IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

Cloud Resource Management Using Load Balancing

Mohith M[1], Brahmanand[1], P A Venkata Sai[1], Punith H N[1], Sharmila Chidaravalli[2]
[1]Students, Department of Information Science and Engineering
[2]Assistant Professor, Department of Information Science and Engineering
Global Academy of Technology, Bengaluru, Karnataka, India
mohithm1ga23is080@gmail.com, brahmamethre2003@gmail.com, venkatsaipermi@gmail.com,
Puni22412@gmail.com, Sharmila.c@gat.ac.in

Abstract : *This research paper aims at exploiting efficient ways of managing cloud resources through dynamic load balancing. In cloud computing architectures, unpredictable computational workloads often lead to resource underutilization, server bottlenecking, and degraded quality of service. This paper examines efficient distribution of computational tasks across virtual machines (VMs), achieved by integrating dynamic load balancing algorithms into cloud resource management frameworks. It presents a performance analysis of several load balancing strategies – Round Robin, Throttled, and Dynamic Least Connection – against a proposed telemetry-driven Composite Load Index scheduler, evaluated through CloudSim-based simulation across uniform, bursty, and mixed heterogeneous workload profiles. The discussion additionally covers computational overhead, deployment trade-offs, and threats to the validity of the reported results. This work supports both the design of optimized virtual server clustering and the operational needs of large-scale enterprise environments requiring real-time task scheduling and efficient resource allocation.*

Keywords: Cloud Computing, Dynamic Load Balancing, Resource Management, Virtual Machines, Quality of Service (QoS), Task Scheduling, CloudSim

I. INTRODUCTION

A. Background & Context

Cloud computing has reshaped how modern organizations think about IT infrastructure, replacing fixed, owned hardware with scalable, on-demand access to shared computing resources [1]. The resources involved – physical servers, virtualized storage, network bandwidth, and software microservices – are abstracted away from the underlying hardware and provisioned dynamically over the internet as needed. As more organizations adopt cloud-native platforms, software-as-a-service (SaaS) offerings, and microservice-based architectures, the backend infrastructure supporting them faces workloads that are increasingly volatile and difficult to predict in advance.

This volatility is compounded by the multi-tenant nature of modern public and private clouds. A single physical host cluster may simultaneously serve a video-streaming workload with steady bandwidth demands, a batch analytics job with bursty CPU spikes, and a transactional database with strict latency requirements. Because these tenants do not coordinate their demand patterns, the aggregate load on any given host is effectively a superposition of independent, unpredictable processes. Traditional capacity-planning approaches, which size infrastructure around average or peak historical demand, tend to either over-provision (wasting capital and energy) or under-provision (risking SLA breaches during demand spikes).

Within this environment, effective Cloud Resource Management (CRM) becomes a foundational operational concern. CRM is chiefly responsible for matching incoming user requests to available physical or virtual computing capacity while still meeting Quality of Service (QoS) targets and honoring Service Level Agreements (SLAs) [9]. Load balancing sits at the heart of this process: it determines how demanding requests get spread across the available server

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

instances so that no single host becomes saturated, queuing delays stay low, and the cluster as a whole keeps up high throughput.

B. Problem Statement

In cloud environments serving many tenants at once, older static load-balancing approaches have difficulty keeping pace with conditions that change from moment to moment [4], [5]. Because static schedulers assign work using fixed formulas or preset host profiles rather than checking current system conditions, they tend to introduce several recurring problems:

- Overloading of individual servers: demanding jobs sent to already-busy hosts create processing backlogs, CPU throttling, thread starvation, and request timeouts.
- Wasted hardware capacity: while some nodes struggle, others may sit mostly idle, still drawing baseline power without doing productive work.
- SLA violations and latency spikes: poorly balanced workloads drive up execution time, queuing delay, and dropped packets, producing breached SLAs and degraded user experience [10].
- Cascading failure risk: once one node in a cluster is overloaded, retried or rerouted requests from that node can themselves overload neighboring nodes, producing a compounding failure pattern under sustained bursty traffic.

These issues point to a clear need for a load-balancing framework that adapts in real time – one that continuously tracks CPU load, memory consumption, and active connections, and uses that live picture to steer tasks toward the most appropriate compute instance.

