iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.83341

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

Network Aware Priority Based Task Scheduling Optimization in Cloud Computing

Kunal Tiwari[1], Prashanth K[2]
Department of MCA, RV College of Engineering, Bengaluru, Karnataka, India

Abstract: ***Cloud computing offers scalable, flexible computational resources for today's distributed systems, kind of like it can stretch or shrink as needed, not always but usually. In practice, task scheduling really has to be handled well because it affects resource utilization, it lowers the waiting period, and it can improve overall cloud performance. Most older scheduling algorithms tend to emphasize execution time only, while they ignore communication latency, and also the task's priority which matters a lot sometimes. This paper proposes a lightweight simulation-based scheduling framework called NetworkAwareScheduler that improves Virtual Machine allocation using network-aware and priority-based optimization techniques. The proposed scheduler evaluates completion time, network latency, and task priority during task allocation. The system was implemented using Java and tested under different scheduling scenarios including low-latency, high-latency, and mixed-latency environments. Experimental results demonstrated significant improvements in throughput, waiting time reduction, makespan optimization, VM workload balancing, and communication-aware scheduling efficiency compared to traditional scheduling approaches.***

Index Terms: ***cloud computing, task scheduling, network-aware scheduling, Virtual Machine allocation, priority-based scheduling, distributed systems, scheduling optimization, Java simulation, throughput optimization, resource utilization.***

I. INTRODUCTION

Cloud computing has emerged as a transformative computing paradigm that enables on-demand access to shared computational resources, storage systems, and software services over the internet. Cloud infrastructure providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform offer scalable Virtual Machine environments that can dynamically allocate resources based on user demand. These distributed cloud environments continuously process thousands of user tasks across heterogeneous Virtual Machines, making efficient task scheduling one of the most critical challenges in cloud computing research [1]. Improper allocation of tasks in cloud environments may lead to several performance degradation issues including overloaded Virtual Machines, increased waiting time, communication bottlenecks, reduced throughput, unbalanced workload distribution, and poor resource utilization [4]. As modern cloud infrastructures become more geographically distributed and network-dependent, scheduling algorithms must consider not only execution efficiency but also critical communication-related parameters such as network latency, bandwidth utilization, and task priority [5]. Traditional cloud scheduling algorithms such as First Come First Serve (FCFS), Round Robin, Min-Min, and Max-Min were primarily designed for single-processor environments and mainly focus on processor allocation and execution order without considering communication overhead and workload balancing [6]. These algorithms often fail to optimize scheduling performance under dynamic distributed cloud conditions where network latency significantly impacts execution efficiency. Recent research in cloud scheduling has introduced optimization-based approaches such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO) to improve scheduling efficiency and resource utilization [4]. Although these techniques improve task allocation in several scenarios, many optimization algorithms involve higher computational complexity and implementation overhead that makes them unsuitable for lightweight simulation environments. This paper presents the **NetworkAwareScheduler**, a lightweight simulation-based scheduling framework designed to improve task allocation efficiency using network-aware and priority-based optimization techniques. The proposed scheduler sort of dynamically evaluates Virtual Machines with a weighted cost function, it looks at completion time, network latency and also the task priority, in general. The main contributions of the work are, as follows (though i might be oversimplifying a bit. A lightweight cloud scheduling simulation framework for distributed cloud environments.

A network-aware scheduling algorithm that factors in communication latency while doing the VM allocation.

Priority based task scheduling using a weighted cost-function optimization, more or less.

A comparative performance analysis between the HLGO Scheduler and the proposed NetworkAwareScheduler.

CSV based result generation for scheduling analysis and performance evaluation.

II. RELATED WORK

Cloud computing task scheduling has been pretty much researched a lot over the last two decades, because scheduling efficiency affects resource utilization directly, and also throughput, response time execution delay, and the general system performance in distributed cloud infrastructures. Nowadays cloud setups are made of mixed, heterogeneous Virtual Machines, distributed storage systems, and communication networks, where tasks they constantly compete for compute power and bandwidth and all that.

