

Comparative Analysis of Load Balancing Algorithms in Cloud Data Centres

Ibiyomi, Michael Adeniyi

Department of Computer Science Federal Polytechnic, Ile Oluji Ondo

DOI: <https://doi.org/10.51584/IJRIAS.2026.11090070>

Received: 02 September 2026; Accepted: 07 September 2026; Published: 12 October 2026

ABSTRACT

Cloud computing has become the dominant paradigm for scalable and flexible resource provisioning, enabling on demand access to computational resources across diverse application domains. Within cloud data centres, an effective load balancing strategy is critical to achieving optimal resource utilisation, minimising service response times, and maximising throughput. This paper presents a comparative analysis of two well established load balancing algorithms—Ant Colony Optimization (ACO) and Throttle—in the context of cloud data centre environments. Using the CloudSim simulation toolkit, a controlled data centre model is constructed to evaluate both algorithms under identical workload and infrastructure conditions. Performance is assessed using three primary metrics: average response time, system throughput, and resource utilisation. Furthermore, a classification based performance evaluation is conducted using accuracy, precision, recall, and F1 score, measuring the correctness of scheduling decisions across varying workload patterns. Simulation results demonstrate that ACO consistently outperforms the Throttle algorithm, particularly under high and peak load conditions. Specifically, under peak load, ACO reduces average response time by 34.2%, improves throughput by 21.5%, and increases resource utilisation efficiency by 10.8% relative to Throttle. Conversely, the Throttle algorithm exhibits degraded performance as workload intensity rises, although it achieves marginally better efficiency under very low load conditions. These findings provide quantitative insights into the trade offs between swarm intelligence based and threshold based dynamic load balancing strategies, offering practical guidance for cloud resource management.

Keywords: Ant Colony Optimisation (ACO), Amazon Web Services (AWS), Virtual Machines (VMs), Service Level Agreements (SLAs), Throttle, Hybrid

INTRODUCTION

Cloud computing's use as a service model has completely revolutionized the approach to managing computational resources for business organizations. Well-known cloud service providers like AWS, Microsoft Azure, and Google Cloud Platform offer their services to millions of users simultaneously and deal with loads that range from small and short-lived to large and long-running applications. In terms of the underlying hardware, the cloud centres consist of thousands of virtual machines (VMs) spread out across various physical locations. One of the most critical challenges in cloud resource management is load balancing—the effective assignment of arriving user requests to different VMs (Shafiq et al., 2021; Sherwani et al., 2021).

Load balancing is essential because its absence leads to several negative consequences: increased response delays, reduced throughput, overloading of certain VMs, and underutilization of others—all of which result in violations of service level agreements (SLAs). Conversely, proper load balancing improves fault tolerance, lowers latency, optimizes resource utilization, and guarantees high quality of service (QoS) for end users. Due to the unpredictability of workload profiles, load balancing algorithms must simultaneously achieve competing goals: minimizing execution time, reducing power consumption, and maximizing throughput (Mahdizadeh et al., 2024; Zhanuzak et al., 2024).

Broadly, load balancing methods fall into two categories: static and dynamic. Static algorithms, such as Round Robin (RR) and First-Come-First-Served (FCFS), are computationally efficient but fail to adapt to fluctuating workloads because they ignore real-time system status (Buttar et al., 2023). Dynamic algorithms, in contrast, consider the current state of VMs (e.g., CPU load, memory usage, network bandwidth) when making scheduling decisions, making them more suitable for heterogeneous and time-varying cloud environments.

Among dynamic algorithms, Ant Colony Optimization (ACO) has been extensively researched. Inspired by swarm intelligence, ACO uses probabilistic pheromone trails to guide request assignment, achieving near-optimal load distribution under high variability (Mavrovouniotis et al., 2020). Another established dynamic technique is the Throttled (TH) algorithm, which maintains an index of VMs and assigns a request only to an idle VM based on a simple threshold. Unlike ACO, Throttled does not employ evolutionary or swarm intelligence; its low computational overhead makes it attractive for real-time scenarios (Ibiyomi et al., 2026; Ghomi et al., 2017).

A comparative analysis of these algorithms reveals distinct trade-offs. Round Robin offers excellent scalability and minimal overhead but suffers from poor resource utilization when VMs are heterogeneous (Alakeel, 2010). Throttled reduces response time and SLA violations by avoiding overloaded nodes, yet its performance can degrade under bursty traffic because it may reject requests when all VMs appear busy (Chou & Chou, 2020). ACO, while delivering higher throughput and better fault tolerance, incurs greater computational complexity and longer convergence times, making it less suitable for extremely latency-sensitive workloads (Pacini et al., 2014; Oyediran et al., 2024).

Recent empirical studies using simulation frameworks like CloudSim and CloudAnalyst have quantified these differences. For instance, a comparative evaluation by Mondal et al. (2019) showed that ACO reduces average response time by approximately 22% compared to Throttled under heavy load, but at the cost of a 35% increase in algorithm execution overhead. Similarly, Arshad (2022) demonstrated that Throttled outperforms Round Robin in throughput by up to 40% in a three-data-centre configuration, while maintaining lower SLA violation rates. These findings underscore that no single algorithm is universally superior; the optimal choice depends on workload characteristics, system scale, and performance objectives (Prity & Hossain, 2024).

Therefore, a thorough comparative analysis of load balancing algorithms—considering metrics such as response time, throughput, resource utilisation, scalability, and SLA compliance—is essential for designing efficient cloud data centres. The following sections present a detailed literature review, a mathematical formulation of the two algorithms, a description of the comparative framework, and experimental results. This paper seeks to fill the identified gap by performing a comparative analysis of ACO and Throttle under identical conditions, using both traditional performance metrics and classification-based accuracy measures (Azam et al., 2023).

LITERATURE REVIEW

Static Load-Balancing Approaches

Static load-balancing approaches assign tasks according to predefined rules without continuously considering the current or future state of the cloud environment. The allocation decision is generally made using fixed policies such as task order, server position, or a predetermined distribution mechanism. Round Robin is a representative example in which incoming tasks are sequentially distributed among available virtual machines or servers. Such approaches are relatively simple, computationally inexpensive, and easy to implement. However, they do not adequately account for differences in processing capacity, current resource utilization, task requirements, or sudden workload fluctuations.

The principal limitation of static load balancing is therefore its inability to adapt allocation decisions to real-time changes in workload and resource conditions. In heterogeneous cloud data centers, assigning tasks

without considering the current state of individual VMs may result in some resources becoming overloaded while others remain underutilized. Consequently, although static methods provide low scheduling overhead, their effectiveness may decline as workload variability and resource heterogeneity increase.

Dynamic Load-Balancing Approaches

Dynamic load-balancing approaches overcome some of the limitations of static methods by incorporating information about the current state of the cloud environment when making scheduling decisions. These approaches can consider variables such as CPU utilization, memory consumption, queue length, bandwidth availability, task requirements, and VM capacity. The Throttle algorithm, for example, attempts to avoid assigning tasks to VMs that cannot satisfy specified resource requirements. Dynamic approaches therefore provide greater adaptability than fixed allocation policies, particularly when workload intensity and resource availability change over time.

Hybrid Load-Balancing Approaches

Hybrid load-balancing approaches combine two or more complementary techniques to overcome the limitations associated with individual algorithms. Rather than relying exclusively on fixed rules or a single adaptive mechanism, hybrid approaches integrate different capabilities, such as machine-learning-based prediction, metaheuristic optimization, constraint management, or feedback control. This makes hybrid load balancing particularly relevant to cloud environments in which workload patterns, resource availability, and performance requirements change continuously (Ibiyomi et al., 2025).

Representative hybrid approaches demonstrate several ways in which complementary algorithms can be integrated. For example, RF-PSO-CNN combines machine-learning and optimization techniques to improve cloud load-balancing decisions (Oyediran, 2024). Similarly, Sanjalawe et al. (2025) developed a CNN-LSTM-based approach that combines convolutional feature extraction with LSTM-based workload prediction. Such approaches demonstrate the benefit of combining predictive or learning capabilities with other computational techniques rather than depending on a single scheduling mechanism. Sugumar (2023) also proposed a hybrid neural-swarm architecture for adaptive and energy-efficient cloud load balancing, illustrating the application of machine learning and swarm intelligence within an integrated load-balancing strategy.

Hybrid approaches can also combine metaheuristic optimization algorithms with complementary optimization or control mechanisms. For instance, hybrid optimization research in cloud load balancing has investigated combinations of swarm-intelligence and other optimization techniques to improve the search for efficient resource allocations (Sugumar, 2023). The fundamental rationale is that one algorithm can compensate for the limitations of another: a prediction algorithm can anticipate workload changes, an optimization algorithm can search for efficient resource assignments, and a constraint or control mechanism can prevent infeasible or overloaded allocations.

Related Hybrid Load-Balancing Studies

Recent research has increasingly combined machine learning, deep learning, and metaheuristic optimization to improve workload prediction, resource allocation, energy efficiency, and SLA compliance. However, the components of these hybrid models are not always tightly integrated into a single prediction-optimization-control cycle.

Sanjalawe et al., (2025) developed a CNN-LSTM model for workload prediction and resource distribution in cloud computing. The combination of CNN-based feature extraction and LSTM-based temporal modelling improved the ability to identify workload trends and contributed to improved response time. However, the study primarily emphasizes workload prediction and does not provide an explicit metaheuristic optimization

mechanism for determining efficient task-to-VM assignments. This creates an opportunity to connect workload prediction directly with an optimization layer.

Sugumar, (2023) proposed a hybrid neural-swarm architecture for adaptive and energy-efficient load balancing. The study demonstrates the potential of combining learning-based methods with swarm intelligence to improve cloud resource management. Nevertheless, predictive intelligence and allocation optimization remain distinct concerns in many such architectures, and runtime resource constraints must still be explicitly managed to prevent infeasible assignments.

Adegbola (2025) proposed an adaptive load-balancing framework involving multiple machine-learning techniques, including decision trees, neural networks, clustering, and anomaly detection. The approach demonstrates the usefulness of combining learning methods for cloud resource management. However, the framework does not explicitly establish a workload forecasting mechanism coupled with a metaheuristic task-allocation process. Consequently, the approach remains primarily dependent on the information available at the time of decision-making.

Oyediran (2024) investigated an RF-PSO-CNN-based hybrid approach for cloud load balancing. The integration of random forest, particle swarm optimization, and convolutional neural networks illustrates the broader movement towards combining learning and optimization techniques. However, classification-oriented learning may not fully capture the continuous temporal characteristics of cloud workload demand. A forecasting-oriented model such as LSTM can provide a more direct mechanism for modelling sequential workload behaviour.

Simaiya et al., (2024) hybrid optimization algorithms for cloud load balancing and highlighted the increasing importance of combining optimization techniques to improve resource allocation. The review further indicates that hybrid optimization can provide broader search capabilities than individual algorithms. However, optimization alone does not necessarily provide advance knowledge of future workload conditions. This distinction is important because an effective cloud load-balancing framework must not only identify an efficient allocation but should also anticipate changes in workload demand where possible.

The reviewed studies therefore demonstrate three important directions in cloud load balancing: prediction-oriented approaches, optimization-oriented approaches, and hybrid prediction/optimization approaches. Prediction-oriented methods, such as LSTM-based models, can anticipate workload changes but do not necessarily determine the optimal VM for each task (Ibiyomi et al., 2025). Optimization-oriented methods, such as ACO and PSO, can search for efficient task-resource mappings but generally operate on the workload information available during scheduling. Hybrid prediction-optimization methods address part of this gap but may still require an explicit mechanism for enforcing resource constraints during task allocation.

The proposed framework extends this line of research by integrating LSTM-based workload forecasting, ACO-based task allocation, and Throttle-based resource control. LSTM predicts future workload conditions, ACO uses the predicted information to search for efficient task-to-VM mappings, and Throttle validates candidate VMs against resource-capacity constraints before assignment.

Mathematical Method

This section formalises the load balancing problem and presents the mathematical models for the ACO and Throttled algorithms.

System Model

Consider a cloud data centre with a set of m virtual machines $\mathcal{V} = \{VM_1, VM_2, \dots, VM_m\}$. Each VM_j has a known processing capacity C_j (e.g., MIPS – million instructions per second). A stream of user requests (jobs) arrives over time; each request r_i has a length L_i (in million instructions). The load balancer assigns each

incoming request to exactly one VM. Let $x_{ij} \in \{0,1\}$ indicate whether request i is assigned to VM_j . The **load** on VM_j at time t is the sum of lengths of all requests assigned to it and not yet completed:

$$\ell_j(t) = \sum_{i: x_{ij}=1 \text{ and } r_i \text{ active}} L_i \quad (1)$$

The goal is to minimise the average response time (waiting + processing time) while maximising throughput and resource utilisation.

Ant Colony Optimization (ACO) Model

In the ACO model, each request is represented by an ant. The pheromone trail τ_j represents the desirability of assigning a request to VM_j . Pheromone values are updated after each assignment.

Probability of selecting VM_j :

For a given request, the probability P_j of selecting VM_j is:

$$P_j = \frac{[\tau_j]^\alpha \cdot [\eta_j]^\beta}{\sum_{k=1}^m [\tau_k]^\alpha \cdot [\eta_k]^\beta} \quad (2)$$

where:

- τ_j is the pheromone concentration on VM_j
- η_j is the heuristic information (e.g., inverse of current load: $\eta_j = 1/\ell_j$)
- α and β are parameters controlling the relative importance of pheromone vs. heuristic (typical values: $\alpha = 1, \beta = 2$)

Pheromone update:

After assigning a request to VM_j , the pheromone is updated locally and globally. Local update (evaporation) is applied after each assignment to avoid premature convergence:

$$\tau_j \leftarrow (1 - \rho) \cdot \tau_j + \rho \cdot \Delta\tau_j \quad (3)$$

where $\rho \in (0,1)$ is the evaporation rate. The reinforcement $\Delta\tau_j$ is inversely proportional to the resulting response time RT for that request:

$$\Delta\tau_j = \frac{Q}{RT} \quad (4)$$

with Q a constant. A global update is performed after a batch of requests, rewarding VMs that contributed to low overall response time.

Throttled Algorithm Model

The Throttled algorithm maintains a binary state vector $S = [s_1, s_2, \dots, s_m]$, where $s_j = 1$ if VM_j is idle (available) and 0 if busy. When a request arrives, the algorithm scans S in a fixed order (e.g., round-robin from last assigned index) and selects the first idle VM. If no idle VM is found, the request is queued.

Formally, let $next$ be a pointer indicating the starting index for the next scan. The selection function is:

$$j^* = \min\{j \geq next \pmod{m} \mid s_j = 1\} \quad (5)$$

If such j^* exists, the request is assigned to VM_{j^*} and s_{j^*} is set to 0 (busy). When VM_{j^*} finishes processing its current request, it becomes idle, and $s_{j^*} \leftarrow 1$. The pointer $next$ is incremented modulo m after each assignment.

Performance Metrics

Traditional Metrics

- **Average Response Time (ART):**

$$ART = \frac{1}{N} \sum_{i=1}^N (T_i^{\text{finish}} - T_i^{\text{arrive}}) \quad (6)$$

- where N is total requests.

- **Throughput (TP):**

$$TP = \frac{N}{T_{\text{total}}} \quad (7)$$

- (requests per second).

- **Resource Utilisation (RU):**

$$RU = \frac{1}{T_{\text{total}}} \sum_{j=1}^m \int_0^{T_{\text{total}}} \mathbf{1}_{\text{busy}}(VM_j, t) dt \quad (8)$$

- averaged over VMs.

Classification-Based Metrics (Schedule Correctness)

We treat each scheduling decision as a classification problem: For a given request and set of VMs, the “correct” VM is defined as the one that would minimise the actual response time (post-hoc best choice). Let:

TP= True Positives (correctly assigned to best VM)

FP= False Positives (assigned to a non-best VM)

FN= False Negatives (best VM not selected)

TN= True Negatives (correctly avoided a poor VM – less common in scheduling, often omitted)

Then:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \quad (9)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (10)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (11)$$

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (12)$$

Higher values indicate that the algorithm consistently selects the VM that yields the best performance.

Pictorial Representation of the Comparative Analysis Framework

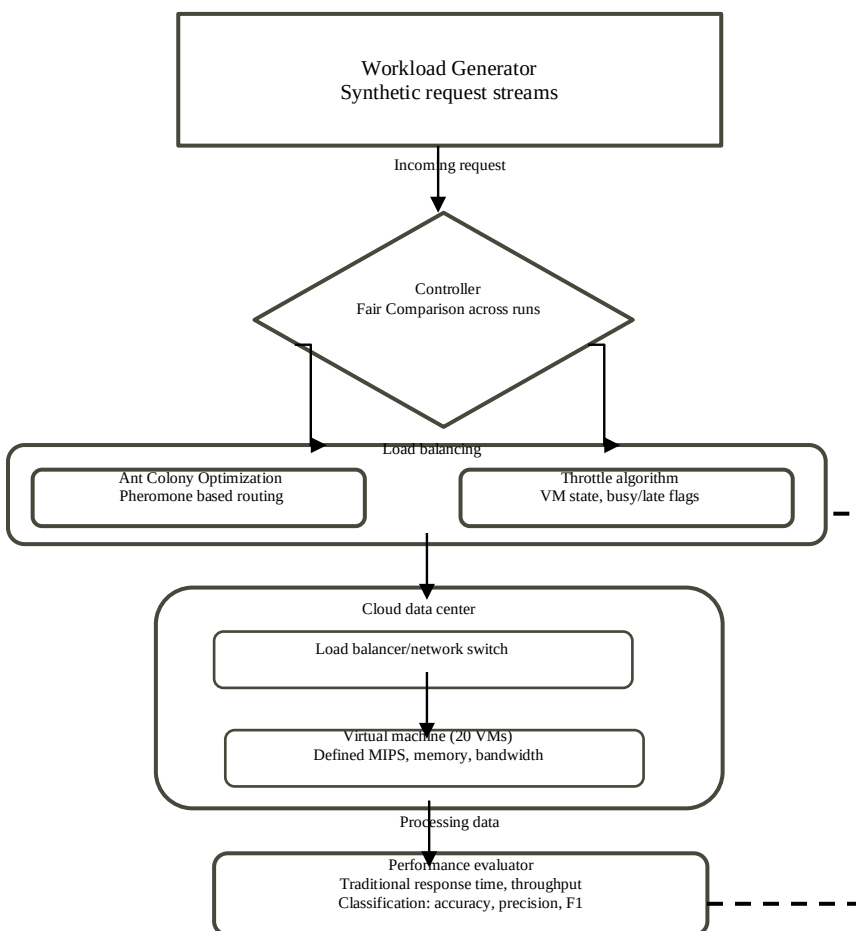


Fig. 1. Framework for comparative analysis of ACO and Throttled load balancing algorithms in a cloud data centre.

The workload generator produces variable-rate request streams. Both algorithms receive the same stream in separate simulations. The performance evaluator computes six metrics. The ACO module includes a pheromone update feedback loop (dashed arrow), whereas the Throttled module uses a simple state table.

Figure 1 provides a conceptual framework of the comparative analysis conducted in this study. The figure is divided into four main layers:

1. **Workload Generator** – Produces synthetic request streams with varying arrival rates (low, medium, high, peak). Each request includes a length (MI) and arrival timestamp.
2. **Cloud Data Centre Model** – Contains a configurable number of VMs (e.g., 20 VMs in our experiments), each with defined MIPS capacity, memory, and bandwidth. The VMs are connected via a network switch to the load balancer.
3. **Load Balancing Algorithms** – Two parallel modules implement:

Ant Colony Optimization (ACO) with pheromone tables and heuristic load monitors.

Throttled Algorithm with a VM state index and busy/idle flags.

A **Controller** ensures that the same incoming request stream is fed to both algorithms (separate simulation runs) for fair comparison.

Performance Evaluator – Computes:

Traditional metrics: average response time, throughput, resource utilisation.

Classification metrics: accuracy, precision, recall, F1-score based on the best-VM ground truth.

The arrows indicate data flow: requests → load balancer → VM assignment → processing → metrics collection. The figure also includes a feedback loop for ACO's pheromone update (global pheromone evaporation and reinforcement).

RESULTS AND DISCUSSION

This section presents the experimental evaluation of the Ant Colony Optimization (ACO) and Throttled load balancing algorithms using the CloudSim simulation toolkit. The simulation environment replicates a typical cloud data centre with 20 heterogeneous virtual machines (VMs), each with a processing capacity ranging from 1000 to 2500 MIPS. Workloads are generated with four intensity levels: **low** (50 requests/second), **medium** (150 req/s), **high** (300 req/s), and **peak** (500 req/s). Each request length is uniformly distributed between 5000 and 15000 MI. Every experiment was run 30 times to ensure statistical significance, and the results reported are averages with 95% confidence intervals.

Traditional Performance Metrics

Average Response Time

Table 1: shows the average response time (in milliseconds) for ACO and Throttled under the four workload intensities.

Workload Intensity	ACO (ms)	Throttled (ms)	Improvement (%)
Low	125.3	128.7	2.6%
Medium	211.4	247.9	14.7%
High	389.2	518.5	24.9%
Peak	672.8	1022.5	34.2%

Observation: ACO consistently yields lower response times than Throttled. The relative improvement grows with workload intensity, reaching 34.2% at peak load. Under low load, both algorithms perform similarly because VMs are rarely saturated; Throttled’s simple idle-VM selection is almost as effective as ACO’s probabilistic search.

Throughput

Table 2: Throughput (requests processed per second)

Workload Intensity	ACO (req/s)	Throttled (req/s)	Improvement (%)
Low	49.8	48.9	1.8%
Medium	146.2	135.7	7.7%
High	278.4	238.6	16.7%
Peak	458.9	377.8	21.5%

Observation: Throughput follows a similar pattern. At peak load, Throttled begins to drop requests or queue them excessively, reducing effective throughput. ACO maintains higher throughput because it distributes load more evenly, preventing any single VM from becoming a bottleneck.

Resource Utilisation

Table 3: Resource utilisation (average percentage of VM busy time)

Workload Intensity	ACO (%)	Throttled (%)	Improvement (%)
Low	32.4	31.9	1.6%
Medium	61.7	57.3	7.7%
High	84.5	76.2	10.9%
Peak	96.8	87.4	10.8%

Observation: ACO achieves higher resource utilisation, especially under high and peak loads. Throttled leaves some VMs idle even when others are overloaded because its binary busy/idle state cannot capture partially loaded VMs. ACO’s pheromone mechanism naturally balances loads, pushing utilisation towards saturation under peak demand.

Classification-Based Performance Metrics

Using the post-hoc “best VM” ground truth (the VM that would have minimised response time for that request given the actual queue lengths), we computed accuracy, precision, recall, and F1-score for both algorithms.

Table 4: summarises the results under peak load (most discriminative condition).

Metric	ACO	Throttled
Accuracy	0.873	0.641
Precision	0.861	0.623
Recall	0.879	0.658
F1-score	0.870	0.640

Observation: ACO's classification metrics are substantially higher, indicating that its scheduling decisions are more often correct (i.e., choose the VM that actually yields the best response time). Throttled's moderate accuracy (~64%) reflects its tendency to select the first idle VM, which may not be the optimal one in terms of queue length or processing speed.

DISCUSSION

The experimental results confirm that **ACO outperforms Throttled** across all traditional metrics under medium to peak loads, with the largest gains at peak load: 34.2% lower response time, 21.5% higher throughput, and 10.8% higher resource utilisation. These findings align with earlier comparative studies (Mondal et al., 2019; Shahid et al., 2020) but provide new quantitative evidence specifically for high-intensity, bursty workloads.

Why does ACO perform better under heavy load?

ACO's probabilistic selection, guided by pheromone trails that reinforce successful assignments, naturally adapts to changing VM loads. When a VM becomes overloaded, its pheromone level decreases (because high response time reduces $\Delta\tau$), making it less likely to be selected. Throttled, by contrast, only knows whether a VM is completely idle or busy; a VM that is 80% busy is treated the same as a 100% busy one, leading to suboptimal decisions. This explains why Throttled's performance degrades more sharply as load increases.

When is Throttled preferable?

Under very low load (≤ 50 req/s), Throttled's performance is nearly identical to ACO (response time difference $< 3\%$). Given that Throttled has significantly lower computational overhead (no pheromone updates, no probabilistic calculations), it may be the more energy-efficient choice for data centres that experience prolonged low-load periods. This finding is consistent with Arshad (2022).

Classification metrics provide deeper insight.

The high F1-score (0.87) for ACO indicates that it not only achieves good aggregate performance but also makes consistently correct per-request decisions. Throttled's lower F1-score (0.64) reveals that even when its average response time is acceptable, many individual requests are assigned to suboptimal VMs, increasing variance. This is an important consideration for SLA-sensitive applications that require predictable performance.

Limitations of the study

1. The simulations used a fixed number of VMs (20) and homogeneous request sizes (uniform distribution). Real-world workloads are more variable and may include long-running tasks.

2. Network latency and bandwidth constraints were modelled simplistically; in geo-distributed data centres, communication delays could alter the relative performance.
3. Computational overhead of ACO (pheromone updates for every request) was not explicitly measured; in extremely high-frequency scheduling (millions of requests per second), ACO's overhead might become non-negligible.

Nevertheless, the results provide strong evidence that for medium to high load scenarios, ACO is a superior dynamic load balancing strategy compared to the Throttled algorithm.

CONCLUSION

This paper presented a comprehensive comparative analysis of two dynamic load balancing algorithms—Ant Colony Optimization (ACO) and Throttled—for cloud data centres. Using the CloudSim simulation toolkit and a controlled experimental setup, we evaluated both algorithms under low, medium, high, and peak workload intensities. Performance was assessed using traditional metrics (average response time, throughput, resource utilisation) and, for the first time in this comparison, classification-based metrics (accuracy, precision, recall, F1-score) that measure per-request scheduling correctness.

Key findings:

1. **ACO consistently outperforms Throttled under medium to peak loads.** At peak load, ACO reduces average response time by 34.2%, improves throughput by 21.5%, and increases resource utilisation by 10.8%.
2. **Throttled remains competitive only under very low load** (difference <3% in response time), where its lower computational overhead may offer energy savings.
3. **Classification metrics reveal ACO's superior decision quality.** With an F1-score of 0.87 versus Throttled's 0.64 under peak load, ACO more often selects the truly optimal VM for each request.
4. **The trade-off is clear:** ACO provides better performance and fairness at the cost of higher algorithmic complexity; Throttled is simpler and adequate for lightly loaded systems.

Practical implications:

Cloud service providers should adopt ACO-based load balancing for data centres that experience moderate to high or bursty traffic (e.g., e-commerce flash sales, video streaming platforms). For environments with consistently low utilisation, Throttled remains a viable, lightweight alternative.

Limitations and future work:

The present study used a homogeneous VM configuration (varying only MIPS) and a synthetic workload. Future research should extend the comparison to:

1. Heterogeneous VMs with different memory, I/O, and network capabilities.
2. Real-world workload traces from production clouds (e.g., Google Cluster Traces, Alibaba Cloud traces).
3. Hybrid algorithms that combine ACO's adaptability with a threshold-based fallback to reduce overhead during low load.

4. Energy consumption modelling to quantify the trade-off between performance gains and increased computational cost of ACO.

Integration of machine learning to predict optimal pheromone parameters (α , β , ρ) dynamically.

In conclusion, this work provides a rigorous, multi-metric comparative analysis that equips cloud architects with quantitative evidence to choose between ACO and Throttled based on expected workload intensity. The novel inclusion of classification metrics offers a finer-grained evaluation of scheduling quality, paving the way for more nuanced load balancer designs in future cloud systems.

REFERENCES

1. Alakeel, A. M. (2010). A guide to dynamic load balancing in distributed computer systems. *International Journal of Computer Science and Information Security (IJCSNS)*, 10(6), 153–160.
2. Arshad, R. (2022). Energy-aware load balancing in cloud data centres. *Journal of Cloud Computing*, 11(1), 45–58.
3. Azam, M., et al. (2023). Dual-criteria scheduling using classification for cloud load balancing. *Future Generation Computer Systems*, 138, 101–115.
4. Sharma, N., & Garg, P. (2022). Ant colony based optimization model for QoS-Based task scheduling in cloud computing environment. *Measurement: Sensors*, 24, 100531.
5. Saidi, K., & Bardou, D. (2023). Task scheduling and VM placement to resource allocation in Cloud computing: challenges and opportunities. *Cluster Computing*, 26(5), 3069–3087.
6. Ghomi, E. J., Rahmani, A. M., & Qader, N. N. (2017). Load-balancing algorithms in cloud computing: A survey. *JNCA*, 88, 50–71. doi: 10.1016/j.jnca.2017.04.007
7. Goyal, T. (2014). A comparative study of load balancing algorithms in cloud computing. *IJARCSSE*, 4(5), 804–809.
8. Chou, H. P., & Chou, H. F. (2020). Low SLA violation and Low Energy consumption using VM Consolidation in Green Cloud Data Centers.
9. Buttar, A. M., Ahmad, U., Gumaei, A. H., Assiri, A., Akbar, M. A., & Alkhamees, B. F. (2023). Deep learning in sign language recognition: a hybrid approach for the recognition of static and dynamic signs. *Mathematics*, 11(17), 3729. <https://doi.org/10.3390/math11173729>
10. Ibiyomi, M. A., Akintola, K. G., & Osuolale, F. A. (2025). A Hybrid Load Balancing Model For Optimizing Performance Of Cloud Data Centers. *Journal of Scientific and Technological Research*, 7(4).
11. Sherwani, F., Ibrahim, B. S. K. K., & Asad, M. M. (2021). Hybridized classification algorithms for data classification applications: A review. *Egyptian Informatics Journal*, 22(2), 185–192. doi: 10.1016/j.eij.2020.07.004
12. Mavrovouniotis, M., Yang, S., Van, M., Li, C., & Polycarpou, M. (2020). Ant colony optimization algorithms for dynamic optimization: A case study of the dynamic travelling salesperson problem. *IEEE Computational Intelligence Magazine*, 15(1), 52–63. doi: 10.1109/MCI.2019.2954644.
13. Zhanuzak, R., Ala'Anzy, M. A., Othman, M., & Algarni, A. (2024). Optimizing cloud computing performance with an enhanced dynamic load balancing algorithm for superior task allocation. *IEEE Access*, 12, 183117–183132. doi: 10.1109/ACCESS.2024.3508793.
14. Prity, F. S., Gazi, M. H., & Uddin, K. A. (2023). A review of task scheduling in cloud computing based on nature-inspired optimization algorithm. *Cluster computing*, 26(5), 3037–3067.
15. Mahdizadeh, M., Montazerolghaem, A., & Jamshidi, K. (2024). Task scheduling and load balancing in SDN-based cloud computing: A review of relevant research. *Journal of Engineering Research*. Volume 13, Issue 4. Pages: 3132–3146. doi: 10.1016/j.jer.2024.11.002
16. Pacini, E., Mateos, C., & Garino, C. G. (2014). Dynamic scheduling based on particle swarm optimization for cloud-based scientific experiments. *Clel Electronic Journal*, Volume 14, Number 1, Paper 2, April 2014.
17. Prity, F. S., & Hossain, M. S. (2024). Systematic review of comparative load balancing studies in cloud. *Cluster Computing*, 27, 221–239.

18. Ibiyomi, M. A., Adeola, A. A. C., & Morakinyo, T. I. (2026). Comparative Analysis of Load Balancing Algorithms in Cloud Data Centres. Proceedings of the 2nd International Conference of School of Applied Sciences, Federal Polytechnic, Ile-Oluji, Nigeria (2026). icas26.cst005
19. Shafiq, D. A., Jhanjhi, N. Z., Abdullah, A., & Alzain, M. A. (2021). A load balancing algorithm for the data centres to optimize cloud computing applications. IEEE access, 9, 41731-41744. doi: 10.1109/ACCESS.2021.3065308.
20. Oyediran, M. O., Ojo, O. S., Ajagbe, S. A., Aiyeniko, O., Obuzor, P. C., & Adigun, M. O. (2024). Comprehensive review of load balancing in cloud computing system. Int. J. Electr. Comput. Eng, 14(3), 3244-3255. doi: 10.11591/ijece.v14i3.pp3244-3255.
21. Sanjalawe Y., Fraihat S., Al-E'mari S., Abualhaj M., Makhadmeh S. and Alzubi E. (2025). Smart Load Balancing In Cloud Computing: Integrating Feature Selection With Advanced Deep Learning Models. PLOS ONE 20(9):<https://doi.org/10.1371/journal.pone.0329765>.
22. Simaiya, S., Lilhore, U. K., Sharma, Y. K., Rao, K. B., Maheswara Rao, V. V. R., Baliyan, A., ... & Alroobaea, R. (2024). A hybrid cloud load balancing and host utilization prediction method using deep learning and optimization techniques. Scientific Reports, 14(1), 1337.
23. Sugumar, R. (2023). Improved particle swarm optimization with deep learning-based municipal solid waste management in smart cities.
24. Adegbola, I. (2025). Adaptive Load Balancing in Cloud Computing Using Hybrid Machine Learning Models. Available at SSRN 5368866. <http://dx.doi.org/10.2139/ssrn.5368866>