximize throughput and resource utilization in cloud environments, many extensions focus on intelligently matching tasks to heterogeneous VMs to distribute the load effectively. This usually involves classifying tasks by length and VMs by processing power. The Modified SJF (MSJF) with Generalized Priority (GP) techniques, such as those presented in [14] and GP-MSJF in [15], are designed to improve load sharing and reduce makespan. These models classify tasks and VMs, often statically assigning long tasks to powerful VMs and short tasks to weaker ones. While this approach effectively leverages resource heterogeneity to balance the initial load, it typically lacks a dynamic mechanism to prevent the indefinite starvation of long tasks that may be preempted or delayed by subsequent arrivals. Similarly, [16] and [17] propose Enhanced SJF (ESJF) and ESJFP models, respectively, which use enhanced priority-based sorting to optimize resource allocation and makespan. However, their primary focus remains on efficient initial allocation rather than a guaranteed time-based starvation mitigation strategy. An alternative approach is the Enhanced SJF with VM Migration proposed by Anastasopoulos et al. [18]. Migration allows for dynamic load redistribution, but it introduces significant overhead due to the process of transferring VM states, which is generally undesirable for high-throughput cloud systems.

C. Hybrid and Advanced Optimization Approaches

For highly complex or multi-objective scheduling problems, researchers often turn to hybrid or metaheuristic algorithms. [19] proposed a comparative analysis that included a Hybrid SJF and Genetic Algorithm (SJF + GA) model. By combining the local optimization of SJF with the global search capabilities of GA, such models can achieve superior makespan and response time.

D. Critical Comparison and Novelty of ESJFM

The comparison clarifies the novelty of the ESJFM model. While many approaches handle heterogeneous resources or address starvation, few combine a lightweight, rule-based heterogeneous assignment with a guaranteed, dynamic starvation prevention mechanism:

- **Decoupled Starvation/Load Management:** Unlike simpler models (e.g., [14]), ESJFM implements a dual-layer strategy: a static classification for initial load balancing and a dynamic priority boosting mechanism ($T_{promote}$) that is separate and guaranteed to prevent starvation, a feature typically missing in MSJF-based heterogeneous models.
- **Adaptive Classification:** By using the average task length as a dynamic threshold, ESJFM's classification is more adaptive to mixed and changing workloads than models that rely on static thresholds or pre-defined resource classes.

This integrated approach allows ESJFM to simultaneously and robustly guarantee starvation avoidance and maximize resource utilization in heterogeneous cloud environments.

III. Materials and Methods

This section presents the tools, datasets, and methodology employed to develop and evaluate the Enhanced Shortest Job First Model (ESJFM) proposed in this study. The research primarily relies on simulation-based testing to assess the performance of the proposed algorithm in heterogeneous cloud computing environments.

A. Proposed Algorithm: Enhanced Shortest Job First Algorithm (ESJFM)

This research proposes the Enhanced Shortest Job First Algorithm (ESJFM), which improves upon the traditional Shortest Job First (SJF) algorithm by better handling heterogeneous resources and optimizing task allocation. The ESJFM is designed to minimize makespan, response time, and waiting time, while maximizing resource utilization. The algorithm addresses key challenges such as starvation and resource underutilization, ensuring that tasks are allocated to Virtual Machines (VMs) according to their processing power and the length of the tasks.

The ESJFM works as follows:

- **Sort Tasks:** All tasks (Cloudlets) are sorted in ascending order by size.
- **Classify Virtual Machines:** VMs are classified into two categories based on their processing capacity (MIPS):
  - **VM1 (Low Power):** VMs with lower processing power.
  - **VM2 (High Power):** VMs with higher processing power.
- **Task Allocation:** Tasks are assigned to VMs based on their length:
  - Tasks shorter than the average task length are assigned to VM1 (Low Power).
  - Tasks longer than the average task length are assigned to VM2 (High Power).

This classification ensures efficient resource utilization, balances the load between different VMs, and prevents starvation, where long tasks might otherwise be delayed indefinitely. To prevent starvation, a dynamic priority boosting mechanism is introduced. If a long task waits beyond a predefined threshold (*T_promote*), its priority is increased, and it is reassigned to a high-power VM for immediate execution. This ensures that all tasks, regardless of length, will eventually be executed.