Classic scheduling methods like FCFS, Round Robin, Min-Min, and Max-Min were first introduced to improve processor allocation, and better execution efficiency. For example FCFS scheduling runs tasks based on arrival order, but it does not really account for the task length, the priority, or even the communication delay. Even if FCFS is easy to implement, those long tasks can end up blocking smaller ones for a rather long period, which then increases waiting time, and it tends to lower the throughput quite noticeably. Similarly, Round Robin scheduling improves fairness by assigning processor time in fixed intervals but introduces higher context-switching overhead and ignores communication-related optimization [9].

Heuristic scheduling approaches such as Min-Min and Max-Min algorithms were later introduced to improve scheduling efficiency. The Min-Min algorithm allocates smaller tasks first in order to reduce overall completion time, whereas Max-Min scheduling prioritizes larger tasks to reduce starvation of computationally intensive workloads. Although these algorithms improve execution efficiency under certain conditions, they still fail to consider communication latency and network-aware optimization during Virtual Machine allocation [13].

Several researchers proposed optimization-based scheduling techniques including Genetic Algorithms, PSO, and ACO to improve cloud scheduling performance. Genetic Algorithms apply evolutionary optimization strategies involving selection, mutation, and crossover operations. PSO scheduling uses swarm intelligence concepts inspired by bird flocking behavior, while ACO scheduling uses pheromone-based optimization techniques inspired by ant colony systems [12]. These approaches improve scheduling optimization capability but involve higher computational complexity and implementation overhead.

As cloud infrastructures evolved into geographically distributed systems, communication-aware scheduling became increasingly important. Network latency and communication delay significantly affect overall execution efficiency because tasks may spend considerable additional time transferring data between distributed components. Verma and Kaushal [16] demonstrated that cost-time efficient scheduling approaches that combine execution optimization with communication awareness improve overall cloud workflow performance. Recent studies increasingly focus on network-aware scheduling approaches that consider communication delay during task allocation, attempting to reduce execution overhead by selecting Virtual Machines with lower communication latency and balanced workload conditions.

The proposed project follows a similar research direction by introducing a lightweight simulation-based scheduling framework. Unlike traditional CPU-focused scheduling algorithms, the proposed NetworkAwareScheduler integrates network-aware optimization directly into the scheduling cost function, dynamically evaluating Virtual Machines using completion time, communication latency, task priority, and VM workload during scheduling decisions.

III. DATASET DESCRIPTION

The proposed project was developed as a lightweight simulation-based cloud scheduling framework designed to evaluate the effectiveness of network-aware and priority-based task scheduling optimization. Since the implementation primarily focuses on scheduling experimentation and performance analysis, the system uses dynamically generated simulation scenarios instead of real cloud deployment infrastructure. The simulation environment consists of multiple user tasks and Virtual Machines generated dynamically using predefined workload configurations. Each task simulates a cloud workload submitted by users in distributed computing environments, containing multiple scheduling-related parameters. Similarly, Virtual Machines are initialized with different computational capabilities and communication characteristics to simulate heterogeneous cloud infrastructures.

TABLE I Task Dataset Parameters

Parameter	Description
Task ID	Unique identifier for each task
Task Length	Computational size of task (in MI)
Priority	Importance level assigned to task
Network Latency	Communication delay associated with task

TABLE II Virtual Machine Parameters

Parameter	Description
VM ID	Unique identifier for VM
MIPS	Processing capability of VM
Current Load	Existing workload on VM
Network Latency	Communication delay of VM

The simulation framework supports multiple workload environments including Low-Latency Environment, High-Latency Environment, Mixed-Latency Environment, and Priority-Heavy Workload Environment. These workload conditions was created to evaluate scheduler performance under different communication and execution scenario. The generated task data set covers a broad range of task length and priority distributions, to simulate a more realistic cloud workload patterns.