C. Research Objectives

To address the shortcomings of static, single-metric scheduling, this study pursues the following objectives:

1) Examine how static task-distribution strategies contribute to resource-provisioning bottlenecks in virtualized data centers.
2) Compare legacy static load-balancing algorithms against telemetry-driven dynamic approaches.
3) Design a real-time monitoring engine evaluating CPU load, memory usage, and active connection counts together.
4) Distribute tasks across heterogeneous VMs to avoid localized hotspots and sustain throughput.
5) Reduce response time (Tresp) and execution delay (Texec) during sudden traffic spikes.
6) Establish an architecture robust enough to guarantee ongoing SLA compliance for commercial deployments.

D. Paper Organization

The remainder of this paper is organized as follows. Section II provides background on static and dynamic load-balancing architectures. Section III reviews related work and identifies the research gap this study addresses. Section IV describes the proposed methodology, including the mathematical formulation of the Composite Load Index. Section V details the experimental setup. Section VI reports results, and Section VII extends this into a broader discussion of overhead and limitations. Section VIII covers strategic relevance, Section IX outlines future work, and Section X concludes the paper.

II. BACKGROUND ANALYSIS

Modern cloud platforms depend heavily on virtualization to carve physical hardware into multiple isolated virtual machines (VMs). Sound CRM means spreading incoming tasks across those VMs in a way that satisfies QoS goals – low latency, strong throughput, and SLA adherence. Load balancing is the mechanism that makes this possible [1], and approaches generally fall into two broad categories.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 1, November 2026 Impact Factor: 8.2

A. Static Load Balancing Architecture

Static load balancers assign incoming requests to VMs according to a predetermined pattern or fixed hardware profile, implicitly assuming uniform host capacity and uniform request demand [2].

1) Operational Mechanics

A classic static algorithm – Round Robin – assigns each task to a VM using a deterministic modulo calculation over the number of active instances, executing in constant, O(1) time without inspecting host health [15]. Weighted variants extend this by assigning each VM a static capacity weight, so that higher-capacity hosts receive proportionally more requests per cycle – but the weight itself is fixed at configuration time and cannot reflect the VM's actual, current load.

2) System Vulnerabilities

Because static routing has no feedback from the running system, it breaks down under uneven or bursty workloads, sending heavy tasks to already-busy nodes and causing thread starvation and long queuing delays [3]. This vulnerability is most pronounced in heterogeneous clusters, where VM instances differ in vCPU count, memory allocation, and underlying hardware generation; a scheduling scheme blind to these differences will systematically misallocate work regardless of how carefully its rotation order is tuned.

B. Dynamic Load Balancing Architecture

Dynamic load balancers maintain an ongoing feedback loop between the Cloud Controller, the Virtual Machine Manager (VMM), and individual hypervisors [1], [17].

1) Operational Mechanics

Rather than deciding in isolation, a dynamic load balancer regularly polls hypervisors for live telemetry – CPU usage, allocated RAM, and active socket connections – before routing each new request [16]. The polling interval itself introduces a design trade-off: shorter intervals give the scheduler a more current picture of cluster state but add polling overhead, while longer intervals reduce overhead at the cost of decision staleness.

2) System Advantages

By continually identifying the least-loaded instance, a dynamic scheduler keeps workload spread evenly, reducing hotspots, lowering latency, and improving host utilization [18]. Because the decision is recomputed per request (or per short batch of requests) rather than fixed in advance, dynamic schedulers also degrade more gracefully as cluster composition changes – for instance, when VMs are added or removed during auto-scaling events.

III. RELATED WORK & COMPARATIVE SYNTHESIS

Task scheduling and resource provisioning in distributed, virtualized environments have been widely studied across academic and industry literature [19].

A. Review of Conventional Load Balancing Algorithms

1) Round Robin (RR) & Weighted Round Robin (WRR)

Standard RR cycles requests sequentially and runs at O(1) cost, but struggles in heterogeneous environments with varying task lengths. WRR assigns static capability weights but still cannot adapt to dynamic workload spikes [2], [4].

2) Least Connection Scheduling

This method routes traffic to the VM with the fewest active connections. It outperforms RR generally, but connection count does not always reflect actual CPU or memory saturation, leading to inaccurate estimates for memory-bound applications [9].

3) Min-Min and Max-Min Algorithms

These batch-scheduling algorithms prioritize short (Min-Min) or long (Max-Min) tasks to optimize makespan, but require advance knowledge of execution times, making them impractical for real-time, unpredictable request streams [5].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 1, November 2026 Impact Factor: 8.2

4) Heuristic & Meta-Heuristic Approaches

PSO, Genetic Algorithms, and Ant Colony Optimization achieve near-optimal offline placements, but their execution latency makes them unsuitable for low-latency, real-time delivery controllers [6], [8], [12].