By distributing tasks based on both the task length and the VM's processing power, the algorithm enhances overall resource utilization, reduces makespan, and improves system throughput, as shown in Figure 1.

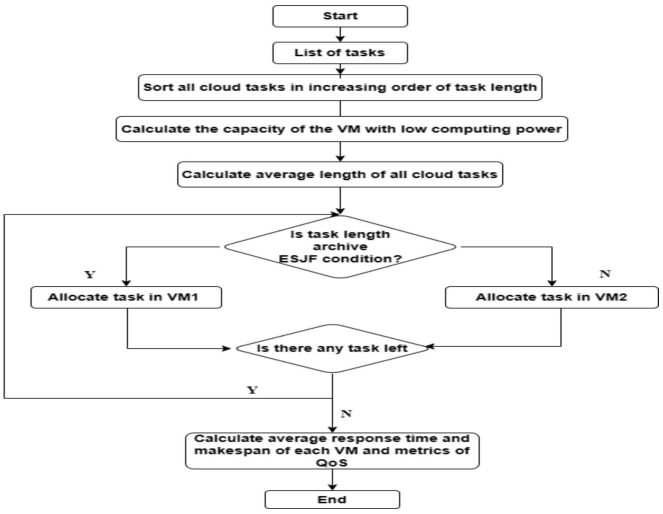

Fig. 1: Flowchart of The Proposed ESJFM.

B. Simulation Environment and Configuration

For this study, we utilized *CloudSim* (version 3.0.3), a Java-based simulation toolkit designed to model cloud computing environments. *CloudSim* enables the simulation of cloud components, including virtual data centers, hosts, and Virtual Machines (VMs), allowing for the testing of various scheduling algorithms without the need for physical hardware. The CloudSim is widely adopted in cloud computing research for its flexibility in simulating key cloud components, such as Cloudlets (tasks) and Datacenters, providing valuable insights into cloud system performance.

The experiments in this research were conducted using two datasets that represent typical cloud workloads:

- **Random Dataset:** A set of tasks with randomly generated sizes ranging from 10,000 to 160,000 Million Instructions (MIs). The experiments were conducted with 20, 40, and 60 tasks to simulate different cloud workloads.
  - **Generation Method**: The dataset was generated randomly, but the exact distribution type (uniform, normal, etc.) was not specified. For reproducibility, a random seed value should ideally be provided in future work.
  - **Task Number Distribution**: The dataset consists of 20, 40, and 60 tasks, but further clarification on how tasks are distributed across sizes would help.
- **Google Cloud Jobs (GoCJ) Dataset:** A dataset derived from real-world cloud workload traces collected from Google clusters. This dataset includes a range of cloud task sizes, from Small (15k MIs) to Huge (900k MIs), providing a more realistic representation of cloud tasks and resource demands. The percentage of jobs in each size category is shown in Figure 2.

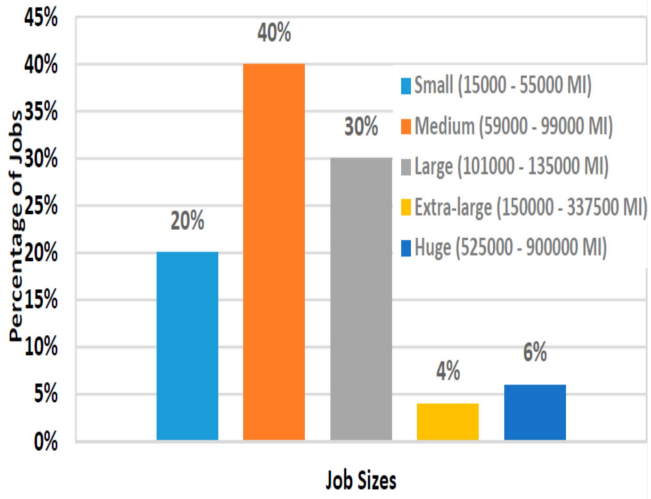

Fig. 2: The Composition of the GoCJ Dataset. The dataset is divided into job sizes ranging from Small (15k-55k MIs) to Huge (525k-900k MIs), but the total number of tasks in the dataset is not specified.

The experiments were conducted with the following Virtual Machine (VM) configurations to simulate a real-world cloud scenario with varying workloads and resource requirements. The simulation environment used two types of VMs, VM1 (Low Power) and VM2 (High Power), which differ in their processing power (MIPS), memory (RAM), and storage.