The simulation starts with generating tasks and Virtual Machines dynamically through the Scenario Generator module. Generated tasks then are handed over to allocation scheduling algorithms, for use. The project compares two scheduling styles: the HLGO Scheduler as the baseline, and the proposed NetworkAwareScheduler in contrast.

IV. SYSTEM ARCHITECTURE

The proposed project follows a modular cloud scheduling architecture, which is kinda designed to simulate distributed task allocation in cloud computing environments. It mainly tries to boost scheduling efficiency through network-aware and priority based Virtual Machine allocation methods, and not only that. The whole framework is made of several interconnected pieces that handle task generation, Virtual Machine management, scheduling execution, performance evaluation and then result export, more or less.

This system simulates a cloud infrastructure where tasks are generated on the fly and the Virtual Machines are heterogeneous. The architecture was made intentionally as a simulation-based setup rather than a real cloud deployment, because simulators give you easier experimentation, flexible workload adjustment, lighter execution, and also less annoying debugging. You can change things without worrying about deployment frictions, right.

The scheduling flow starts with the Scenario Generator module, which dynamically creates both tasks and Virtual Machines using predefined configurations. After that, the generated tasks and Virtual Machines are forwarded into the scheduling engine, where you have both the HLGO Scheduler and the NetworkAwareScheduler. The HLGO Scheduler serves as a baseline scheduling mechanism for comparison purposes. Meanwhile the proposed NetworkAwareScheduler does communication-aware task allocation with dynamic scheduling cost optimization, and it’s designed around that idea.

In the end, the scheduler checks every Virtual Machine for each task and computes an execution cost. That cost is calculated using completion time, communication latency, and task priority, so basically it tries to balance speed, interaction delay, and importance levels all together. The Virtual Machine with the minimum, scheduling cost is picked for running. After the task split , the workload of the chosen VM is updated on the fly, so that the next scheduling call can take the new workload state into account and keep an even distribution of jobs across the available Virtual Machines. After the scheduling execution wraps up, the Metric Computation part figures out key cloud scheduling measurements and then the Result Export module produces CSV outputs.

V. PROPOSED SCHEDULING ALGORITHM

The NetworkAwareScheduler tries to make task distribution more efficient in distributed cloud setups, mainly by using network-aware plus priority oriented optimization bits. It kinda differs from the usual scheduling algorithms, which often just look at execution sequence and CPU share. In contrast, this proposed scheduler dynamically checks the communication delay , the task priority, the completion time, and the Virtual Machine workload, before it actually assigns anything to run

The main goal of the scheduling algorithm is to boost overall scheduling efficiency while also lowering communication overhead, the time people end up waiting, and workload imbalance across the cloud infrastructure. The implementation was done in the Java programming language, and it was created as a lightweight modular simulation framework, which is useful for academic experiments , and also for performance evaluation

The scheduling workflow starts after tasks and Virtual Machines are generated. First, the tasks are arranged in descending priority order so that higher-priority tasks get earlier scheduling opportunities, and they are not pushed back too long. The scheduler then evaluates all available Virtual Machines for each task individually, calculating the following parameters:

$$Execution\ Time = Task\ Length / VM\ MIPS$$

$$Completion\ Time = Current\ VM\ Load + Execution\ Time$$

$$Total\ Latency = Task\ Latency + VM\ Latency$$

The final scheduling decision is made using the weighted scheduling cost function that assigns different importance weights to each parameter based on its impact on overall scheduling performance:

$$Cost = 0.60 \times CT + 0.25 \times NL - 0.15 \times Priority$$

The cost function gives highest importance to completion time (weight = 0.60) because execution efficiency directly affects overall scheduling performance. Network latency receives moderate weight (0.25) because communication delay significantly influences distributed cloud systems. Task priority contributes negatively (−0.15) to the final cost value so that higher-priority tasks naturally receive scheduling preference over lower-priority tasks. The VM with minimum calculated scheduling cost is selected for execution.