B. Comparative Summary

Table I summarizes the qualitative trade-offs among the surveyed approaches and the proposed model along four dimensions: telemetry awareness, decision overhead, adaptability to bursty traffic, and typical deployment context.

TABLE I: QUALITATIVE COMPARISON OF SCHEDULING PARADIGMS

Algorithm	Telemetry Awareness	Overhead	Best Suited For
RR / WRR	None / Static	O(1)	Uniform, steady traffic
Least Connection	Partial (connections only)	O(1)	Session-heavy, low variance
Min-Min / Max-Min	Requires prior knowledge	O(N log N)	Offline batch scheduling
Metaheuristic (PSO/GA/ACO)	High, offline	High (iterative)	Latency-tolerant batch placement
Proposed (Composite Load Index)	Continuous, multifactor	O(N)	Heterogeneous, bursty, mixed workloads

C. Detailed Comparative Explanation of Scheduling Paradigms

1) Real-Time Telemetry & Awareness

Standard RR operates blind to host conditions; WRR improves on this with static weights that cannot adjust later. Meta-heuristics achieve high awareness offline; the proposed model instead maintains continuous awareness via live polling of CPU, memory, and connection metrics [14].

2) Computational Complexity & Decision Overhead

RR and WRR run at negligible O(1) overhead; heuristic methods incur significant delay from multi-iteration search. The proposed model balances precision and speed at O(N) local complexity, where N is the number of active VMs under consideration – a single linear pass to compute Li per candidate and select the minimum, rather than an iterative search over a solution space.

3) Adaptability to Workload Spikes

RR and WRR perform poorly under bursty traffic; heuristics adapt moderately with latency; the proposed model reroutes instantly away from saturated nodes [20].

4) Deployment Environments

Static RR suits uniform, steady-traffic environments; WRR suits static heterogeneous servers; meta-heuristics suit latency-tolerant batch processing; the proposed model targets modern heterogeneous cloud infrastructure with unpredictable traffic [19].

D. Identified Research Gap

Existing dynamic load-balancing systems often rely on a single metric (CPU load alone, or connection count alone), which breaks down for mixed workloads that are memory-intensive or network-bound rather than compute-bound [13]. This study addresses that gap with a multi-factor Composite Load Index (Li) evaluating CPU, RAM, and connection density together, while remaining lightweight enough for real-time use.

IV. PROPOSED METHODOLOGY

A. System Architecture & Core Components

The proposed CRM framework is a multi-tier, telemetry-driven microservice architecture with five components:

1) User Client Tier: external applications, IoT streams, and interfaces submitting requests (HTTP/HTTPS, REST APIs, compute-bound tasks).

2) Cloud Controller Gateway: entry point for traffic; extracts QoS metadata and forwards payloads.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 1, November 2026 Impact Factor: 8.2

3) Dynamic Telemetry Load Balancer: central decision engine computing node availability via a composite formula.
4) Virtual Machine Manager (VMM) & Hypervisors: manage virtualization, poll hardware metrics, report status.
5) Virtual Compute Nodes (VMs): execute assigned workloads and return results, triggering utilization updates.
Each component communicates over a lightweight internal messaging channel, allowing the telemetry loop (components 3–5) to operate independently of the request-handling loop (components 1–2), so that metric collection does not block request forwarding.

B. Mathematical Model & Multi-Factor Decision Engine

Let V = {VM1, VM2, …, VMn} be the active VM pool and T = {t1, t2, …, tm} be the incoming request stream. For each virtual machine VMi, the Composite Load Index is defined as:

$$Li = w1UCPU(i) + w2URAM(i) + w3CActive(i) \quad (1)$$

where UCPU(i), URAM(i), and CActive(i) represent the current CPU utilization, memory utilization, and normalized active connection density of VMi, respectively. The weights satisfy:

$$w1 + w2 + w3 = 1, \; w1, w2, w3 \geqslant 0 \quad (2)$$

The weights are configurable according to workload characteristics. For compute-intensive workloads, a larger value of w1 is preferred, whereas memory-intensive workloads use a larger value of w2. This configurability is what allows a single formula to serve heterogeneous workload classes without requiring a separate scheduler implementation for each class.