- **VM1 (Low Power)**: 1,000 MIPS processing power, 512 MB RAM.
- **VM2 (High Power)**: 8,000 MIPS processing power, 512 MB RAM.

The data center included one host and two VMs with different processing capacities. This setup allows the simulation of a real-world cloud environment with tasks distributed across VMs of varying capabilities.

TABLE I: Virtual Machine Configuration for the Simulation.

VM Type	MIPS	RAM (MB)	Storage (MB)
VM1 (Low Power)	1,000	512	10,000
VM2 (High Power)	8,000	512	10,000

Additionally, the host configuration for the data center is as follows:

TABLE II: Configuration of Host

MIPS	Memory	Bandwidth	Storage
10,000	16,384 MB	10,000 MB	1,000,000 MB

The simulation uses these configurations to evaluate the performance of the Enhanced Shortest Job First Algorithm (ESJFM) and compare it to other scheduling algorithms in terms of key performance metrics.

C. ESJFM pseudo-code

The step-by-step description and pseudo-code for the ESJFM are provided in Algorithm 1.

Algorithm 1 Enhanced Shortest Job First Algorithm (ESJFM)

Input CloudletList, VMList
Output SortedList, SortedVM, QoS metrics

totalCloudlet ← cloudletList.size(), totalVM ← vmList.size()
avgCloudlet ← $\frac{\sum cloudletLength}{totalCloudlet}$, avgVM ← $\frac{\sum vmLength}{totalVM}$
for j = 1 to totalVM **do**
  if vmList.get(j).Length < avgVM **then**
    VM[1].add(vmList.get(j)) ▷ Low power VM
  else
    VM[2].add(vmList.get(j)) ▷ High power VM
  end if
end for
VM[1].capacity ← $\frac{totalCloudlet \times totalVM \times VM[1].Length}{VM[2].Length}$
cloudletList.sort() ▷ Sort tasks by length
for i = 1 to totalCloudlet **do**
  if cloudletLength[i] < avgCloudlet&VM[1].Length < VM[1].capacity **then**
    Assign cloudletList[i] to VM[1]
  else
    Assign cloudletList[i] to VM[2]
  end if
  if cloudletWaitTime[i] > $T_{promote}$ **then**
    Increase cloudletPriority[i] and assign to VM[2]
  end if
end for
Calculate QoS metrics: Makespan, Response Time, Waiting Time, Throughput, Resource Utilization
End

D. Performance Metrics

The scheduling of tasks over cloud environment resources is a complex problem, influenced by various metrics such as makespan, response time, waiting time, resource utilization, and throughput. These metrics play a crucial role in evaluating the performance of task scheduling algorithms. However, not all algorithms can meet all these metrics due to various factors such as the execution environment, task size, and resource provisioning. Cloud service providers and users have different objectives—while users are concerned with the efficiency of task execution, cloud providers aim to optimize resource utilization.

The optimization criteria for task scheduling in a cloud environment, as applied to the proposed algorithm, are as follows:

1) Makespan: Makespan refers to the total time required to complete all tasks executed by the available VMs [20]. It is defined as the time taken to complete the last task, and task scheduling should aim to minimize the makespan. The makespan is calculated as:

$$\textbf{Makespan} = CT_n \quad (1)$$

Where CT_n represents the completion time of the last task in the system.

The average makespan (Avg. Makespan) for all VMs is computed as:

$$\textbf{Avg.Makespan} = \frac{\sum_{j=1}^{m} Makespan_j}{m} \quad (2)$$

Where:

- m represents the number of VMs in use.
- $Makespan_j$ is the makespan for the j-th VM.

2) Response Time (RT): Response time is the time taken by the system to start executing a task after it has been requested, and minimizing this time is important for improved performance [21]. It is calculated as the difference between the completion time and the starting time of the task:

$$RT_i = CT_i - ST_i \quad (3)$$

Where:

- CT_i is the completion time of task i.
- ST_i is the starting time of task i.

The average response time (Avg.RT) is then calculated as:

$$\text{Avg.RT} = \frac{\sum_{i=1}^{n} RT_i}{n} \quad (4)$$

Where:

- n is the total number of tasks.