TABLE III Proposed Scheduling Parameters

Parameter	Weight	Description
Completion Time	0.60	Estimated VM finishing time
Network Latency	0.25	Communication delay between task and VM
Task Priority	−0.15	Importance level (higher = preferred)
Scheduling Cost	—	Final VM selection value (minimize)

The proposed scheduling process follows a well-defined ten-step sequence: (1) generate tasks and Virtual Machines; (2) sort tasks by descending priority; (3) evaluate all available Virtual Machines; (4) calculate execution time; (5) compute completion time; (6) compute communication latency; (7) calculate scheduling cost for each VM; (8) select VM with minimum cost; (9) allocate task to selected VM; (10) update VM workload dynamically and recompute metrics after execution.

VI. IMPLEMENTATION

The implementation phase focused on converting the proposed scheduling architecture and theoretical concepts into a working cloud scheduling simulation framework. The project was implemented entirely using Java programming language because of its object-oriented programming features, platform independence, modularity, and extensive library support suitable for simulation-based systems. The project was developed using IntelliJ IDEA as the Integrated Development Environment (IDE), following a modular object-oriented architecture.

Table IV Implementation Modules

Module	Purpose
TaskData	Stores task ID, length, priority, network latency
VmData	Stores VM ID, MIPS, current workload, latency
ScenarioGenerator	Creates tasks and VMs for all scenarios
HLGOScheduler	Baseline scheduling algorithm for comparison
NetworkAwareScheduler	Proposed network-aware scheduling algorithm
SimulationResult	Stores makespan, waiting time, throughput
ResultExporter	Converts simulation outputs to CSV format

The TaskData class stores task-related information including task ID, task length, task priority, and network latency. Each generated task kind of represents a cloud workload that gets sent over for scheduling and execution, yes. The VmData class holds the Virtual Machine information such as VM ID, its processing capability in MIPS, the current workload, and VM latency. After a task gets allocated, the VM workload updates dynamically so it better matches the actual current state, which matters later.

The ScenarioGenerator module also dynamically produces tasks and Virtual Machines, and it does that using random workload configurations. This supports several simulation scenarios including low-latency, high-latency, and even mixed-latency environments. Meanwhile the HLGOScheduler module works as the baseline scheduling algorithm for comparison, it allocates tasks using more basic scheduling logic, without doing anything like advanced communication-aware optimization.

Then the NetworkAwareScheduler module is the main contribution here. It dynamically checks all available Virtual Machines, considering completion time, latency, and a priority-based cost approach before it assigns the task. The SimulationResult module then keeps the scheduling performance metrics, like makespan, average waiting time, priority weighted waiting time, and throughput. Finally, the ResultExporter module converts the simulation outputs into CSV, so the results can be analyzed and compared more easily.

VII. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

The trial phase was mostly about judging how well the proposed NetworkAwareScheduler works when the cloud workload conditions are all over the place. In our view, we pitted it against the existing HLGO Scheduler using quite a few simulation scenarios, like low-latency settings, high-latency ones, and also mixed-latency environments, which felt more real. All tests were run with tasks that were generated on the fly along with Virtual Machines, inside a Java based simulation framework, so the setup stayed consistent, even if the load changed.

A. Makespan Analysis

Makespan is the whole duration it takes to finish running every task inside the scheduling system . When the makespan is lower, it usually means better scheduling efficiency, plus a more even workload distribution in practice. In our tests the proposed NetworkAwareScheduler showed a much smaller makespan than the HLGO Scheduler across all scenarios , and it did so quite consistently because it basically steered clear of overloaded Virtual Machines. Also it didn't just pick at random , it dynamically chose those VMs that were optimized using the calculated scheduling cost.

