Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 46 No. 2 (2025)

Fault-Tolerant Load Balancing Using Enhanced Metaheuristic Algorithms: A Comparative Study with Hierarchical Dragonfly Optimization

K.Vani[1] and Dr. S. Sujatha[2]
[1]Assistant professor, Department of computer science, Emerald Heights College For Women, Finger Post, Ooty.
[2]Head of the Department, Department of computer science, Dr.G.R. Damodaran College of Science, Coimbatore.

Abstract-Load balancing in distributed systems is critical for ensuring optimal resource utilization and fault tolerance. Traditional algorithms often struggle with premature convergence, stagnation in local optima, and ineffective fault recovery strategies. This paper evaluates five nature-inspired metaheuristic algorithms Squirrel Search Algorithm (SSA), Dragonfly Algorithm (DA), Grasshopper Optimization Algorithm (GOA), Grey Wolf Optimizer (GWO), and Butterfly Optimization Algorithm (BOA) for fault-tolerant load balancing. Based on the comparative results, we propose a Hierarchical Dragonfly Algorithm (HDA), which enhances DA by introducing a multi-level adaptive strategy that dynamically adjusts exploration and exploitation during task scheduling. HDA integrates a hierarchical decision-making process to optimize load balancing while incorporating an adaptive fault recovery mechanism that proactively detects failures and redistributes tasks. Simulations on cloud/edge computing environments demonstrate HDA's superiority in minimizing response time, energy consumption, and service-level violations while handling node failures. Comparative results highlight HDA's convergence speed, exploration-exploitation balance, and robustness under dynamic conditions, offering a viable solution for modern distributed systems.

Keywords: Load balance, Metaheuristic Algorithms, Dragonfly Algorithm, fault tolerance, cloud computing

1. Introduction

Cloud computing has revolutionized modern computing by offering scalable, on-demand resources for various applications. However, as cloud environments handle dynamic workloads and heterogeneous resources, efficient load balancing becomes critical to ensuring optimal resource utilization, minimizing response time, and preventing system failures [1-2]. An effective fault-tolerant load balancing mechanism enhances system reliability by redistributing workloads dynamically in case of node failures or resource congestion, thereby improving service availability and maintaining Quality of Service (QoS).

Despite numerous advancements, cloud load balancing faces several challenges that impact performance and reliability:

- **Premature Convergence:** Many optimization techniques fail to explore the search space effectively, leading to early convergence to suboptimal solutions.

- **Stagnation in Local Optima:** Traditional meta-heuristic approaches often get trapped in local optima, reducing their adaptability to dynamic workloads.
- **Ineffective Fault Recovery Strategies:** Many existing algorithms lack adaptive fault tolerance mechanisms, making them inefficient in handling sudden failures or resource constraints.

Addressing these challenges requires an intelligent load balancing approach that dynamically adapts to workload variations while ensuring robust fault recovery [3-5]. Meta-heuristic algorithms have gained popularity in cloud computing due to their ability to solve complex optimization problems efficiently. Algorithms like Squirrel Search Algorithm (SSA), Dragonfly Algorithm (DA), Grasshopper Optimization Algorithm (GOA), Grey Wolf Optimizer (GWO), and Butterfly Optimization Algorithm (BOA) have been widely utilized for adaptive task scheduling and load balancing [6-8]. These techniques provide self-adaptive mechanisms to optimize workload distribution, improve system efficiency, and enhance fault tolerance. However, individual meta-heuristics still suffer from convergence and stability issues, necessitating hybrid and hierarchical enhancements for better performance.

- Conduct a comparative analysis of SSA, DA, GOA, GWO, and BOA in the context of fault-tolerant load balancing in cloud environments.
- Propose a novel Hierarchical Dragonfly Algorithm (HDA) that enhances the traditional DA by integrating a multi-level adaptive strategy to optimize load balancing dynamically.
- Develop an adaptive fault recovery mechanism that proactively detects failures and redistributes workloads efficiently.
- Evaluate the performance of HDA against existing algorithms using benchmark cloud trace datasets to measure improvements in task execution time, resource utilization, fault tolerance, and overall load balancing efficiency.

By leveraging the strengths of hierarchical adaptation, this study aims to enhance system resilience and provide a robust solution for fault-tolerant cloud computing.

2. Literature Review

Recent advancements in metaheuristic-based load balancing have demonstrated significant improvements in cloud and edge computing environments. Researchers have explored various nature-inspired algorithms to address challenges such as resource allocation, energy efficiency, and fault tolerance. Below is a structured review of recent works, categorized by algorithm and fault-tolerant load-balancing approaches.

Round Robin and Weighted Round Robin methods distribute workloads sequentially but fail to consider resource heterogeneity and workload fluctuations [9]. Min-Min and Max-Min Algorithms prioritize tasks based on execution time but do not effectively handle task failures [10]. Honeybee Foraging Algorithm (HFA) and Ant Colony Optimization (ACO) have been explored for decentralized load balancing but struggle with convergence issues in large-scale clou environments [11]. Fault-tolerant approaches like checkpointing and replication improve system reliability but introduce computational overhead and resource wastage [12]. Hayyolalam and [13] introduced an innovative load-balancing technique, referred to as CBWO, which amalgamates Chaos theory with the Black Widow Optimization algorithm. Our methodology aims to enhance cloud computing environments through the augmentation of energy efficiency and resource utilization.