Let Tmax denote the maximum permissible load threshold (typically 0.85). Virtual machines satisfying Li > Tmax are excluded from receiving new requests. The scheduler assigns an incoming task tj to the least loaded active virtual machine:

$$VMk = \arg\min VMi \in Vactive\} Li \quad (3)$$

C. Algorithmic Implementation

Algorithm 1 formalizes the dispatch logic.

Algorithm 1: Dynamic Telemetry-Based Load Balancing Input: Task Tj; VM set {VM1, …, VMn}
Output: Target VMk
1. Set weights w1, w2, w3 (sum = 1)
2. Set Tmax = 0.85
3. Vactive = {}
4. For each VMi:
• Read CPU, RAM, Active Connections
• Compute Li
• If Li < Tmax, add VMi to Vactive
• Else mark VMi as SATURATED
5. If Vactive is empty: Trigger auto-scaling
6. Else: Select VMk = arg min(Li); Dispatch task to VMk; Update VM state

D. Complexity Analysis

Steps 4 and 6 of Algorithm 1 dominate its running time. Step 4 iterates once over all n candidate VMs to compute Li and classify saturation, an O(n) operation. Step 6's arg-min selection can be folded into the same pass by tracking a running minimum, keeping the overall per-request dispatch cost at O(n) rather than O(n log n). This is asymptotically worse than RR's O(1) but remains lightweight relative to meta-heuristic approaches, whose iterative search procedures typically require multiple O(n) or O(n2) passes per convergence cycle.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

V. EXPERIMENTAL SETUP & SIMULATION

A. Simulation Environment Specifications

Testing on production cloud data centers carries operational risk. Therefore, this study uses CloudSim, a discrete-event cloud simulator, for experimental evaluation. The simulation environment models two virtualized data centers across thirty physical hosts (3.0–3.6 GHz multi-core CPUs, 64–128 GB RAM, and Gigabit Ethernet networking). A total of fifty heterogeneous virtual machines (VMs) are provisioned with the following configurations:

- Small VM: 1 vCPU, 2 GB RAM, 1000 MIPS.
- Medium VM: 2 vCPUs, 4 GB RAM, 2500 MIPS.
- Large VM: 4 vCPUs, 8 GB RAM, 5000 MIPS.

B. Workload Datasets & Scenario Modeling

To evaluate system performance under unpredictable conditions, simulated task streams (Cloudlets) ranging from 1,000 to 10,000 concurrent requests were generated. Workloads are categorized into three distinct operational profiles:

1) Uniform Steady Workloads: Tasks with identical instruction lengths (2,000 to 5,000 MI) submitted at a constant, predictable rate to establish baseline metrics.

2) Dynamic Bursty Traffic Spikes: Simulated sudden, unexpected traffic spikes where request volume increases by 300% over short time windows, testing algorithm adaptability during sudden bursts.

3) Mixed Heterogeneous Workloads: A combination of short API tasks, memory-bound database queries, and compute-intensive batch processing operations, designed to test the algorithm's multi-factor telemetry capabilities under varied resource demands.

C. Baseline Algorithms for Comparison

The proposed Composite Load Index scheduler is benchmarked against three baselines commonly used in cloud simulation literature: standard Round Robin, the Throttled algorithm (which caps the number of concurrent requests routed to a given VM), and Dynamic Least Connection (which routes based on active connection count alone, without CPU or memory awareness).

VI. RESULTS & PERFORMANCE EVALUATION

A. Response Time & Execution Latency Analysis

CloudSim results show clear gains for the proposed model over legacy static scheduling. Static algorithms perform adequately under steady workloads but degrade sharply under bursty or heterogeneous traffic, as tasks queue behind long-running jobs on overloaded hosts. The proposed model, by continuously tracking CPU, RAM, and connection density and rerouting away from saturated nodes, reduces average response latency by approximately 28%–35% during high-density traffic.

B. Cluster Resource Utilization & Load Balance Index

Under Round Robin, high-capacity instances reach 90–98% utilization while neighboring instances sit under 15%, wasting hardware capacity. The proposed Composite Load Index, by factoring memory and connection count alongside CPU usage and enforcing Tmax = 0.85, produces a markedly more even load distribution, raising average cluster efficiency above 82% and eliminating localized bottlenecks.

C. SLA Compliance & Throughput Analysis

At 10,000 concurrent requests, static Round Robin produced an SLA breach rate of nearly 14.2% from queue timeouts on overloaded nodes. The proposed dynamic balancer reduced this to under 1.8%, supporting its suitability for enterprise-grade QoS requirements.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

VII. DISCUSSION

A. Interpreting the Overhead–Accuracy Trade-off

The results in Section VI reflect a deliberate design trade-off: the Composite Load Index incurs O(n) decision overhead per request in exchange for load-awareness that O(1) static schedulers cannot provide. In practice, this overhead is small relative to network round-trip time and task execution time, particularly for the medium and large VM tiers used in this simulation, so the latency reduction from better placement outweighs the added scheduling cost. For extremely latency-sensitive, sub-millisecond workloads, however, this trade-off would need re-evaluation, since even an O(n) scan can become non-negligible at very large cluster sizes.

B. Sensitivity to Weight Configuration

Because w1, w2, and w3 are operator-configured rather than learned, the scheduler's effectiveness depends on selecting weights that reasonably reflect the dominant resource constraint of the deployed workload. A compute-bound deployment with w1 set too low, for instance, would under-weight CPU saturation and could still permit hotspots the model is otherwise designed to prevent. This is a limitation the future-work directions in Section IX (particularly predictive, learning-based weight adjustment) are intended to address.

C. Threats to Validity

As with any simulation-based evaluation, several factors limit how directly these findings generalize to production environments:

• Simulator fidelity: CloudSim models network and hypervisor behavior at a level of abstraction that does not capture every real-world variable (e.g., NUMA effects, live-migration overhead, or noisy-neighbor interference at the physical host level).

• Synthetic workload generation: while the three workload profiles are designed to approximate uniform, bursty, and mixed real-world traffic, they remain synthetic and may not fully reflect the temporal correlation structure of production traffic traces.

• Fixed threshold selection: Tmax = 0.85 was chosen as a representative operating point; its optimal value likely varies by workload class and would benefit from empirical tuning in a deployed setting.

VIII. STRATEGIC RELEVANCE & APPLICATIONS

A. Industry Relevance

• Enterprise Multi-Tenant Cloud Platforms: Public and private cloud providers (SaaS, PaaS, IaaS) can deploy telemetry-driven scheduling engines to ensure seamless auto-scaling and continuous zero-downtime operations during unexpected user surges.

• E-Commerce & Digital Commerce Infrastructure: Online platforms experiencing severe traffic surges during flash sales or promotional events benefit from instantaneous real-time task rerouting, preventing shopping cart session timeouts and server crashes.

• Financial Transaction Processing: High-frequency banking applications and payment gateways require deterministic response times; real-time dynamic load management ensures critical transactions bypass congested processing nodes.

B. Societal Impact & Green Computing Initiatives

• Energy Efficiency & Carbon Footprint Reduction: Data centers consume vast amounts of electrical power. By optimizing host utilization and preventing idle physical hardware from drawing baseline power without processing tasks, dynamic balancing supports sustainable, green computing initiatives.

• Reliability of Public Critical Services: Cloud-hosted public infrastructure – including healthcare monitoring systems, emergency communication portals, and distance learning platforms – gains continuous, resilient operational uptime even during emergency usage spikes.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

IX. FUTURE WORK & RESEARCH DIRECTIONS

While the proposed multi-factor dynamic model demonstrates high efficacy, several promising avenues exist for future research:

1) Machine Learning & Predictive Workload Forecasting: Integrating deep learning time-series models (such as Long Short-Term Memory networks or Reinforcement Learning agents) to predict incoming traffic spikes and proactively rebalance VM clusters prior to node saturation [14].

2) Energy-Aware Multi-Objective Scheduling: Expanding the Composite Load Index formula (Li) to incorporate physical host temperature and power consumption metrics, prioritizing energy-efficient hardware nodes.

3) Container-Level Orchestration (Kubernetes Integration): Adapting the telemetry model to lightweight containerized environments (Docker/Kubernetes) for microservice pod balancing.

4) Fault-Tolerant & Security-Aware Routing: Developing Byzantine fault-tolerant dynamic routing protocols capable of identifying and isolating compromised compute nodes under Distributed Denial of Service (DDoS) cyberattacks.

5) Adaptive Weight Learning: Replacing static, operator-configured w1, w2, w3 values with weights learned online from observed workload behavior, reducing the sensitivity described in Section VII-B.