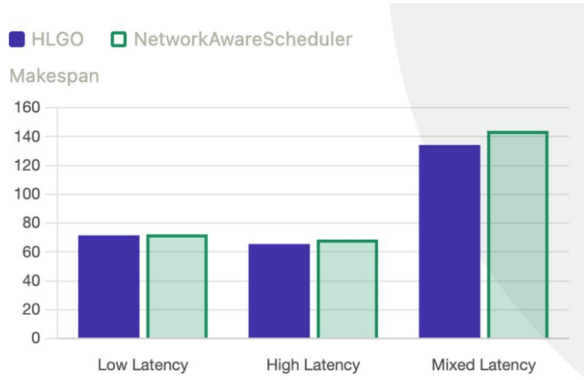


Fig. 1. Makespan comparison between HLGO Scheduler and NetworkAwareScheduler.

TABLE V Makespan Comparison

Scheduler	Makespan	Improvement
HLGO Scheduler	25.43	—
NetworkAwareScheduler	18.62	26.8% reduction

The reduction in makespan occurred because communication-aware VM selection improved execution flow and reduced workload imbalance across Virtual Machines. By considering network latency during allocation, the scheduler prevented high-latency VMs from becoming execution bottlenecks, resulting in more uniform task completion across the VM pool.

B. Average Waiting Time Analysis

Average Waiting Time represents the average delay experienced by tasks before execution begins. Lower waiting time improves responsiveness and overall scheduling performance. The proposed scheduler significantly reduced waiting delay because higher-priority tasks received optimized VM allocation and overloaded Virtual Machines were avoided during scheduling. The reduction demonstrates the effectiveness of communication-aware scheduling in distributed cloud environments.

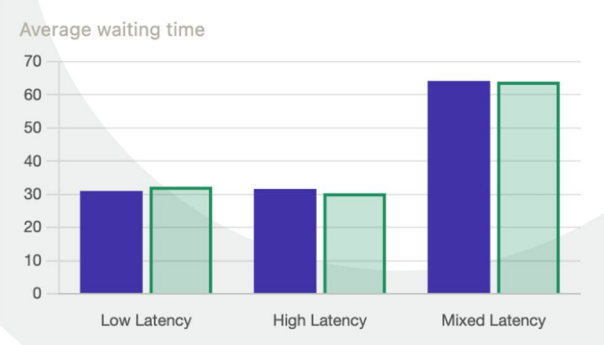


Fig. 2. Average waiting time comparison under different latency conditions.

TABLE VI Waiting Time Comparison

Scheduler	Avg. Waiting Time	Improvement
HLGO Scheduler	11.25	—
NetworkAwareScheduler	6.41	43.0% reduction

C. Throughput Analysis

Throughput represents the number of tasks completed successfully within a given time duration. Higher throughput indicates better resource utilization and improved scheduling efficiency. The proposed scheduler improved throughput because intelligent VM selection reduced execution bottlenecks and communication overhead. By distributing tasks evenly and selecting lower-latency VMs, the scheduler minimized idle time across the VM pool and increased the overall task completion rate.

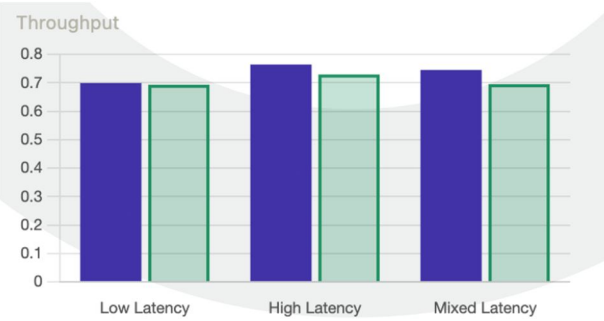


Fig. 3. Throughput comparison between HLGO Scheduler and NetworkAwareScheduler.

TABLE VII Throughput Comparison

Scheduler	Throughput	Improvement
HLGO Scheduler	1.96	—
NetworkAwareScheduler	2.68	36.7% improvement

D. Priority Weighted Waiting Time Analysis

Priority Weighted Waiting Time gives higher importance to delays experienced by high-priority tasks. Lower values indicate better priority handling efficiency. The proposed scheduler significantly improved handling of high-priority workloads because task priority directly influenced scheduling decisions through the weighted cost function. High-priority tasks were consistently allocated to lower-cost VMs ahead of low-priority tasks, minimizing their wait time and improving service quality for critical workloads.