For all VMs, the mean of total average response time (M.Avg.RT) is:

$$\sum_{m} \text{Avg.RT} \quad \text{M.}$$

$$\text{Avg.RT} = \frac{\sum_{j=1}^{} j}{m} \quad (5)$$

Where:

- $Avg.RT_j$ is the average response time for the j-th VM.
- m is the number of VMs in use.

3) Waiting Time (WT): Waiting time refers to the total amount of time a task spends in the ready queue before execution [22]. It can be calculated as the turnaround time minus the task length, as shown below:

$$WT_i = TT_i - TL_i \quad (6)$$

Where:

- TT_i is the turnaround time for task i.
- TL_i is the task length for task i.

The average waiting time (AWT) is calculated as:

$$AWT = \frac{\sum_{i=1}^{n} WT_i}{n} \quad (7)$$

Where:

- n is the total number of tasks.

4) Resource Utilization (RU): Resource utilization is a critical metric, as it measures the ability of the system to allocate resources effectively without causing issues or wasting capacity [23]. Maximizing resource utilization is crucial for cloud providers. It can be calculated as:

$$RU = \frac{\sum_{j=1}^{m} Makespan_j}{m \times Max.Makespan} \quad (8)$$

Where:

- Max.Makespan represents the maximum makespan value for all tasks across all VMs, i.e., the largest makespan observed for any VM during the scheduling process.

5) Throughput (TH): Throughput refers to the number of tasks completed within a unit of time [24]. It is an important metric for evaluating the efficiency of a scheduling algorithm. The throughput is calculated as:

$$TH = \frac{n}{Makespan} \quad (9)$$

Where:

- n is the total number of tasks completed.
- Makespan is the time taken to complete all tasks.

IV. Results and Discussion

The results of implementing the Enhanced Shortest Job First Algorithm (ESJFM) were compared with those obtained from the Modified SJF (MSJF) and SJF algorithms. As explained in the previous section, in ESJFM, tasks are categorized based on their lengths—those shorter than the average are assigned to low-power VMs, and those longer than the average are assigned to high-power VMs. This ensures that longer tasks do not have to wait excessively and are executed on more powerful VMs, while shorter tasks are allocated to less powerful machines, optimizing the resource allocation process.

A. Experimental Setup

For clarity, tasks in CloudSim are referred to as cloudlets. The experiments were conducted by creating one datacenter, one host, and two virtual machines (VMs) with different processing capacities. The dataset was simulated using two different sets of tasks: 20, 40, and 60 tasks, respectively. These tasks were assigned to the VMs following the ESJFM scheduling algorithm. The tasks were executed by the allocated VMs, and the execution results (including makespan, response time, and resource utilization) were obtained from the datacenter broker. The simulation was repeated with different datasets to validate and generalize the results and compare the performance of ESJFM with SJF and MSJF.

B. Task Allocation to VMs

Table III presents the task allocation to VMs during a simulation with 20 tasks using the GoCJ dataset. Each task is allocated to a specific VM, showing the start times, execution times, and completion times for each task.

TABLE III: Task Allocation to Virtual Machines

VM ID	Task No.	Start Time	Execution Time	Makespan
2	Task 4	0.1	5.88	5.97
2	Task 5	5.97	6.12	12.1
1	Task 0	0.1	15	15.1
2	Task 6	12.1	6.38	18.48
2	Task 7	18.48	6.62	25.1
2	Task 8	25.1	6.88	31.98
2	Task 9	31.98	7.38	39.35
1	Task 1	15.1	27.5	42.6
2	Task 10	39.35	7.62	46.98
2	Task 11	46.98	7.88	54.85
2	Task 12	54.85	8.12	62.98
2	Task 13	62.98	8.37	71.35
2	Task 14	71.35	8.88	80.22
1	Task 2	42.6	40	82.6
2	Task 15	80.22	9.12	89.35
2	Task 16	89.35	9.38	98.72
2	Task 17	98.72	9.62	108.35
2	Task 18	108.35	9.88	118.22
1	Task 3	82.6	45	127.6
2	Task 19	118.22	10.12	128.35

C. Makespan Comparison

Table IV summarizes the makespan results for different numbers of tasks using the ESJFM, MSJF, and SJF algorithms. The table clearly shows that ESJFM achieves a significantly lower makespan in all experiments, with the GoCJ dataset showing the best performance.

TABLE IV: Makespan of ESJFM, MSJF and SJF Algorithms

DataSet	Task No.	ESJFM	MSJF	SJF
Random	Experiment 1 (20 Tasks)	73.60	97.50	312.10
	Experiment 2 (40 Tasks)	225.40	297.50	960.60
	Experiment 3 (60 Tasks)	442.84	580.00	1,939.10
GoCJ	Experiment 1 (20 Tasks)	128.35	154.60	509.10
	Experiment 2 (40 Tasks)	265.10	337.60	1,053.60
	Experiment 3 (60 Tasks)	372.60	492.10	1,562.60

As shown in Table V, ESJFM consistently outperforms MSJF and SJF across all datasets and task numbers, achieving the shortest makespan.

TABLE V: Average Makespan of ESJFM, MSJF, and SJF Algorithms

DataSet	Task No.	ESJFM	MSJF	SJF
Random	Experiment 1 (20 Tasks)	72.50	83.92	177.81
	Experiment 2 (40 Tasks)	222.94	254.48	544.59
	Experiment 3 (60 Tasks)	439.17	502.39	1,097.00
GoCJ	Experiment 1 (20 Tasks)	127.98	133.54	288.63
	Experiment 2 (40 Tasks)	247.73	279.45	592.70
	Experiment 3 (60 Tasks)	360.60	412.88	881.23

D. Response Time (RT)

The average response time for each algorithm is summarized in Table VI. The results show that ESJFM achieves a significantly lower response time compared to MSJF and SJF, with the best performance observed for both the Random and GoCJ datasets.

TABLE VI: Mean of Total Average Response Time of ESJFM, MSJF and SJF Algorithms

DataSet	Task No.	ESJFM	MSJF	SJF
Random	Experiment 1 (20 Tasks)	37.05	43.64	80.97
	Experiment 2 (40 Tasks)	99.82	115.08	222.60
	Experiment 3 (60 Tasks)	184.99	213.45	426.10
GoCJ	Experiment 1 (20 Tasks)	64.50	68.81	131.34
	Experiment 2 (40 Tasks)	110.57	128.29	258.39
	Experiment 3 (60 Tasks)	157.15	185.48	374.91

E. Resource Utilization (RU)

Table VII compares resource utilization for ESJFM, MSJF, and SJF. The results show that ESJFM provides the highest resource utilization in all experiments, confirming its superiority in maximizing the efficiency of available resources.

TABLE VII: Resource utilization of ESJFM, MSJF and SJF Algorithms

DataSet	Task No.	ESJFM	MSJF	SJF
Random	Experiment 1 (20 Tasks)	0.9851	0.8607	0.5697
	Experiment 2 (40 Tasks)	0.9891	0.8554	0.5669
	Experiment 3 (60 Tasks)	0.9917	0.8662	0.5657
GoCJ	Experiment 1 (20 Tasks)	0.9971	0.8637	0.5669
	Experiment 2 (40 Tasks)	0.9345	0.8277	0.5625
	Experiment 3 (60 Tasks)	0.9678	0.8390	0.5639

V. Discussion

This research introduces the Enhanced Shortest Job First Model (ESJFM), an advanced task scheduling algorithm designed to enhance the performance of cloud computing systems by addressing the limitations of traditional scheduling models like SJF and MSJF. The algorithm optimizes critical Quality of Service (QoS) metrics such as makespan, response time, and resource utilization, providing a more efficient solution for heterogeneous cloud environments.

The core challenge in cloud computing is the dynamic nature of workloads and the diversity of resources, which often leads to inefficient resource allocation and load imbalances. Traditional algorithms like SJF and its variants, such as MSJF, address some of these issues but still struggle with problems like starvation and underutilization of high-power Virtual Machines (VMs). ESJFM solves these problems by dynamically allocating tasks based on the VM power and task length, ensuring optimal task distribution and improving overall cloud performance.

A. Key Findings and Contribution

The key contributions of this study are summarized as follows:

- **Improved Makespan**: ESJFM significantly reduces makespan compared to SJF and MSJF by efficiently classifying tasks and allocating them to VMs based on processing power. This dynamic allocation minimizes delays, ensuring all tasks are completed faster.
- **Reduced Response Time**: ESJFM demonstrated superior performance in reducing response time, particularly on low-power VMs, as shorter tasks are quickly assigned to available VMs, improving system efficiency.
- **Higher Resource Utilization**: By balancing the load between VMs more effectively, ESJFM maximizes resource utilization. Both low-power and high-power VMs are kept busy, reducing idle time and increasing system throughput.
- **Balanced Load Distribution**: ESJFM's dynamic task allocation ensures that no single VM is overloaded, preventing bottlenecks and ensuring fairness in resource distribution across the system.

These findings show that ESJFM not only improves performance but also addresses the major challenges faced by traditional cloud scheduling algorithms, particularly in heterogeneous environments.

B. Broader Implications and Future Directions

The implications of this work are significant for the future of cloud task scheduling. As cloud environments continue to grow and evolve, scheduling algorithms must adapt to handle dynamic workloads efficiently. ESJFM presents a promising solution, capable of scaling with the increasing complexity of cloud systems.

Future research can explore the following avenues:

- **Integration with Fog and Edge Computing**: The principles of ESJFM can be extended to fog and edge computing systems, where decentralized computing is required. Scheduling algorithms such as as as as ESJFM can optimize resource allocation in these systems, ensuring efficient data processing closer to the source.
- **Incorporating Additional Constraints**: Future work could incorporate cost and deadline constraints into ESJFM, making it even more applicable to real-world cloud computing scenarios, where time and cost are critical factors.

- **Hybrid Approaches**: Combining ESJFM with other scheduling approaches, such as priority-based or genetic algorithms, could offer more flexibility and improved performance in handling diverse cloud workloads.

VI. Conclusion

In conclusion, ESJFM represents a significant advancement in cloud task scheduling. By dynamically adapting task allocation based on task length and VM processing power, ESJFM improves makespan, reduces response time, and maximizes resource utilization. Our simulation experiments show that ESJFM outperforms traditional algorithms like SJF and MSJF, offering a more scalable and efficient solution for modern heterogeneous cloud environments. This work contributes to the field by providing a robust, adaptive scheduling model that addresses critical issues such as load balancing, resource underutilization, and task starvation in cloud systems.

References

[1] J. Gonza´lez-San-Mart´ın, L. Cruz-Reyes, C. Go´mez-Santilla´n, H. Fraire-Huacuja, N. Rangel-Valdez, B. Dorronsoro, and M. Quiroz-Castellanos, "A comprehensive review of the task scheduling problem in cloud computing: Recent advances and comparative analysis," p. 299-313, 2024.

[2] Y. Zhang, H. Zhang, and C. Song, "A hybrid algorithm for multi-objective task scheduling in heterogeneous cloud computing," *Journal of Supercomputing*, vol. 81, 2025.

[3] W. K. Awad, K. A. Z. Ariffin, M. Z. A. Nazri, and E. T. Yassen, "Resource allocation strategies and task scheduling algorithms for cloud computing: A systematic literature review," *Journal of Intelligent Systems*, vol. 34, no. 1, p. 20240441, 2025. [Online]. Available: https://doi.org/10.1515/jisys-2024-0441

[4] P. Tamilarasu and G. Singaravel, "Quality of service aware improved coati optimization algorithm for efficient task scheduling in cloud computing environment," *Journal of Engineering Research*, vol. 12, no. 4, pp. 768–780, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2307187723002493

[5] I. Sharma, R. Gupta, and P. Singh, "Task scheduling in cloud using multi-objective hybrid approach," *Cluster Computing*, vol. 28, 2025.

[6] G. Chen, J. Qi, Y. Sun, X. Hu, Z. Dong, and Y. Sun, "A collaborative scheduling method for cloud computing heterogeneous workflows based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 141, pp. 284–297, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X22004034

[7] N. Chauhan, N. Kaur, K. S. Saini, S. Verma, A. Alabdulatif, R. A. Khurma, M. Garcia-Arenas, and P. A. Castillo, "A systematic literature review on task allocation and performance management techniques in cloud data center," 2024, preprint arXiv 2402.13135.

[8] I. Behera and S. Sobhanayak, "Task scheduling optimization in heterogeneous cloud computing environments: A hybrid ga-gwo approach," *Journal of Parallel and Distributed Computing*, vol. 183, p. 104766, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731523001363

[9] R. Stan, L. Ba˘jenaru, C. Negru, and F. Pop, "Evaluation of task scheduling algorithms in heterogeneous computing environments," *Sensors*, vol. 21, no. 17, p. 5906, 2021.

[10] S. Seth and N. Singh, "Dynamic heterogeneous shortest job first (dhsjf): a task scheduling approach for heterogeneous cloud computing systems," *International Journal of Information Technology*, vol. 11, no. 4, pp. 653–657, 2018. [Online]. Available: file:///mnt/data/a3c4cf7d-cb2d-49c2-a48e-0be2f248d20b.pdf

[11] G. Pasare, A. Gade, and R. Bhat, "Enhance dynamic heterogeneous shortest job first (dhsjf): A task scheduling approach for heterogeneous cloud computing systems," *International Research Journal of Engineering and Technology (IRJET)*, vol. 6, no. 4, pp. 7577–7583, 2019, impact Factor value: 7.211 — ISO 9001:2008 Certified Journal. [Online]. Available: file:///mnt/data/a3c4cf7d-cb2d-49c2-a48e-0be2f248d20b.png

[12] I. Behera and S. Sobhanayak, "Htsa: A novel hybrid task scheduling algorithm for heterogeneous cloud computing environment," *Simulation Modelling Practice and Theory*, vol. 137, p. 103014, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X2400128X

[13] A. E. Evwiekpaefe, A. Ibrahim, and M. N. Musa, "Improved shortest job first cpu scheduling algorithm," *Dutse Journal of Pure and Applied Sciences (DUJOPAS)*, vol. 8, pp. 114–127, 2022.

[14] A. Pandit *et al.*, "Resources load sharing using rr, fcfs, sjf, msjf, gp algorithms in cloud computing," *International Journal of Cloud Computing*, vol. 13, pp. 78–90, 2022.

[15] N. K. Gupta, A. Walia, and A. Sharma, "GP-MSJF: An improved load balancing generalized priority-based modified SJF scheduling in cloud computing," in *Innovations in Computer Science and Engineering*, ser. Lecture Notes in Networks and Systems, V. Goar *et al.*, Eds. Springer, Singapore, 2022, vol. 436.

[16] Y. Pachipala, K. S. Sureddy, A. B. S. S. Kaitepalli, N. Pagadala, S. S. Nalabothu, and M. Iniganti, "Optimizing task scheduling in cloud computing: An enhanced shortest job first algorithm," *Procedia Computer Science*, vol. 233, pp. 604–613, 2024.

[17] J. Laha, S. Pattnaik, K. S. Chaudhury, and G. Palai, "Reducing makespan and enhancing resource usage in cloud computing with ESJFP method: A new dynamic approach," *Internet Technology Letters*, 2024, accepted: 26 October 2024.

[18] D. Anastasopoulos *et al.*, "Enhanced shortest job first algorithm with vm migration for cloud task scheduling," *Future Generation Computer Systems*, vol. 127, pp. 93–105, 2022.

[19] P. S. Reddy, S. Bhaskaran, K. Trisha, and N. Sampath, "Enhanced task scheduling in cloud computing: A comparative analysis and hybrid algorithm implementation using cloudsim," in *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, 2024.

[20] S. Gupta, S. Iyer, G. Agarwal, P. Manoharan, A. D. Algarni, G. Aldehim, and K. Raahemifar, "Efficient prioritization and processor selection schemes for heft algorithm: A makespan optimizer for task scheduling in cloud environment," *Electronics*, vol. 11, no. 16, p. 2557, 2022.

[21] M. Maqsood, S. A. Lashari, M. A. Saare, S. A. Sari, Y. M. Hussein, and H. O. Hatem, "Minimization response time task scheduling algorithm," in *IOP Conference Series: Materials Science and Engineering*, vol. 705, no. 1. IOP Publishing, 2019, p. 012008.

[22] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems*, vol. 91, pp. 407–415, 2019.

[23] S. A. Hamad and F. A. Omara, "Genetic-based task scheduling algorithm in cloud computing environment," *Int. J. Adv. Comput. Sci. Appl*, vol. 7, no. 4, pp. 550–556, 2016.

[24] A. V. Lakra and D. K. Yadav, "Multi-objective tasks scheduling algorithm for cloud computing throughput optimization," *Procedia Computer Science*, vol. 48, pp. 107–113, 2015.