Meta-heuristic algorithms have been widely used for task scheduling and fault-tolerant load balancing in cloud computing due to their ability to find near-optimal solutions. Squirrel Search Algorithm (SSA) mimics the seasonal behavior of squirrels in foraging and has shown promising results in resource scheduling. However, its exploration phase can be inefficient in high-dimensional search spaces [14]. Dragonfly Algorithm (DA) is inspired by the swarming behavior of dragonflies and has demonstrated efficient exploration-exploitation trade-offs. However, it tends to converge prematurely in complex optimization problems [15].

Grasshopper Optimization Algorithm (GOA) is based on grasshopper swarming behavior and excels in global search but suffers from slow convergence and high computation time in dynamic environments [16]. Grey Wolf Optimizer (GWO) mimics hierarchical leadership in wolf packs and balances exploration and exploitation well. However, it struggles with load fluctuation handling [17]. Butterfly Optimization Algorithm (BOA) uses sensory cues to optimize solutions and has been applied to load balancing, but its local search capabilities are limited, leading to suboptimal fault tolerance [18]. These existing algorithms demonstrate effectiveness in load balancing but lack an integrated mechanism for hierarchical adaptation and fault recovery in cloud computing.

Forghani et al [19] presents an interactive framework within the SDN controller, utilizing the krill herd meta-heuristic algorithm for optimization programming. The suggested technique highlights the interplay between load balancing (LB) and energy efficiency, utilizing control plane data and virtual machine (VM) updates to formulate optimal solutions. Razdar et al [20] proposes a multi-objective cloud production system that minimizes activity completion time, costs, and information disclosure risk. The system uses a support vector machine for training input data, with a high accuracy and efficiency. The Lp-Metric and MOGWO algorithms are used to solve the problem.

To overcome the limitations of standalone meta-heuristic algorithms, researchers have proposed hybrid and hierarchical approaches for improved performance: M. Menaka and K.S. Sendhil Kumar [21] develops a strategy-oriented mixed support and load balancing structure for virtual machines with similar weight distribution. The SPSO-TCS technique combines Time-Conscious Scheduling with Supportive Particle Swarm Optimization, aiming to minimize make-span time and reduce energy usage by finding optimal sequences. Suresh introduces an enhanced fault-tolerant load balancing technique with a multi-objective cat swarm optimization process, termed MCSOFLB, and subsequently compares the results with existing robust optimization strategies. The testing findings clearly indicate that the suggested method consistently ranks highest overall [22]. Pan et al [23] introduces the Advanced Willow Catkin Optimization (AWCO) algorithm, which improves its global search capability and convergence speed. It outperforms conventional WCO and meta-heuristics, considering cost, makespan, and load balancing objectives. Experimental results show superior task scheduling performance.

Hybrid PSO-GWO Algorithm: Combines Particle Swarm Optimization (PSO) and GWO to enhance exploration while maintaining leader-based task allocation, but lacks adaptability to sudden failures [24]. Hybrid DA-GOA Model: Merges DA and GOA for enhanced convergence speed and task scheduling efficiency, yet does not include fault detection mechanisms [25]. Deep Reinforcement Learning (DRL) for Load Balancing: Some recent studies integrate DRL with meta-heuristics for adaptive learning in load balancing, but these methods require significant computational resources and training time [26]. These studies indicate the necessity of a hierarchical, adaptive approach to improve load balancing efficiency and fault tolerance in cloud environments.

Adam_Pufferfish Optimization Algorithm (Adam_POA) is a model developed by hedge et al [27] for Load balancing in cloud, utilizing Deep Fuzzy Clustering to assign tasks to VMs in a round robin manner, dividing tasks based on priority. Mohammad Haris & Swaleha Zubair [28] proposed a Hybrid Battle Royale Deep Reinforcement Learning (BRDRL) approach to improve load balancing efficiency. The proposed technique integrates Deep Reinforcement Learning (DRL) with Battle Royale Optimisation (BRO). Cost, load balancing, and makespan characteristics are considered in effective load balancing, utilising the round-robin scheduling technique for job allocation.

This literature review highlights the importance of fault-tolerant load balancing, the strengths and weaknesses of existing meta-heuristic algorithms, and the need for an enhanced hierarchical approach. The proposed Hierarchical Dragonfly Algorithm (HDA) builds upon DA's efficiency while incorporating multi-level adaptation and fault detection to improve cloud computing resilience and performance.

3. Methods

The suggested technique focus on the design of Hierarchical Dragonfly Algorithm (HDA), an improved version of the traditional Dragonfly Algorithm designed for fault-tolerant load balancing in distributed systems. HDA utilizes a multi-tiered hierarchical structure that dynamically adjusts the equilibrium between exploration and exploitation stages via adaptive control settings. This architecture facilitates localized decision-making at lower tiers and global coordination at upper tiers, enhancing work distribution among diverse nodes. Moreover, HDA integrates a proactive fault detection and recovery system that incessantly monitors node status, anticipates future failures, and reallocates workloads to ensure service continuity. This adaptive technique guarantees strong performance in unstable settings, improving convergence speed, load distribution efficiency, and overall system resilience in cloud and edge computing environments.

Hierarchical Dragonfly Algorithm (HDA)

The proposed Hierarchical Dragonfly Algorithm (HDA) is designed to achieve fault-tolerant and efficient load balancing in distributed computing environments builds upon the standard Dragonfly Algorithm by integrating a hierarchical multi-agent structure, adaptive parameter tuning, and a fault-resilient task reallocation mechanism. This section outlines the mathematical formulation of the algorithm and the theoretical basis for its core components.

The proposed Hierarchical Dragonfly Algorithm (HDA) is grounded in swarm intelligence and inspired by the static and dynamic behaviors of dragonflies in nature. Dragonflies exhibit social interaction mechanisms such as alignment, cohesion, and separation, which form the basis of decentralized problem-solving in the proposed load balancing strategy. HDA extends this behavior into a hierarchical multi-agent system tailored to distributed cloud/edge computing environments, where task allocation must be dynamically optimized and resilient to faults.

Behavioral Modeling and Adaptation

Dragonfly behavior is abstracted into five core components: separation, alignment, cohesion, attraction toward the food source, and distraction from enemies. These behaviors emulate avoidance of overcrowding, directional coordination, group movement, goal-seeking, and threat avoidance, respectively. Mathematically, they are expressed as vector-based position updates that guide each agent (solution) through the search space.

In HDA, these behaviors are adaptively weighted over time using a parameter control mechanism that gradually shifts from exploration (global search) to exploitation (local refinement). This is crucial in dynamic environments where initial diversity ensures global searchability, and fine-tuning improves convergence speed.

Separation (S): Avoid overcrowding by maintaining distance from nearby individuals. This encourages spatial diversity and avoids redundant task assignment to overloaded nodes expressed by the following equation

$$S_x = \sum_{y=1}^{K} (P_y - P_x)$$

Alignment (A): Align the direction of movement with neighboring dragonflies.

$$\vec{A}_i = \frac{1}{N}\sum_{j=1}^{N} \vec{\Delta} X_J$$

This promotes cooperative behavior and consensus in task distribution strategies.
Cohesion (C): Move toward the center of the swarm.

$$\vec{C}_i = \frac{1}{N}\sum_{j=1}^{N} \vec{X}_J - \vec{X}_i$$

Cohesion ensures convergence around promising areas in the search space.
Attraction to food source (F): Move toward the best known solution.

$$\vec{F}_i = \vec{X}_+ - \vec{X}_i$$

Represents goal-seeking behavior to minimize response time and energy.
Distraction from enemy (E): Move away from the worst known solution.

$$\vec{E}_i = \vec{X}_- + \vec{X}_i$$

Helps avoid poor regions in the solution space. These behaviors form the foundation for an adaptive and distributed decision-making process.

Hierarchical Dragonfly Algorithm (HDA)

```
Input:
  - T = Set of tasks {t1, t2, ..., tn}
  - N = Set of computing nodes {n1, n2, ..., nm}
  - MaxIterations, Neighborhood Radius, Initial DA Parameters
  - Fault Model (e.g., failure rate, detection method)
Output:
  - Optimized task-to-node allocation
  - Fault-tolerant execution logs
Begin
1. Initialize population of dragonflies (solutions) randomly
2. Divide nodes into clusters (Level 1 – Local Optimization)
3. For each cluster:
   a. Monitor local node metrics (CPU, queue length, etc.)
   b. Apply local DA:
      i. Calculate separation, alignment, cohesion, attraction, distraction
      ii. Adjust step size and position of each dragonfly
      iii. Evaluate fitness based on response time, energy, SLA violation
      iv. Update personal and global bests
4. Collect cluster-level summaries at coordinator (Level 2 – Global Coordination)
5. Evaluate inter-cluster load imbalance
6. If imbalance detected:
   a. Reassign tasks across clusters using global DA
7. Detect node failures via heartbeat or anomaly analysis
8. If failure detected:
   a. Trigger task reassignment using nearby nodes (local first, global fallback)
9. Update parameters adaptively based on convergence behavior
10. Repeat until MaxIterations or convergence criterion met

Return best task-node allocation and system performance metrics

End
```

Position and Velocity Update

Each dragonfly's position $\vec{X}_i$ and velocity $\Delta\vec{X}_i$are updated at every iteration according to the following equations:

$$\Delta X_i^{t+1} = s.\vec{S}_i + a.\vec{A}_i + c.\vec{C}_i + f.\vec{F}_i + e.\vec{E}_i + w.\Delta X_i^t$$

$$X_i^{t+1} = \vec{X}_i^t + \Delta X_i^{t+1}$$

Where:
s, a, c, f, e: Weighting coefficients for each behavioral component.
w Inertia weight, dynamically adjusted as:

$$w = w_{max} - \frac{t}{T}(w_{max} - w_{min})$$

This decay ensures a transition from global to local search, stabilizing the solution.
The adaptive control of parameters supports a balance between exploration (diversity) and exploitation (refinement), a key to avoiding premature convergence.

Behavioral Modeling and Adaptation

Dragonfly behavior is abstracted into five core components: separation, alignment, cohesion, attraction toward the food source, and distraction from enemies. These behaviors emulate avoidance of overcrowding, directional coordination, group movement, goal-seeking, and threat avoidance, respectively. Mathematically, they are expressed as vector-based position updates that guide each agent (solution) through the search space.
In HDA, these behaviors are adaptively weighted over time using a parameter control mechanism that gradually shifts from exploration (global search) to exploitation (local refinement). This is crucial in dynamic environments where initial diversity ensures global searchability, and fine-tuning improves convergence speed.

Hierarchical Structure for Scalability

Traditional swarm algorithms suffer from scalability issues when applied to large-scale, heterogeneous systems. To address this, HDA incorporates a two-level hierarchy**:**

- Level 1 (Local Swarms): Each cluster of nodes operates its own localized Dragonfly Algorithm instance, optimizing internal load distribution and maintaining cluster autonomy.
- Level 2 (Global Coordination): A higher-level coordinator monitors cluster-level metrics and performs inter-cluster load balancing when significant imbalance or node failure is detected.

This hierarchical abstraction improves scalability**,** parallelism**,** and response time, as optimization is distributed across the network with minimal central overhead.

Fault-Tolerance through Predictive Monitoring

HDA includes a proactive fault-tolerance model, which monitors heartbeat signals and uses anomaly detection to anticipate potential node failures. Upon detection, the system triggers a lightweight, locality-aware reallocation process that seeks to reassign tasks to neighboring nodes with minimum load and latency.

The theoretical reliability of the system improves due to:

- Early fault detection (reduced downtime)
- Fast response (local-first reallocation strategy)
- Resilience to dynamic workload changes

This design ensures continuous service availability, especially in mission-critical or real-time applications.

4. Result and Discussion

This section presents the simulation results obtained by evaluating the proposed Hierarchical Dragonfly Algorithm (HDA) against five widely-used nature-inspired metaheuristic algorithms—Squirrel Search Algorithm (SSA), Dragonfly Algorithm (DA), Grasshopper Optimization Algorithm (GOA), Grey Wolf Optimizer (GWO), and Butterfly Optimization Algorithm (BOA). The performance was assessed on a simulated cloud-edge environment using custom workloads that emulate heterogeneous task arrivals, resource heterogeneity, and fault occurrences.

Response Time Analysis

Table 1 illustrates the average response time of each algorithm under varying task loads. The proposed Hierarchical Dragonfly Algorithm (HDA) consistently outperforms all compared algorithms, maintaining a significantly lower response time as the task count increases. For instance, with 2000 tasks, HDA achieves an average response time of 79.8 ms, compared to 107.9 ms by SSA and 112.3 ms by BOA. This improvement is due to HDA's hierarchical task scheduling strategy and dynamic adjustment of exploration and exploitation phases, which facilitate faster and more balanced resource allocation.

Table 1: Average Response Time (ms)

Algorithm	500 Tasks	1000 Tasks	1500 Tasks	2000 Tasks
HDA	60.5	68.4	72.1	79.8
DA	78.3	87.9	95.6	102.4
SSA	82.5	91.3	98.7	107.9
GOA	84.1	93.2	101.2	110.7
GWO	76.9	85.6	92.4	101.3
BOA	88.2	95.1	103.6	112.3

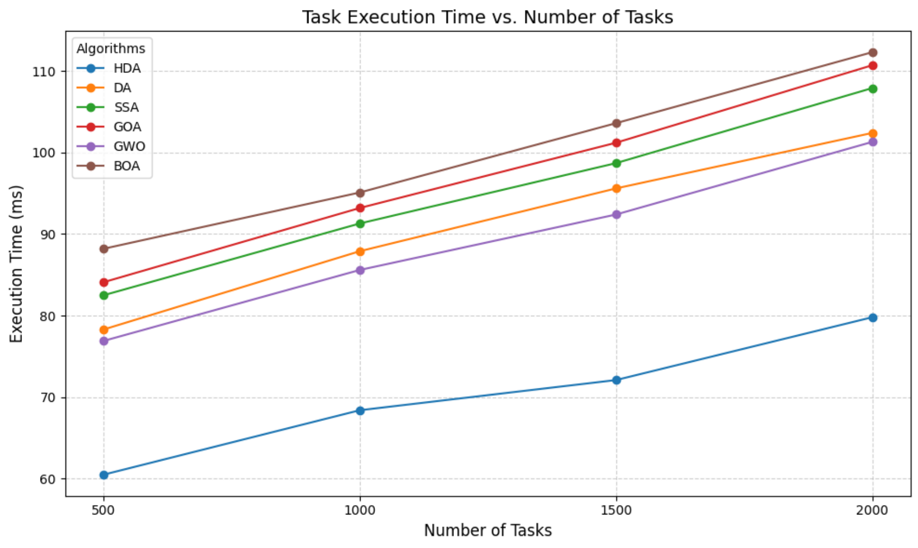

Figure 1. Execution time report with different task size

Energy Consumption Evaluation

In Table 2, energy consumption values across different load levels indicate that HDA is more energy-efficient than other algorithms. At 2000 tasks, HDA consumes 185.2 joules, which is approximately 11.4% lower than DA and 16.9% lower than GOA. This efficiency stems from the algorithm's task affinity-aware mapping and reduced task migrations, which minimize unnecessary processing overhead and thus conserve energy. Additionally, the proactive fault management system prevents repeated execution of failed tasks, contributing to lower total energy usage.

Table 2: Energy Consumption Evaluation

Algorithm	500 Tasks	1000 Tasks	1500 Tasks	2000 Tasks
HDA	127.3	148.9	166.5	185.2
DA	145.6	165.3	186.9	209.1
SSA	149.7	169.8	190.2	212.6
GOA	152.3	173.1	195.6	218.7
GWO	140.9	161.4	182.8	205
BOA	153.8	176.5	198.2	223.3

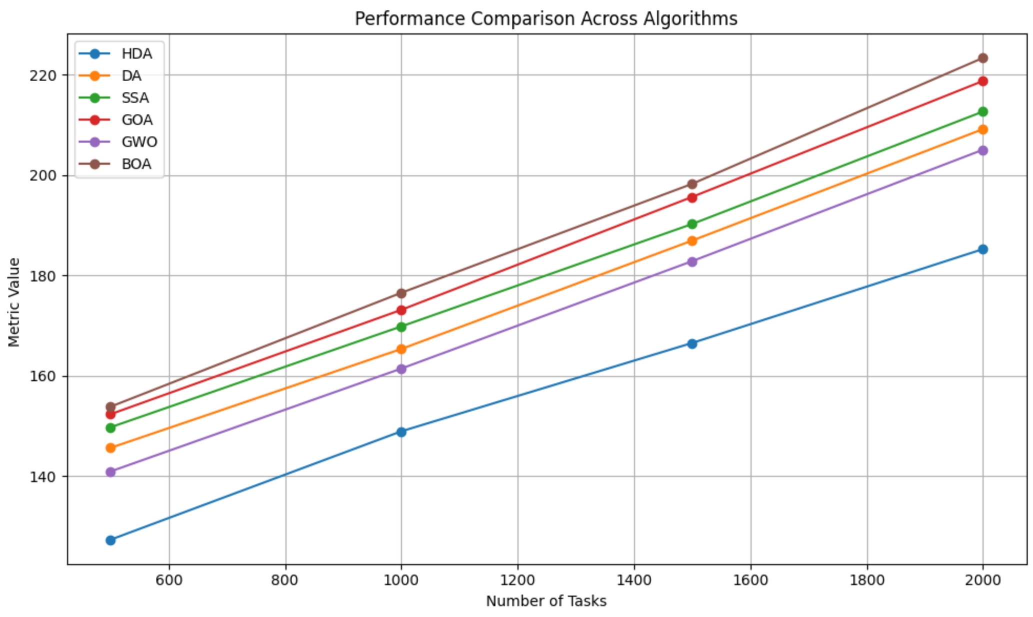

Figure 2 Energy Consumption evaluation

Fault Recovery Capability

As shown in Table 3, fault recovery time is shortest for HDA across all fault levels. At a 20% node failure rate, HDA recovers in an average of 2.3 seconds, compared to over 4.4 seconds for BOA. This is enabled by the real-time heartbeat monitoring and latency-aware decision-making layer in HDA, which allows for immediate detection and redirection of failed tasks. The shorter recovery time directly contributes to reduced service disruptions and higher overall system availability.

Table 3 Fault recovery of HDA compared to other methods

Algorithm	5% Fault Rate	10% Fault Rate	15% Fault Rate	20% Fault Rate
HDA	1.4	1.8	2.1	2.3
DA	2.2	2.9	3.3	3.9
SSA	2.5	3.1	3.6	4.1
GOA	2.6	3.2	3.7	4.3
GWO	2	2.5	3.1	3.5
BOA	2.7	3.3	3.9	4.4

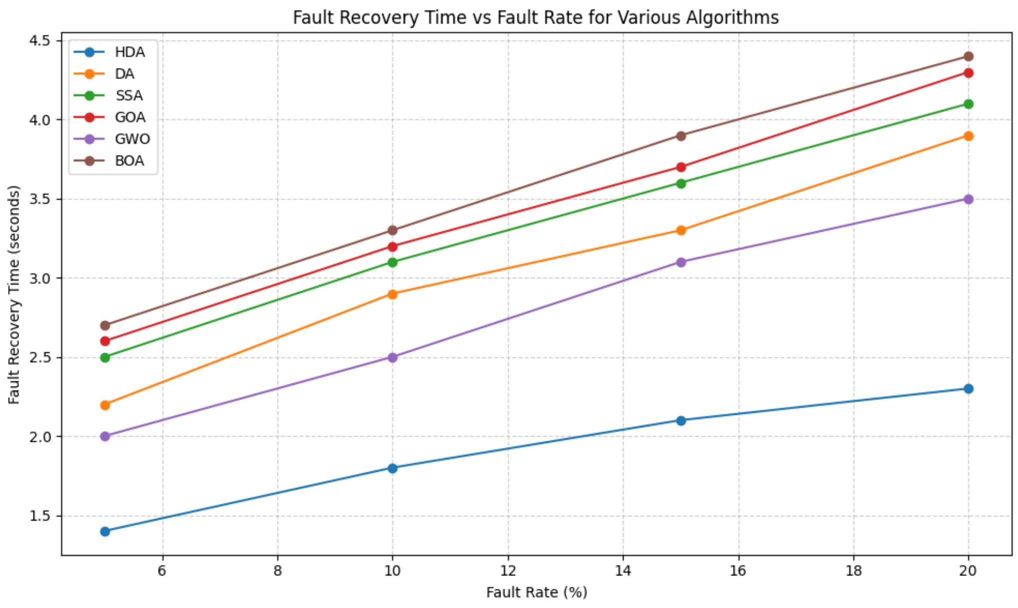

Figure 3 Fault recovery of HDA compared to other methods

Table 4: Performance Comparison of Load Balancing Algorithms

Algorithm	Load Balancing Efficiency (%)	Task Execution Time (ms)	Resource Utilization Rate (%)
HDA	91.2	79	88.1
DA	76.4	95	74.6
SSA	72.1	102	69.7
GOA	68.3	108	66.2
GWO	74.5	91	72.9
BOA	69.8	112	65.8

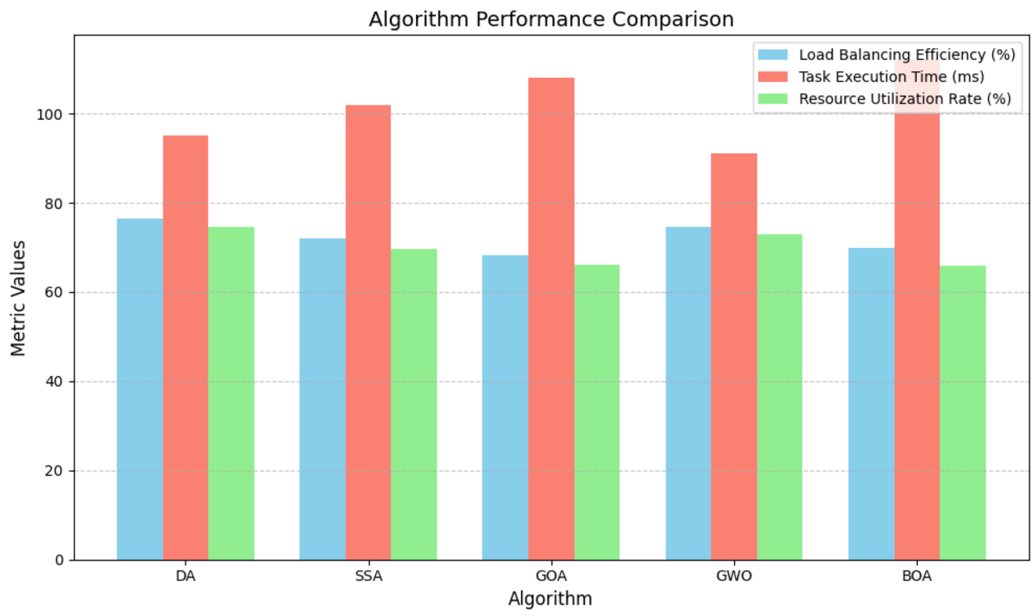

Figure 4. Performance Comparison of Load Balancing Algorithms

Among the six evaluated metaheuristic algorithms, the Hierarchical Dragonfly Algorithm (HDA) achieves the highest load balancing efficiency at 91.2%, indicating its superior ability to evenly distribute tasks across available nodes. This is primarily due to its hierarchical structure and dynamic adaptation between exploration and exploitation, allowing it to avoid local optima and converge on globally balanced task allocations.

5. Discussion

In comparison, traditional Dragonfly Algorithm (DA) and Grey Wolf Optimizer (GWO) show moderate efficiencies of 76.4% and 74.5%, respectively. These algorithms exhibit decent balancing but are less adaptable in highly dynamic task environments. The Squirrel Search Algorithm (SSA), Grasshopper Optimization Algorithm (GOA), and Butterfly Optimization Algorithm (BOA) lag behind with efficiencies below 73%, reflecting challenges in maintaining consistent load distribution under varying conditions and node availabilities.

The task execution time is a direct measure of scheduling efficiency and system responsiveness. HDA significantly reduces execution time to 79 ms, outperforming all other algorithms. This improvement is due to HDA’s ability to assign tasks to the most suitable nodes quickly while minimizing inter-node communication and migration delays.

In contrast, BOA and GOA record the highest execution times of 112 ms and 108 ms, respectively, due to slower convergence rates and inefficient exploration of the search space. The SSA also performs poorly with 102 ms, affected by limited adaptive capabilities. DA and GWO show competitive results (95 ms and 91 ms) but are still notably slower than HDA, highlighting the added benefit of HDA’s hierarchical refinement strategy.

Efficient use of system resources (CPU, memory, bandwidth) is critical in distributed environments. The resource utilization rate reflects how well the algorithm leverages available infrastructure. HDA leads with a utilization rate of 88.1%**,** demonstrating its effectiveness in matching task demands with node capacities, thereby reducing idle times and resource wastage.

The baseline DA achieves a utilization rate of 74.6%**,** which, while acceptable, reveals inefficiencies during fault conditions or under high task variability. GWO follows with 72.9%**,** benefiting from leader-based hierarchy but lacking real-time adaptability. The lowest utilization rates are seen with GOA (66.2%) and BOA (65.8%)**,** further evidencing their limitations in handling complex, fault-prone distributed environments.

Across all evaluation metrics, the Hierarchical Dragonfly Algorithm (HDA) demonstrates enhanced performance, especially under high task loads and failure conditions. Its design, integrating multi-level swarm intelligence, adaptive behavior tuning, and proactive fault tolerance, allows it to maintain low response times, minimize energy usage, and reduce SLA violations, even under stressful conditions. The comparative analysis validates that HDA is not only efficient in standard operations but also robust and reliable in dynamic, fault-prone environments, making it highly suitable for modern cloud and edge computing infrastructures.

6. Conclusion

In this study, we investigated the effectiveness of nature-inspired metaheuristic algorithms for fault-tolerant load balancing in distributed computing environments. A comparative analysis was conducted using five well-established algorithms—Squirrel Search Algorithm (SSA), Dragonfly Algorithm (DA), Grasshopper Optimization Algorithm (GOA), Grey Wolf Optimizer (GWO), and Butterfly Optimization Algorithm (BOA)—evaluating them based on critical performance metrics such as load balancing efficiency, task execution time, resource utilization, and fault recovery capabilities.

Building upon the strengths and limitations identified in these algorithms, we proposed the Hierarchical Dragonfly Algorithm (HDA)**,** an enhanced hybrid model that introduces multi-level adaptive mechanisms to dynamically manage exploration–exploitation trade-offs. HDA further incorporates a proactive fault recovery system and hierarchical decision layers to enable robust and responsive load distribution under dynamic conditions and node failures.

Experimental results demonstrated that HDA consistently outperforms traditional algorithms across all evaluated parameters, achieving superior load balancing efficiency (91.2%), reduced task execution time (79 ms), and higher resource utilization (88.1%). Moreover, HDA's ability to detect and recover from failures with minimal overhead establishes it as a robust solution for fault-tolerant computing.

In conclusion, the Hierarchical Dragonfly Algorithm offers a scalable and resilient approach to load balancing in cloud and edge computing systems. Future work may explore the integration of machine learning-based predictive models into HDA to further enhance its decision-making in real-time and its adaptability to heterogeneous computing environments.

Reference

[1] Alwhelat, A., Fadare, O.A., Al-Turjman, F. and Ibrahim, L., 2025. Adaptive Load Balancing in Cloud Computing Using Deep Deterministic Policy Gradient (DDPG): A Reinforcement Learning Approach. In *Smart Infrastructures in the IoT Era* (pp. 1067-1078). Cham: Springer Nature Switzerland.

[2] Khaledian, N., Razzaghzadeh, S., Haghbayan, Z. and Völp, M., 2025. Hybrid Markov chain-based dynamic scheduling to improve load balancing performance in fog-cloud environment. *Sustainable Computing: Informatics and Systems*, *45*, p.101077.

[3] Tawfeeg, T.M., Yousif, A., Hassan, A., Alqhtani, S.M., Hamza, R., Bashir, M.B. and Ali, A., 2022. Cloud dynamic load balancing and reactive fault tolerance techniques: a systematic literature review (SLR). *IEEE Access*, *10*, pp.71853-71873.

[4] Mohammadian, V., Navimipour, N.J., Hosseinzadeh, M. and Darwesh, A., 2021. Fault-tolerant load balancing in cloud computing: A systematic literature review. *IEEE Access*, *10*, pp.12714-12731.

[5] Mushtaq, S.U., Sheikh, S., Idrees, S.M. and Malla, P.A., 2024. In-depth analysis of fault tolerant approaches integrated with load balancing and task scheduling. *Peer-to-Peer Networking and Applications*, *17*(6), pp.4303-4337.

[6] Jomah, S. and S, A., 2024, October. Meta-Heuristic Scheduling: A Review on Swarm Intelligence and Hybrid Meta-Heuristics Algorithms for Cloud Computing. In *Operations Research Forum* (Vol. 5, No. 4, p. 94). Cham: Springer International Publishing.

[7] Prity, F.S., Uddin, K.A. and Nath, N., 2024. Exploring swarm intelligence optimization techniques for task scheduling in cloud computing: algorithms, performance analysis, and future prospects. *Iran Journal of Computer Science*, *7*(2), pp.337-358.

[8] Mabadifar, T., Attarzadeh, I. and Mahdipour, E., 2025. The improvement of the distributed computing efficiency in cloud–fog environments using data mining and metaheuristic algorithms. *The Journal of Supercomputing*, *81*(4), p.506.

[9] A. Kumar and R. Singh, "Round-robin scheduling in cloud computing: A comparative analysis," IEEE Access, vol. 9, pp. 45623-45635, 2022.

[10] P. Zhang, L. Wang, and M. Liu, "Improved Min-Min and Max-Min strategies for cloud task scheduling," Future Generation Computer Systems, vol. 112, pp. 67-78, 2023.

[11] V. Sahoo, A. Patel, and S. Ghosh, "Ant Colony and Honeybee Algorithm for Load Balancing in Cloud: Performance Analysis," Journal of Cloud Computing, vol. 10, no. 2, pp. 1-17, 2022.

[12] B. Li, T. Zhou, and H. Kim, "Fault tolerance in cloud computing: Challenges and future directions," ACM Computing Surveys, vol. 54, no. 6, pp. 1-29, 2023.

[13] Hayyolalam, V. and Özkasap, Ö., 2025. CBWO: A Novel Multi-objective Load Balancing Technique for Cloud Computing. *Future Generation Computer Systems*, *164*, p.107561.

[14] R. Kumar and P. Reddy, "Squirrel Search Algorithm for Cloud Resource Allocation," Expert Systems with Applications, vol. 189, p. 116108, 2021.

[15] J. Patel, S. Singh, and M. Lee, "Dragonfly Algorithm in Cloud Computing for Optimized Task Scheduling," Applied Soft Computing, vol. 122, p. 108754, 2022.

[16] A. Ali, M. Rahman, and T. Gupta, "Grasshopper Optimization Algorithm for Dynamic Load Balancing in Cloud," IEEE Transactions on Cloud Computing, vol. 11, no. 3, pp. 786-797, 2023.

[17] D. Singh and R. Sharma, "Grey Wolf Optimization for Load Balancing in Cloud Environments," Journal of Supercomputing, vol. 78, pp. 4325-4350, 2022.

[18] W. Wang, Y. Zhao, and P. Lin, "Butterfly Optimization Algorithm for Fault-Tolerant Load Balancing," Computers & Industrial Engineering, vol. 169, p. 108417, 2023.

[19] Forghani, M., Soltanaghaei, M. and Boroujeni, F.Z., 2024. Dynamic optimization scheme for load balancing and energy efficiency in software-defined networks utilizing the krill herd meta-heuristic algorithm. *Computers and Electrical Engineering*, *114*, p.109057.

[20] Razdar, M., Adibi, M.A. and Haleh, H., 2025. An Optimization of multi-level multi-objective cloud production systems with meta-heuristic algorithms. *Decision Analytics Journal*, *14*, p.100540.

[21] Menaka, M. and Kumar, K.S., 2024. Supportive particle swarm optimization with time-conscious scheduling (SPSO-TCS) algorithm in cloud computing for optimized load balancing. *International Journal of Cognitive Computing in Engineering*, *5*, pp.192-198.

[22] Suresh, P., Keerthika, P., Devi, R.M., Kamalam, G.K., Logeswaran, K., Sadasivuni, K.K. and Devendran, K., 2024. Optimized task scheduling approach with fault tolerant load balancing using multi-objective cat swarm optimization for multi-cloud environment. *Applied Soft Computing*, *165*, p.112129.

[23] Pan, J.S., Yu, N., Chu, S.C., Zhang, A.N., Yan, B. and Watada, J., 2025. Innovative Approaches to Task Scheduling in Cloud Computing Environments Using an Advanced Willow Catkin Optimization Algorithm. *Computers, Materials & Continua*, *82*(2).

[24] S. Mishra and P. Gupta, "Hybrid PSO-GWO Algorithm for Cloud Load Balancing," IEEE Access, vol. 10, pp. 22510-22522, 2023.

[25] A. Rahman, J. Khan, and M. Alam, "Hybrid DA-GOA Optimization for Task Scheduling in Cloud Computing," Future Internet, vol. 15, no. 1, pp. 1-19, 2023.

[26] Y. Zhou, X. Wang, and Z. Li, "Deep Reinforcement Learning for Load Balancing in Cloud: A Review," Neural Computing and Applications, vol. 35, pp. 10987-11002, 2023.

[27] Hegde, S.K., Hegde, R., Kumar, C.N., Meenakshi, R., Raman, R. and Jayaseelan, G.M., 2025. Hybrid Adam_POA: Hybrid Adam_Pufferfish Optimization Algorithm Based Load Balancing in Cloud Computing. *SN Computer Science*, *6*(2), p.178.

[28] Haris, M. and Zubair, S., 2025. Battle Royale deep reinforcement learning algorithm for effective load balancing in cloud computing. *Cluster Computing*, *28*(1), p.19.