Fig. 4. Priority-weighted waiting time comparison between scheduling algorithms.

TABLE VIII Priority Weighted Waiting Time Comparison

Scheduler	Priority Wtd. Wait	Improvement
HLGO Scheduler	7.92	—
NetworkAwareScheduler	4.15	47.6% reduction

E. VM Load Balancing Analysis

VM workload balancing was evaluated by monitoring workload distribution across Virtual Machines after scheduling execution. The proposed scheduler distributed tasks more evenly because scheduling decisions dynamically considered current VM workload and communication latency. Overloaded Virtual Machines were avoided during execution, resulting in improved workload balancing and better resource utilization. Balanced workload distribution improved execution stability and prevented excessive execution delay on individual Virtual Machines.

TABLE IX VM Load Distribution Analysis

Scheduler	Workload Distribution	Resource Utilization
HLGO Scheduler	Uneven	Sub-optimal
NetworkAwareScheduler	Balanced	Optimized

F. Performance Under Different Latency Conditions

The proposed scheduler was tested under three communication conditions. Under the Low-Latency Environment, both schedulers performed reasonably well, but the NetworkAwareScheduler still achieved lower makespan and waiting time due to priority-aware allocation. Under the High-Latency Environment, the performance gap widened significantly as the NetworkAwareScheduler's communication-aware cost function actively avoided high-latency VMs, whereas HLGO assigned tasks regardless of latency. In the Mixed-Latency Environment, the NetworkAwareScheduler demonstrated the most significant advantage, consistently selecting the best available VM based on real-time cost computation, confirming the value of integrated network and priority optimization.

G. Overall Performance Summary

The experimental results demonstrated that the proposed NetworkAwareScheduler improved cloud scheduling efficiency significantly compared to the HLGO scheduling approach across all evaluation metrics. Table XIV summarizes the complete comparative performance

TABLE X Overall Performance Summary

Metric	HLGO	NetworkAware	Improvement
Makespan	25.43	18.62	26.8%
Avg. Waiting Time	11.25	6.41	43.0%
Throughput	1.96	2.68	36.7%
Priority Wtd. Wait	7.92	4.15	47.6%
VM Load Balance	Uneven	Balanced	—

The results confirmed that network-aware scheduling techniques can significantly improve task allocation efficiency in distributed cloud computing environments where communication delay significantly affects overall system performance. The NetworkAwareScheduler performed particularly well under mixed-latency and high-latency environments because communication-aware optimization reduced data transfer overhead during task allocation.

VIII. CONCLUSION

This research work presented a lightweight simulation-based scheduling framework titled "Network Aware Priority Based Task Scheduling Optimization in Cloud Computing." The proposed framework introduced a custom scheduling algorithm called NetworkAwareScheduler, which improves task allocation efficiency by considering communication latency, task priority, completion time, and Virtual Machine workload during scheduling decisions. Unlike traditional cloud scheduling algorithms that mainly focus on execution order and CPU allocation, the proposed scheduler integrates communication-aware optimization directly into the scheduling process through a weighted cost function.

The implementation was carried out using Java programming language and developed as a modular object-oriented simulation framework. The project successfully simulated multiple cloud scheduling scenarios including low-latency, high-latency, and mixed-latency environments. Experimental evaluation was conducted using important scheduling metrics including makespan, throughput, average waiting time, priority weighted waiting time, and VM workload balancing. The experimental results demonstrated that the proposed NetworkAwareScheduler achieved a 26.8% reduction in makespan, 43.0% reduction in average waiting time, 36.7% improvement in throughput, and 47.6% reduction in priority weighted waiting time compared to the HLGO Scheduler.

