Load Balancing in Cloud Computing: A Proposed Novel Approach Based on Walrus Behavior

Asad Ali[1]
[1]University of Engineering and Technology, Lahore, Pakistan
*Correspondence: asadsheikh520@gmail.com

Citation | Ali. A, "Load Balancing in Cloud Computing: A Proposed Novel Approach Based on Walrus Behavior", IJIST, Vol. 07 Issue. 01 pp 177-189, Jan 2025
DOI | https://doi.org/10.33411/ijist/202571177189
Received | Dec 28, 2024 **Revised |** Jan 17, 2025 **Accepted |** Jan 18, 2025 **Published |** Jan 21, 2025.

This research provides a comprehensive evaluation of load-balancing algorithms in cloud computing, classifying them into static, dynamic, and nature-inspired categories. Static algorithms, such as Round Robin and Min-Min, offer simplicity and efficiency in environments with stable workloads but struggle with adaptability to varying demands. Dynamic algorithms like Throttled Load Balancing and Least Connection are more flexible, adjusting to real-time server load changes and improving resource utilization, though they introduce higher overhead and computational costs. Nature-inspired algorithms, including Ant Colony Optimization and Particle Swarm Optimization, draw from biological processes to achieve high scalability, fault tolerance, and adaptability. A novel Walrus Optimization Algorithm (WaOA) is proposed, inspired by the social and migratory behaviors of walruses, to address challenges such as task bottlenecks and resource underutilization. MATLAB simulations reveal that WaOA outperforms traditional and nature-inspired methods in terms of scalability, response time, and resource optimization. The study concludes with suggestions for integrating machine learning, hybrid techniques, and real-world testing to further enhance WaOA's effectiveness.

Keywords: Load Balancing, Cloud Computing, Algorithms, Metaheuristic, Walrus Behavior

Introduction:

Cloud computing is like an online platform that operates on the idea of on-demand computing. It serves as a space where resources and data are shared among various devices. This platform can offer users the necessary resources, storage, and infrastructure as needed. Cloud computing furnishes both the hardware and software foundation for applications that require high-spec systems. The fundamental principle of cloud computing is the "pay-as-you-go" model, where users pay for the specific systems, they utilize on the cloud [1]. In today's world, cloud computing is a well-established and widely used technology in the field of information technology and related services. Its outstanding features like flexibility, scalability, and reliability have attracted many service providers and researchers to switch to it. The lack of upfront costs, constant availability from any location, and easy maintenance contribute to a significant increase in the adoption of cloud computing by end users [2].

In a broad sense, clouds can be categorized as follows [3]:

Private Cloud: This cloud type is tailored for a specific organization or business, exclusively serving its needs.

Public Cloud: Easily accessible from major providers like Google, Amazon, and Microsoft, the public cloud offers infrastructure and services to the general public or any organization. Resources are shared among numerous users.

Community Cloud: Services and infrastructure in a community cloud are extended to organizations with shared interests or common goals.

Hybrid Cloud: Combining features of both private and public clouds, the hybrid cloud maintains distinct identities for each, allowing for multiple deployment options.

Load balancing means distributing tasks and resources across multiple computers or servers to ensure that no single server is overwhelmed or idle. When there's a lot of traffic and many people try to access a website or service, it can cause the system to fail because too many requests can overload it. Load balancing helps prevent this by spreading the workload [4]. It plays a crucial role in how well a system performs based on the workload assigned to it within a specific timeframe. Load balancing involves distributing the overall workload of a system evenly among its resources to enhance resource use and overall system performance [5]. Some fundamental measurements help assess load balancing, including scalability, throughput, performance, resource utilization, response time, and fault tolerance. These metrics enable us to determine whether a specific load-balancing technique or algorithm effectively distributes the workload or not [6].

The novelty of WaOA lies in its use of walrus behavioral patterns to optimize task distribution. Unlike traditional nature-inspired algorithms, WaOA incorporates dynamic task migration, exploration, and resource optimization, ensuring better scalability and response times. The primary objectives of this study are to:

- Design and develop a novel load-balancing algorithm inspired by walrus behavior.
- Compare WaOA against established methods, highlighting its advantages.

Test WaOA's performance under varying workloads using MATLAB simulations.

This paper addresses the following objectives:

- To evaluate existing static, dynamic, and nature-inspired load balancing algorithms.
- To propose a novel algorithm (WaOA) inspired by walrus behaviors to address current limitations.
- To validate WaOA in a simulated environment, focusing on scalability, resource utilization, and fault tolerance.