6) Production Trace Validation: Re-evaluating the reported gains against real production traffic traces (rather than synthetic Cloudlet generation) to address the threats to validity raised in Section VII-C.

X. CONCLUSION

Dynamic load balancing is foundational to modern cloud resource management. Static scheduling is poorly suited to bursty, unpredictable cloud traffic, producing hotspots, wasted capacity, and SLA violations. This paper presented a telemetry-driven framework built around a multi-factor Composite Load Index (Li) evaluating CPU, RAM, and active connections jointly. The approach reduces response time, improves cluster utilization, and strengthens QoS guarantees while maintaining low computational overhead, as confirmed by the simulation results. At the same time, the overhead–accuracy trade-off, weight-configuration sensitivity, and simulation-based threats to validity discussed in Section VII outline concrete, bounded directions for the follow-on work proposed in Section IX.

REFERENCES

[1] S. Kaur, D. Saxena, and A. K. Singh, "Characterizing Dynamic Load Balancing in Cloud Environments Using Virtual Machine Deployment Models," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/document/8856192

[2] "Load Balancing Algorithms in Cloud Computing: A Survey of Modern Techniques," in Proc. IEEE Conf., 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7396341

[3] "A Survey of Load Balancing Challenges in Cloud Environment," in Proc. IEEE Conf., 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7894537

[4] "A Comparative Survey on Load Balancing Algorithms in Cloud Computing," in Proc. IEEE Conf. [Online]. Available: https://ieeexplore.ieee.org/document/6963138

[5] "A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms," in Proc. IEEE Conf. [Online]. Available: https://ieeexplore.ieee.org/document/6472470

[6] "Load Balancing in Cloud Computing: A Survey on Comparison of Two Algorithms PSO and SJF-MMBF," in Proc. IEEE Conf., 2019. [Online].
Available: https://ieeexplore.ieee.org/document/8820268

[7] "Load Balancing in Cloud Computing: Methodological Survey on Different Types of Algorithm," in Proc. IEEE Conf., 2018. [Online].
Available: https://ieeexplore.ieee.org/document/8300865

[8] "Intelligent Cloud Algorithms for Load Balancing Problems: A Survey," in Proc. IEEE Conf., 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7397223

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 1, November 2026 Impact Factor: 8.2

[9] "A Survey on Scheduling and Load Balancing Techniques in Cloud Computing Environment," in Proc. IEEE Conf., 2014. [Online]. Available: https://ieeexplore.ieee.org/document/7001474
[10] "A Survey on and Performance Analysis of Load Balancing Algorithms using Meta Heuristics Approach in Public Cloud-Service Provider's Perspective," in Proc. IEEE Conf., 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8997138
[11] "Significance of Load Balancing with Task Scheduling in Cloud Environment," in Proc. IEEE Conf. [Online]. Available: https://ieeexplore.ieee.org/document/11096508
[12] "Load Balancing Task Scheduling Based on Genetic Algorithm in Cloud Computing," in Proc. IEEE Conf., 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6945680
[13] R. Zhanuzak, M. A. Ala'Anzy, M. Othman, and A. Algarni, "Optimizing Cloud Computing Performance With an Enhanced Dynamic Load Balancing Algorithm for Superior Task Allocation," IEEE Access, vol. 12, pp. 183117–183132, 2024, doi: 10.1109/ACCESS.2024.3508793.
[14] E. Gures, I. Shayea, M. Ergen, et al., "Machine Learning-Based Load Balancing Algorithms in Future Heterogeneous Networks: A Survey," IEEE Access, vol. 10, pp. 37689–37717, 2022, doi: 10.1109/ACCESS.2022.3161511.
[15] "The Load Balancing Algorithm in Cloud Computing Environment," in Proc. IEEE Conf., 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6526078
[16] "Improved Dynamic Load Balancing Approach in Cloud Computing," in Proc. IEEE Conf., 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9290602
[17] "An Efficient Dynamic Load Balancing Algorithm for Virtual Machine in Cloud Computing," in Proc. IEEE Conf., 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9065292
[18] "Dynamic Load Balancing and Distribution Algorithm in Distributed Cloud Computing," in Proc. IEEE Conf., 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10515775
[19] "Load Balancing Strategies for Cloud Computing: A Comprehensive Review," in Proc. IEEE Conf., 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10467233
[20] "Response Time Analysis of Dynamic Load Balancing Algorithms in Cloud Computing," in Proc. IEEE Conf., 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9210305