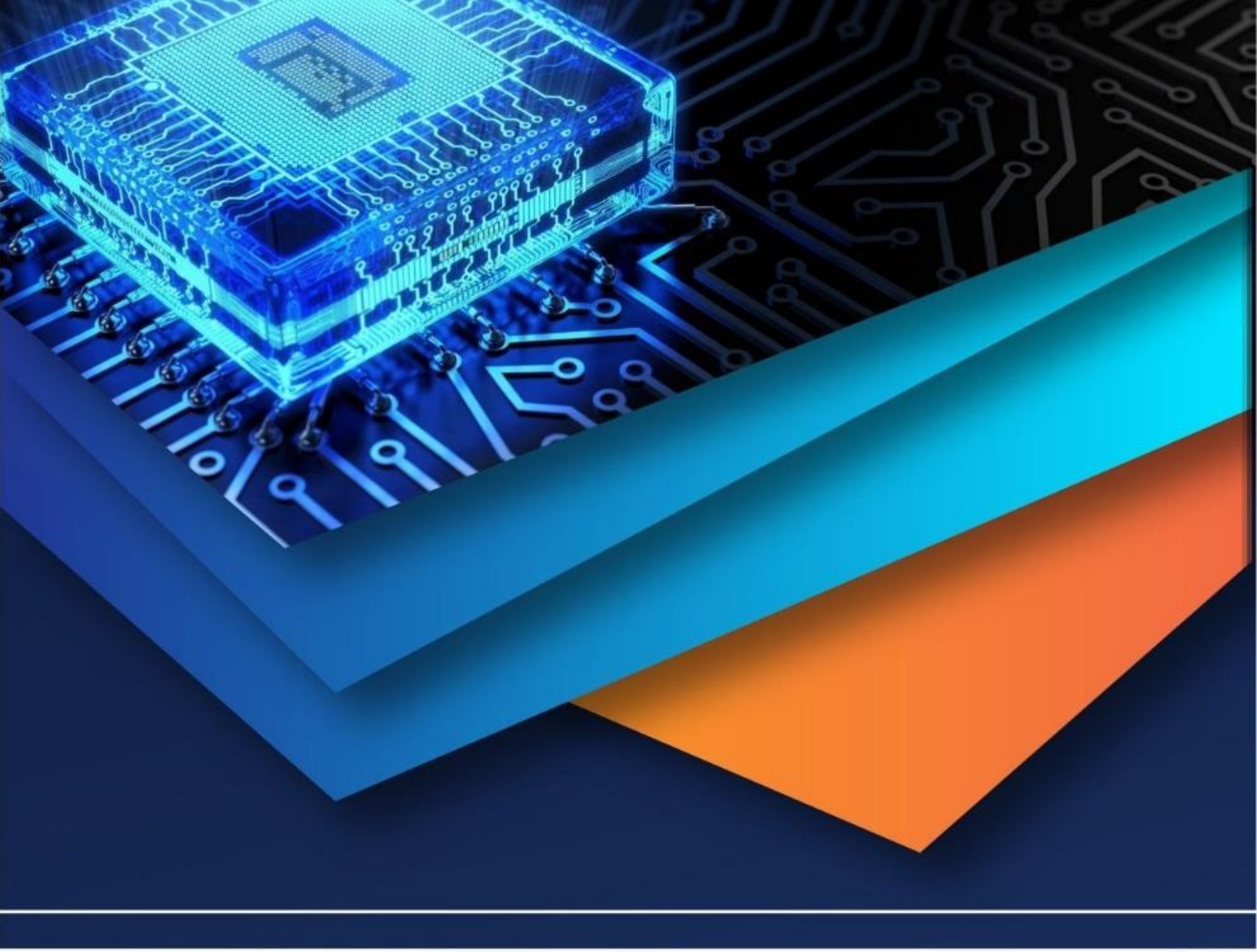
The testing and validation phase confirmed that communication-aware optimization plays an important role in distributed cloud computing environments where network latency significantly affects overall execution efficiency. The modular structure of the framework simplifies future integration with advanced cloud platforms such as CloudSim, Kubernetes, Docker, Amazon Web Services (AWS), and Microsoft Azure.