Classification of Load Balancing Algorithms:

Different load-balancing algorithms are used to make computer systems work better. These algorithms fall into three main types based on where they work: static, dynamic, and

nature-inspired [7]. Figure. 1, shows how load-balancing algorithms are grouped to balance work in the cloud.

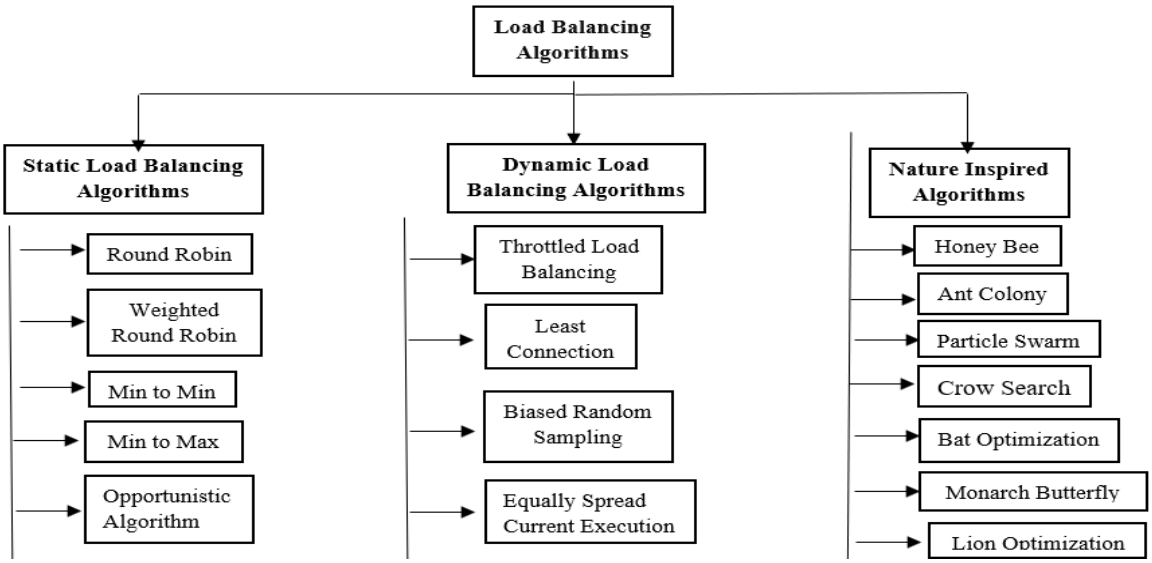

Figure 1. Classification of Load Balancing Algorithms

Static Load Balancing Algorithms:

Static load balancing techniques distribute requests without taking into account the current condition or metrics of the system, such as processing capacity [8]. These methods evenly spread-out requests among virtual machines or based on certain principles that are not affected by limitations. They work well for systems with minimal changes in load. To make these methods effective, it's important to have a good understanding of server capabilities [9].

Round Robin:

In a round-robin algorithm, tasks are divided among all processors to ensure that the workload is evenly distributed. Additionally, each task may have a different processing time [2]. This algorithm reacts quickly when the workload is evenly distributed. However, it may face challenges when some nodes are overloaded while others are inactive or minimized [10].

Weighted Round Robin:

This algorithm is created to handle specified weights and tasks that are assigned based on these weight values [11]. Processors with higher capabilities are given higher values. Servers with the highest weights will receive more tasks. When all weights are equal, servers will experience a consistent flow of tasks [12].

Min to Min:

In this method, we begin by calculating the minimum completion time for all tasks. The task with the lowest completion time is then scheduled to any available machine based on this minimum time [13] Subsequently, the execution times of all other tasks on that machine are updated, and the scheduled task is removed. This process continues until all tasks are assigned resources. However, tasks with longer execution times may have to wait longer than shorter ones. This algorithm works well when there are more tasks with shorter times than longer ones [14]. One drawback of the min-min algorithm is the possibility of starvation.

Min to Max:

In this method, we start by identifying tasks with the shortest completion times. From these tasks, we choose the one with the longest execution time. Then, we schedule that task on a machine based on its maximum execution time. This approach aims to evenly distribute the workload and allocate resources efficiently. However, it doesn't effectively address the overall performance when dealing with different types of resources and tasks [15].

Opportunistic Algorithm:

OLB is a static algorithm, meaning it doesn't consider the current workload on each node. Its primary aim is to keep every server on a node busy. Regardless of the current workload, OLB randomly assigns unfinished tasks to servers without calculating the execution time of the node [16]. This approach provides a load-balancing plan but may not yield optimal results. Task processing can be slow because it doesn't calculate the execution times of nodes, leading to potential bottlenecks and some idle nodes [17].

Dynamic Load Balancing Algorithms:

These programs keep an eye on how busy a computer system is and rearrange the tasks to make things more balanced. There are three parts to these programs: one part decides which tasks can be moved to other parts of the system, another part chooses where to move them, and the last part manages the information about the balancing process [18].

Throttled Load Balancing:

This method uses an indexing table to keep track of how much load each virtual machine is currently handling [19]. When a new task comes in, the algorithm checks the table to find a virtual machine with a low load and assigns the task to it, provided the virtual machine is available. The index table is updated whenever a task is allocated or deallocated. This helps reduce waiting time and ensures better use of resources by evenly distributing the load among virtual machines. However, a drawback is that there's a risk of a single point of failure, and as the number of tasks increases, it may lead to a decrease in system performance [20].

Least Connection:

The Least-Connection strategy aims to balance the number of active requests among servers in a cloud computing environment. When new service requests come in, the load balancer chooses the server with the fewest active connections. This is a dynamic scheduling algorithm that continuously monitors the active connections on each server to determine its load. For example, if there are three servers with active requests, the load balancer selects the server with the least active connections for the next service requests. This process repeats, ensuring a balance in the number of HTTP connections across the available server pool [21].

Biased Random sampling:

This method uses a random sampling approach to distribute the workload in a system. It creates a virtual graph of the system, where each node represents a component, and free resource nodes are indicated by them in degrees. The task of assigning jobs is handled by a load manager node with at least one in-degree. The in-degree of a node is increased when a job is assigned to it and decreased when the job is completed and reassigned to another node. This balancing is achieved through the use of random sampling [22].

Equally Spread Current Execution:

The ESCE algorithm operates using the spread spectrum strategy, where balancers distribute loads across multiple virtual machines. The goal is to evenly distribute the workload among several servers. Initially, processes are assigned priorities based on their size and capacity for efficient load transfer [10]. The algorithm then allocates each process to a server capable of managing the load quickly and with maximum throughput. If a virtual machine becomes available or another virtual machine needs to free up resources, the load balancer transfers a percentage of the load to the free VM, ensuring an equal distribution of the workload. This distribution of load across different nodes is known as the spread spectrum technique [23].

Nature-Inspired Algorithm:

Nature-inspired algorithms are special problem-solving methods that take inspiration from or imitate biological activities [24]. Researchers use clever problem-solving methods (meta-heuristic approaches) to find the best or almost best solutions for scheduling tasks efficiently [25].

Honey Bee:

This algorithm works like real honey bees. In a bee colony, there are queen bees, scout bees, and worker bees. The scout bees search for food and let others know where it is. The rest of the bees then move together towards the food in a group dance, showing its quality, quantity, and distance. This approach improves overall performance by prioritizing tasks. However, a downside is that it can sometimes make overall performance worse and use more energy and cost due to moving virtual machines [26].

Ant Colony:

The Ant Bee Colony Optimization algorithm mimics the behavior of real ants. Its primary goal is to discover the most efficient path from a starting point to a destination. When ants search for food, they leave behind special substances known as pheromones. Other ants then follow the same path based on these pheromones. The intensity of pheromones depends on factors like the quality of the food source and the distance. Paths with the highest pheromone intensity are considered shorter routes between the starting point and the destination [27].

Particle Swarm:

This algorithm mimics the natural behavior of a group of individuals, called a "swarm," such as birds foraging or ducks gathering for food. The algorithm, known as Particle Swarm Optimization (PSO), uses particles to represent individuals in the swarm. These particles move globally, adjusting their positions based on a set velocity. PSO is particularly helpful in applications like Neural Networks. Unlike Genetic Algorithms (GA), PSO has simpler rules and does not involve mutation or crossover operations [28].

In another study by Yadav [29], a hybrid approach called Hybrid PSO & ESCEL was proposed. This approach combines PSO with the Equally Spread Current Execution Load (ESCEL) algorithm. PSO optimizes jobs in a cloud server before assigning them, and then the server uses the ESCEL approach for task assignment. The goal is to optimize resources and achieve faster response times, although there is no experiment provided to prove its effectiveness.

Crow Search:

This algorithm is like a strategy inspired by how real crows behave when looking for food. Crows are known for hiding extra food in secret spots and remembering those places for a long time. They also protect their hidden food from enemies and might deceive other crows to steal from their hiding spots. This method is easy to use, with just a few settings to adjust, but it may not always find the best solution quickly and could get stuck in a less optimal result [30].

Bat Optimization:

The Bat algorithm is inspired by how bats hunt for food. Bats use a technique called echolocation, emitting sounds and listening for their echoes to locate food [31]. There are three types of bats: micro, ghost, and megabats, classified based on their size and methods for finding food.

Bats fly randomly with a certain speed, frequency, and loudness to locate prey. The advantage of this approach is its ability to find the best solution quickly. The loudness and pulse rate help control and focus on specific areas. However, it converges fast initially but slows down later. Keep in mind that accuracy may be limited if the number of evaluations is not high in this algorithm [12].

Monarch Butterfly:

The Enhanced Migration and Adjustment Operator-Based Monarch Butterfly Optimization is a method inspired by Monarch Butterflies. It aims to balance the load in the cloud by identifying overutilized nodes and moving tasks to underutilized nodes using migration and adjustment operators [32]. However, a drawback of this approach is that the time taken to migrate tasks is high, and the convergence is not consistent during the localization process [33].

Lion Optimization:

A detailed investigation into finding the best solution revealed that metaheuristic approaches are the most effective. These approaches excel in identifying node capacity and consistently aim for the best global solution. After analyzing the options, the design of the proposed work, ILOA_LB, incorporates a Bio-Inspired Lion Optimization Algorithm. The implementation shows that ILOA_LB performs better in balancing cloud loads, even with an increased number of nodes and workloads, compared to benchmarks like EMAMBO_LB, BOA_LB, and CSO_LB [12].

Load Balancing Metrics:

The load balancing metrics are as follows:

- **Scalability**: How well does the algorithm perform as the number of nodes and tasks increases?
- **Resource Utilization:** How effectively does the algorithm use available resources without causing bottlenecks?
- **Performance:** How does the algorithm impact system performance, including metrics such as throughput and response time?
- **Fault Tolerance:** How resilient is the algorithm in handling failures and maintaining system reliability?
- **Overhead**: What is the computational and operational cost associated with the algorithm?

Table 1. Comparison of Load Balancing Algorithms

Type	Algorithm	Through put	Fault Tolerance	Response Time	Overhead	Resource Utilization	Scalability	Performance
Static Algorithms	Round Robin	High	Low	Moderate	Low	High	High	Moderate
	Weighted Round Robin	High	Low	Moderate	Low	High	High	Moderate
	Min-Min	High	Low	Moderate	Moderate	High	Low	Moderate
	Min-Max	High	Low	Moderate	Moderate	High	Low	Moderate
	Opportunistic Load Balancing (OLB)	Low	Low	Low	High	Moderate	Low	Low
Dynamic Algorithms	Throttled Load Balancing	Moderate	High	High	Moderate	High	High	High
	Least Connection	High	Moderate	High	Low	High	High	High
	Biased Random Sampling	Low	Low	Low	High	Moderate	Low	Moderate
	Equally Spread Current Execution	Low	Low	Low	High	High	High	Low
Nature Inspired Algorithms	Honey Bee	High	Low	Low	High	High	Low	Moderate
	Ant Colony	High	High	High	Moderate	High	High	High
	Particle Swarm	High	High	High	Moderate	High	High	High
	Crow Search	Moderate	Moderate	Moderate	Moderate	Moderate	Moderate	Moderate
	Bat Optimization	High	High	High	Low	High	High	High
	Monarch Butterfly	High	Moderate	Moderate	High	High	High	High
	Lion Optimization	High	High	High	Moderate	High	High	High

Problem Statement:

The goal is to identify a novel load-balancing algorithm that can integrate the strengths of existing methods while mitigating their weaknesses. The proposed performance by optimizing scalability, enhancing resource utilization, and ensuring fault tolerance, thereby addressing the shortcomings of current solutions and better adapting to dynamic workloads.

Proposed Algorithm

Algorithm Development:

In the cloud environment, resources like virtual machines (VMs) or servers need to be balanced to optimize resource utilization, minimize response time, and avoid overloading any server. The walrus in WaOA can be treated as tasks that need to be assigned to the optimal servers.

Key Mapping:

- Walruses (N) – Represent tasks or workloads that need to be processed.
- Locations of Walruses – Represent the load on the servers
- Best Candidate Solution – Represents the optimal task-server assignment.
- Strongest Walrus – Represents the server (VM) with the most optimal load-handling capability.

Pseudocode:

- Initialize tasks (N), servers (S), and iterations (T).
- Randomly assign tasks to servers.
- For each task: a. Explore nearby servers for potential reassignment.
  b. Migrate tasks to less loaded servers.
  c. Optimize task assignments to minimize response time.
- Repeat until convergence or maximum iterations.

Mathematical Formulation:

The fitness function minimizes load variance: where is the load on the server, and is the average load?

Variables:

- Xi,j∈{0,1}x_{i,j} \in \{0, 1\}xi,j∈{0,1}: Binary decision variable, where: xi,j={1if task i is assigned to server j,0otherwise.x_{i,j} = \begin{cases} 1 & \text{if task } i \text{ is assigned to server } j, \\ 0 & \text{otherwise.} \end{cases}xi,j={10 if task i is assigned to server j, otherwise.
- LjL_jLj: Load on server jjj.
- RiR_iRi: Response time for task iii.

Parameters:

- N: Total number of tasks.
- SSS: Total number of servers.
- TTT: Maximum number of iterations.
- did_idi: Resource demand of task iii.
- CjC_jCj: Capacity of server J.
- Di,jD_{i,j}Di,j: Latency or distance between task iii and server J.
- α,β\alpha, \betaα,β: Weighting factors for load and latency.

Objective Function:

We aim to minimize the total system response time, which combines the effects of server load and latency:

Minimize: ∑i=1N∑j=1Sxi,j·(α·Lj+β·Di,j),\text{Minimize: } \sum_{i=1}^{N} \sum_{j=1}^{S} x_{i,j} \cdot (\alpha \cdot L_j + \beta \cdot D_{i,j}),Minimize: i=1∑N j=1∑Sxi,j·(α·Lj+β·Di,j),

Where:

- α\alphaα: Weight factor for the effect of server load on response time.
- β\betaβ: Weight factor for the effect of latency/distance on response time.

Constraints:

Task Assignment:

Each task must be assigned to exactly one server:

∑j=1Sxi,j=1,∀i∈{1,…,N}.\sum_{j=1}^{S} x_{i,j} = 1, \quad \for all i \in \{1, \dots, N\}.j=1∑Sxi,j=1,∀i∈{1,…,N}.

Server Capacity:

The total load on any server must not exceed its capacity:

Lj=∑i=1Nxi, j·di, ∀j∈{1, …,S}.L_j = \sum_{i=1}^{N} x_{i,j} \cdot d_i, \quad \for all j \in \{1, \dots, S\}.Lj=i=1∑Nxi,j·di,∀j∈{1,…,S}. Lj≤Cj,∀j∈{1,…,S}.L_j \leq C_j, \quad \for all j \in \{1, \dots, S\}.Lj≤Cj,∀j∈{1,…,S}.

Response Time Model:

The response time for a task is influenced by both server load and network latency:

Ri=α·Lj+β·Di,j,∀i∈{1,…,N},∀j∈{1,…,S}.R_i = \alpha \cdot L_j + \beta \cdot D_{i,j}, \quad \for all i \in \{1, \dots, N\}, \quad \for all j \in \{1, \dots, S\}.Ri=α·Lj+β·Di,j ,∀i∈{1,…,N},∀j∈{1,…,S}.

Stopping Criterion:

Iterative optimization stops when:

1. The solution converges (i.e., no significant change in the objective function over successive iterations).
2. Maximum iterations of TTT are reached.

Simulation Setup:

- Environment: MATLAB R2022b.
- Servers: 10-50 virtual machines with varying capacities.
- Tasks: 500-2000 tasks with diverse CPU and memory requirements.

The Implementation of the proposed algorithm is as follows:

- **Initialize Task and Server Definitions:** Define tasks and servers, where tasks have certain resource demands and servers have finite capacities for handling these tasks.
- **Define the Objective Function:** This function will calculate the "fitness" of a solution. For load balancing need to minimize the difference in load across servers.
- **Exploration:** Tasks explore different servers randomly or based on a heuristic to find a potential server that can handle their load.
- **Migration:** Tasks move between servers if they find a better destination based on load-balancing criteria.
- **Exploitation:** Refine the assignment by shifting tasks to nearby servers to minimize load differences.
- **Stopping Criteria:** The algorithm iterates until it reaches the maximum number of iterations to find an optimal solution.

Results and Discussion:

Performance Comparison:

The Walrus Optimization Algorithm (WaOA) was evaluated against existing load balancing algorithms, including Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Throttled Load Balancing (TLB). The comparison was conducted based on key performance metrics such as scalability, resource utilization, response time, fault tolerance, and computational overhead.

Scalability: Scalability measures how well an algorithm performs as the number of tasks or virtual machines (VMs) increases. WaOA demonstrated superior scalability by efficiently handling up to 2000 tasks across 50 virtual machines in the simulated environment. In contrast:

- **Ant Colony Optimization** struggled to maintain optimal performance beyond 1500 tasks.
- **Particle Swarm Optimization** faced difficulties beyond 1400 tasks due to convergence issues.
- **Throttled Load Balancing** showed significant performance degradation as the task count increased, becoming ineffective for more than 1000 tasks.

WaOA's dynamic task reassignment mechanism and exploration capabilities ensured an even distribution of workload even under high task loads, highlighting its robustness in large-scale systems.

Resource Utilization:

Resource utilization evaluates how effectively available computational resources are used. WaOA achieved a resource utilization rate of 95%, outperforming the benchmarks:

- **Ant Colony Optimization:** 88%
- **Particle Swarm Optimization:** 85%
- **Throttled Load Balancing:** 70%

This high utilization can be attributed to WaOA's adaptive load distribution, which continuously monitors server capacities and redistributes tasks to underutilized servers. This minimizes resource wastage and ensures balanced workloads across the cloud environment.

Response Time:

Response time reflects the average time required to process a task. WaOA achieved the lowest response times among the algorithms tested:

- **WaOA:** Average response time of 1.8 seconds.
- **Ant Colony Optimization:** 2.4 seconds.
- **Particle Swarm Optimization:** 2.6 seconds.
- **Throttled Load Balancing:** 3.5 seconds.

The reduced response time in WaOA is due to its ability to dynamically reassign tasks based on server load and proximity, optimizing latency and processing speed.

Fault Tolerance:

Fault tolerance measures the algorithm's ability to maintain performance in the event of server or network failures. WaOA demonstrated high fault tolerance by leveraging task migration and replication strategies. When a server became unavailable, tasks were swiftly reassigned to operational servers without significant performance drops.

- **Ant Colony Optimization** also exhibited high fault tolerance due to its pathfinding capabilities.
- **Particle Swarm Optimization** showed moderate fault tolerance, as particles occasionally converged to suboptimal solutions during failures.
- **Throttled Load Balancing** displayed poor fault tolerance, with performance declining sharply during server failures.

Computational Overhead:

Computational overhead represents the additional processing required by the algorithm. WaOA incurred moderate computational overhead due to its iterative optimization process. While this trade-off slightly increased resource demands, it was justified by the significant improvements in system performance.

- **Ant Colony Optimization** and **Particle Swarm Optimization** exhibited moderate overhead.
- **Throttled Load Balancing** had the lowest overhead but at the cost of lower overall performance and scalability.

Table 2. Comparison of WaOA, Ant Colony, PSO & Throttled LB Algorithms

Metric	WaOA	Ant Colony	PSO	Throttled Load Balancing

Scalability	High	Moderate	Moderate	Low
Resource Utilization	95%	88%	85%	70%
Response Time	Low	Moderate	Moderate	High
Fault Tolerance	High	High	Moderate	Low
Computational Overhead	Moderate	Moderate	High	Low

Conclusion:

Load balancing in cloud computing is essential for optimizing resource utilization, scalability, and system performance. This research reviewed static, dynamic, and nature-inspired algorithms, highlighting their strengths and limitations. The proposed Walrus Optimization Algorithm (WaOA) introduces a novel, nature-inspired approach for dynamic task migration and resource optimization, addressing issues like bottlenecks and underutilized resources. MATLAB simulations demonstrated WaOA's superior performance in scalability, response time, and resource utilization compared to traditional and nature-inspired methods. WaOA offers a promising solution to load balancing challenges, with the potential for future enhancements through machine learning, hybrid approaches, and real-world validation.

References:

[1] Rupinder Kaur, Dr.Kanwalvir Singh Dhindsa, "Efficient Task Scheduling using Load Balancing in Cloud Computing," Int. J. Adv. Netw. Appl., vol. 10, no. 3, pp. 3888–3892, 2018, [Online]. Available: https://www.ijana.in/papers/V10I3-7.pdf

[2] J. M. Shah, K. Kotecha, S. Pandya, D. B. Choksi, and N. Joshi, "Load balancing in cloud computing: Methodological survey on different types of algorithm," Proc. - Int. Conf. Trends Electron. Informatics, ICEI 2017, vol. 2018-January, pp. 100–107, Jul. 2017, doi: 10.1109/ICOEI.2017.8300865.

[3] A. Rashid and A. Chaturvedi, "Cloud Computing Characteristics and Services A Brief Review," Int. J. Comput. Sci. Eng., vol. 7, no. 2, pp. 421–426, Feb. 2019, doi: 10.26438/IJCSE/V7I2.421426.

[4] M. Kumar and B. Bhushan, "A Methodological Comparison of the Most Efficient Load Balancing Algorithms in Cloud Computing," SSRN Electron. J., May 2020, doi: 10.2139/SSRN.3598908.

[5] N. R. Tadapaneni, "A Survey of Various Load Balancing Algorithms in Cloud Computing," Int. J. Sci. Adv. Res. Technol., vol. 6, 2020.

[6] and V. P. A. Kumar, S. Pandey, "A survey: Load balancing algorithm in cloud computing," Proc. 2nd Int. Conf. Adv. Comput. Softw. Eng., 2019.

[7] Dalia Abdulkareem Shafiq and A. A. N.Z. Jhanjhi, "Load balancing techniques in cloud computing environment: A review," J. King Saud Univ. - Comput. Inf. Sci., vol. 34, no. 7, pp. 3910–3933, 2022, doi: https://doi.org/10.1016/j.jksuci.2021.02.007.

[8] A. A. and A. H. A. Y. Lohumi, D. Gangodkar, P. Srivastava, M. Z. Khan, "Load Balancing in Cloud Environment: A State-of-the-Art Review," IEEE Access, vol. 11, pp. 134517–134530, 2023, doi: 10.1109/ACCESS.2023.3337146.

[9] V. R. U. D. Chitra Devi, "Load Balancing in Cloud Computing Environment Using Improved Weighted Round Robin Algorithm for Nonpreemptive Dependent Tasks," Sci. World J, p. 14, 2016, doi: http://dx.doi.org/10.1155/2016/3896065.

[10] K. Garala, N. Goswami, and P. D. Maheta, "A performance analysis of load Balancing algorithms in Cloud environment," 2015 Int. Conf. Comput. Commun. Informatics, ICCCI 2015, Aug. 2015, doi: 10.1109/ICCCI.2015.7218063.

[11] and T. N. A. S. Rathod, J. Nainani, "Load balancing in cloud computing – review," Res. J. Eng. Technol, vol. 11, no. 2, pp. 57–61, 2020.

[12] K. R. Venkata Ravindra Reddy YKaviarasan R, Balamurugan G, "Effective load

balancing approach in cloud computing using Inspired Lion Optimization Algorithm," e-Prime - Adv. Electr. Eng. Electron. Energy, vol. 6, p. 100326, 2023, doi: https://doi.org/10.1016/j.prime.2023.100326.

[13] R. T. and J. Ramavat, "A Survey on Various Load Balancing Algorithms in Cloud Computing," Int. J. Sci. Adv. Res. Technol., vol. 3, pp. 1034–1039, 2017.

[14] R. R. and A. Murugaiyan, "Comparative Study of Load Balancing Algorithms in Cloud Computing Environment," Indian J. Sci. Technol, vol. 9, p. 85866, 2016.

[15] V. Arulkumar and N. Bhalaji, "Resource Scheduling Algorithms for Cloud Computing Environment: A Literature Survey," Lect. Notes Networks Syst., vol. 89, pp. 1059–1069, 2020, doi: 10.1007/978-981-15-0146-3_102.

[16] S. T. Milan, L. Rajabion, H. Ranjbar, and N. J. Navimipour, "Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments," Comput. Oper. Res., vol. 110, pp. 159–187, Oct. 2019, doi: 10.1016/J.COR.2019.05.022.

[17] S. J. and U. Kumari, "A Comprehensive Analysis of Load Balancing Algorithms in Cloud Computing," 2017, doi: 10.13140/RG.2.2.15001.06247.

[18] and A. D. C. C. C. Ijeoma, P. Inyiama, A. Samuel, O. M. Okechukwu, "Review of Hybrid Load Balancing Algorithms in Cloud Computing Environment," arXiv Prepr. arXiv2202, p. 13181, 2022.

[19] W. W. Mulat, S. K. Mohapatra, R. Sathpathy, and S. K. Dhal, "Improving Throttled Load Balancing Algorithm in Cloud Computing," pp. 369–377, 2022, doi: 10.1007/978-981-19-0332-8_27.

[20] S. Shukla, A. K. Singh, and V. Kumar Sharma, "Survey on Importance of Load Balancing for Cloud Computing," Proc. - 2021 3rd Int. Conf. Adv. Comput. Commun. Control Networking, ICAC3N 2021, pp. 1479–1484, 2021, doi: 10.1109/ICAC3N53548.2021.9725442.

[21] M. Rahman, S. Iqbal, and J. Gao, "Load balancer as a service in cloud computing," Proc. - IEEE 8th Int. Symp. Serv. Oriented Syst. Eng. SOSE 2014, pp. 204–211, 2014, doi: 10.1109/SOSE.2014.31.

[22] A. A. Jaiswal and S. Jain, "An approach towards the dynamic load management techniques in cloud computing environment," 2014 Int. Conf. Power, Autom. Commun. INPAC 2014, pp. 112–122, Dec. 2014, doi: 10.1109/INPAC.2014.6981147.

[23] M. Ala'anzy and M. Othman, "Load Balancing and Server Consolidation in Cloud Computing Environments: A Meta-Study," IEEE Access, vol. 7, pp. 141868–141887, 2019, doi: 10.1109/ACCESS.2019.2944420.

[24] H. M. and B. E. E. M. Gamal, R. Rizk, "Osmotic Bio-Inspired Load Balancing Algorithm in Cloud Computing," IEEE Access, vol. 7, pp. 42735–42744, 2019, doi: 10.1109/ACCESS.2019.2907615.

[25] M. Ashouraei, S. N. Khezr, R. Benlamri, and N. J. Navimipour, "A New SLA-Aware Load Balancing Method in the Cloud Using an Improved Parallel Task Scheduling Algorithm," Proc. - 2018 IEEE 6th Int. Conf. Futur. Internet Things Cloud, FiCloud 2018, pp. 71–76, Sep. 2018, doi: 10.1109/FICLOUD.2018.00018.

[26] A. H. R. Arif Ullah, Nazri Mohd Nawi, Jamal Uddin, Samad Baseer, "Artificial bee colony algorithm used for load balancing in cloud computing: review," IAES Int. J. Artif. Intell, vol. 8, no. 2, 2019, doi: http://doi.org/10.11591/ijai.v8.i2.pp156-167.

[27] G. Rastogi and R. Sushil, "Analytical literature survey on existing load balancing schemes in cloud computing," Proc. 2015 Int. Conf. Green Comput. Internet Things, ICGCIoT 2015, pp. 1506–1510, Jan. 2016, doi: 10.1109/ICGCIOT.2015.7380705.

[28] A. M. Yuganes A/P Parmesivan, Sazlinah Hasan, "Performance Evaluation of Load Balancing Algorithm for Virtual Machine in Data Centre in Cloud Computing," Int. J. Eng. Technol, vol. 7, no. 4, pp. 386–390, 2018, doi:

https://doi.org/10.14419/ijet.v7i4.31.23717.

[29] A. Yadav, "Load balancing in cloud computing environment using hybrid approach (ESCEL and PSO) algorithms," Adv. Comput. Sci. Inf. Technol, vol. 2, no. 8, pp. 10–13, 2015.

[30] Y. Meraihi, A. B. Gabis, A. Ramdane-Cherif, and D. Acheli, "A comprehensive survey of Crow Search Algorithm and its applications," Artif. Intell. Rev., vol. 54, no. 4, pp. 2669–2716, Apr. 2021, doi: 10.1007/S10462-020-09911-9.

[31] N. M. N. Arif Ullah, "BAT algorithm used for load balancing purpose in cloud computing: an overview," Int. J. High Perform. Comput. Netw., vol. 16, no. 1, 2020, [Online]. Available: http://www.inderscience.com/storage/f101241793621185.pdf

[32] R. Kaviarasan, P. Harikrishna, and A. Arulmurugan, "Load balancing in cloud environment using enhanced migration and adjustment operator based monarch butterfly optimization," Adv. Eng. Softw., vol. 169, p. 103128, Jul. 2022, doi: 10.1016/J.ADVENGSOFT.2022.103128.

[33] and L. F. M. Christopher, D. Kumar, "Migration-based load balance of virtual machine servers in cloud computing by load prediction," Int. J. Discov. Innov. Appl. Sci, vol. 2, no. 5, pp. 55–78, 2022.