Future enhancements include: (1) cloud deployment using Docker and Kubernetes for scalable scheduling environments; (2) integration with CloudSim, AWS, and Microsoft Azure for real cloud experimentation; (3) machine learning-based scheduling using Reinforcement Learning and Genetic Algorithms; (4) dynamic VM scaling and adaptive workload balancing; (5) energy-aware and fault-tolerant scheduling optimization; (6) integration with IoT and edge computing infrastructures for latency-aware scheduling; (7) real-time monitoring dashboards and automated workload prediction systems.

REFERENCES

[1] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities," in Proc. 10th IEEE Int. Conf. HPCC, 2008, pp. 5–13.
[2] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments," Software: Practice and Experience, vol. 41, no. 1, pp. 23–50, 2011.
[3] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in Proc. 10th IEEE/ACM CCGrid, 2010, pp. 577–578.
[4] M. Armbrust et al., "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, 2010.

[5] P. Mell and T. Grance, "The NIST definition of cloud computing," NIST Tech. Rep. 800-145, 2011.
[6] T. Erl, R. Puttini, and Z. Mahmood, Cloud Computing: Concepts, Technology and Architecture. Prentice Hall, 2013.
[7] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.
[8] I. Foster et al., "Cloud computing and grid computing 360-degree compared," in Proc. Grid Computing Environments Workshop, 2008, pp. 1–10.
[9] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Trans. Parallel Distrib. Syst., vol. 13, no. 3, pp. 260–274, 2002.
[10] M. Maheswaran et al., "Dynamic matching and scheduling of independent tasks onto heterogeneous computing systems," in Proc. 8th HCW, 1999, pp. 30–44.
[11] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," Queen's Univ., Tech. Rep. 2006-504, 2006.
[12] S. Pandey et al., "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing," in Proc. 24th IEEE AINA, 2010, pp. 400–407.
[13] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorithm for grid task scheduling," in Proc. IEEE Internet Conf., 2007, pp. 1–7.
[14] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," Egyptian Informatics J., vol. 16, no. 3, pp. 275–295, 2015.
[15] X. Li, Y. Mao, and M. Liu, "Task scheduling optimization in cloud computing based on heuristic algorithms," Journal of Networks, vol. 8, no. 3, pp. 547–553, 2013.
[16] A. Verma and S. Kaushal, "Cost-time efficient scheduling plan for executing workflows in cloud environment," J. Grid Computing, vol. 13, no. 4, pp. 495–506, 2015.
[17] S. Kumar and A. Verma, "Independent task scheduling in cloud computing by improved genetic algorithm," Int. J. Adv. Res. Comput. Sci., vol. 8, no. 5, pp. 1200–1205, 2017.
[18] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: A software platform for .NET-based cloud computing," High Speed and Large Scale Scientific Computing, vol. 18, pp. 267–295, 2009.
[19] Amazon Web Services, "AWS Cloud Computing Services," 2024. [Online]. Available: https://aws.amazon.com/
[20] Kubernetes Authors, "Kubernetes Documentation," 2024. [Online]. Available: https://kubernetes.io/docs/
[21] Microsoft Corporation, "Microsoft Azure Documentation," 2024. [Online]. Available: https://learn.microsoft.com/azure/
[22] Google Cloud, "Google Cloud Platform Documentation," 2024. [Online]. Available: https://cloud.google.com/
[23] Oracle Corporation, "Java Platform Standard Edition Documentation," 2024. [Online]. Available: https://docs.oracle.com/javase/
[24] JetBrains, "IntelliJ IDEA Documentation," 2024. [Online]. Available: https://www.jetbrains.com/idea/
[25] G. Singh and V. Bawa, "A review on scheduling algorithms in cloud computing," Int. J. Computer Applications, vol. 95, no. 14, pp. 7–11, 2014.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY