International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

Hybrid GA–AI Framework for Adaptive Load Balancing

Vaishnav Anand
Networking and Communication
SRM Institute of Science and Technology
Kattankaluthur, India
va5149@srmist.edu.in

C. Malathy
Networking and Communication SRM Institute of Science and Technology
Kattankaluthur, India
malathyc@srmist.edu.in

CH. Vasanth Kumar
Networking and Communication
SRM Institute of Science and Technology
Kattankaluthur, India
vasanthc2@srmist.edu.in

Devadharshini S
Networking and Communication
SRM Institute of Science and Technology
Kattankaluthur, India
ds9527@srmist.edu.in

Adithyan M
Networking and Communication SRM Institute of Science and Technology
Kattankaluthur, India
am1783@srmist.edu.in

Deepak Kumar R
Networking and Communication
SRM Institute of Science and Technology
Kattankaluthur, India
dr5661@srmist.edu.in

***Abstract*- Load balancing in dynamic cloud environments is challenging when using standard, conventional strategies. The study proposes a hybrid GA-AI framework for applying Genetic Algorithms capabilities combined with lighter-weight Artificial Intelligence to enable adaptive task scheduling capabilities for cloud. A tree structure knowledge base allows for recognition and reuses established solutions for repetitive workloads to reduce wasteful calculations and achieve a higher scheduling efficiency. The proposed framework was evaluated by implementing three prototypes of the system on a local Kubernetes cluster: a baseline GA-AI hybrid, a more advanced resource-aware version that includes similarity matching capabilities, and an ML-enhanced version of the system. Overall results depict a situation where the baseline model offers more rapid scheduling decisions while all resource aware configurations made scheduling decisions with higher latency. The resource aware similarity-matching version tasks performed competitively, yet has limited use of the knowledge base due to strict safety thresholds and incurred considerable decision costs. The ML-enhanced version successfully utilized online learning to manage the majority of scheduling decisions (54%), validating the system's ability to learn real-time resource relationships. The results suggested that there was strong potential for scaling the framework and also emphasizes the compute load as being a factor in identifying advanced forms of pattern matching. The proposed hybrid method integrates GA optimization with AI-driven for future work.**

Keywords— Genetic Algorithm, Artificial Intelligence, Load Balancing, Kubernetes, Cloud Computing, Adaptive Scheduling, Knowledge Base, Machine Learning, Resource Optimization, Container Orchestration..

I. INTRODUCTION

Cloud computing has greatly changed the way by which the resources are provided. It offers scalable and on-demand capabilities. However, modern workloads have a tendency to produce sudden spikes in web traffic or heavy workloads, which helped to reveal the disadvantages of traditional infrastructure. Classic load balancers [8][11], like Round Robin or First-Fit, relies on fixed rules and lack the intelligence or techniques to adapt to the varying requirements of the workload. To address the problem, researchers have applied metaheuristic methods like Genetic Algorithms [12][16]. These algorithms can search large solution spaces to find near optimal task-node mappings. Despite the advantages, GAs can be computationally intensive, especially when the system tries to re-optimizes similar workloads. Recent studies have shown that the connection between AI and metaheuristics could be utilized to address these issues, but each method has its own disadvantages. In AI-based solutions, Kalaiselvi et al. [2] suggested a load balancer that combines Decision Trees, K-means clustering, and Reinforcement Learning. Even though it helped to improved latency and security, the AI monitoring component added a significant amount of computational overhead. Similarly, Vachhani et al. [1] used LSTM and ARIMA models to predict workloads in Azure, achieving a 20% increase in utilization. However, their solution was limited to a single cloud provider and did not take into consideration of important factors like energy and security.

In the domain of metaheuristics [3][6][12][20], several studies have attempted to improvise the GA implementations. Bothra et al. [3] proposed a Hybrid GA methodology that includes heuristics for scientific workflows, which reported about a 21% reduction in the cost. Still, their evaluation relied on simulations and did not involve real infrastructure. While Mogal & Sonaje [6] proved that the performance of different versions of the GA surpassed the performance of static methods in terms of reducing load imbalance, the authors recognized the problem of scalability in large-scale environments. Furthermore, the work done by Chen & Wen [10] which successfully applied GA to microservice placement, improving resource use, it did consider GPU-aware scheduling or large-scale testing. Together, these studies point to a major research gap: the lack of real- time, adaptive, and memory-augmented systems that can effectively combine GA optimization with AI-based pattern recognition in the actual cloud environments. The proposed framework combines the searching power of GAs with AI-driven workload recognition. The framework utilizes a tree-structured knowledge base which helps to store and retrieve optimized solutions for recurring patterns. The proposed approach helps in minimizing redundant calculations while ensuring adaptability.

II. PROPOSED SYSTEM

The proposed framework aims to enhance the process of task scheduling through a structured pipeline. It begins with workload analysis, where the system analyses the incoming tasks by extracting a numerical feature vector [13][14]. This vector is based on the total number of containers, overall CPU and RAM demands as well as the computational intensity. This process turns the scheduling request into

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

structured data input. Once the extraction process is complete, the system performs pattern matching where in which it queries a tree-structured knowledge base and applies Euclidean distance to compare the current vector with historical data. This allows the immediate reuse of an optimal solution if a pattern is found with a certain level of similarity threshold. If there is no similar match, the framework moves to Neural Predictive Modeling. In this phase, it uses a trained Multi-Layer Perceptron (MLP) to predict the best node assignments based on the resource- capacity relationships learned during the process. In new scenarios where new optimization is needed, the system uses a Genetic Algorithm (GA) to generate a variety of schedules that reduce load imbalance and fragmentation of the resources. This stage promotes continuous learning. The optimized solution is then stored in the knowledge base and is used to retrain the neural network, ensuring that the model improves over time. The framework ends with Task Distribution.

III. IMPLEMENTATION

All three framework versions implementation were carried out using Python 3.12 and were deployed on a local Kubernetes cluster which was orchestrated using kind. The system utilized the official Kubernetes Python client (v29+) to manage pod orchestration, perform resource introspection, and monitor performance metrics via the Kubernetes Watch API. Even though all implementations had a similar modular architecture, they differed in terms of their Genetic Algorithm (GA) configurations, fitness function structures, knowledge base logic and AI components integration.

Code 1: Baseline GA–AI Hybrid - The baseline code termed as Code 1, was developed for lightweight, high- frequency scheduling scenarios. The workload consisted of two to four containers per run, randomly selected with moderate resource demands (100–300m CPU, 128–256Mi RAM) and short running times of 5 to 15 seconds. For this code, a lightweight Genetic Algorithm configuration was considered which utilized a small population of 10 individuals that evolved over 5 generations where in which the selection relied on fitness-weighted random choices, the single-point crossover and a mutation rate of 10% helped with the evolution. The fitness function was simplified to calculate the standard deviation of container counts per node, ignoring specific resources such as CPU and memory usage. The associated Knowledge Base (KB) used a scalar feature vector representing the container count, using a direct O (1) dictionary lookup that required exact matches, with no similarity thresholding.

Code 2: Enhanced GA–AI with Resource Awareness – Utilizing the baseline code as a base, Code 2 was developed to include multi-dimensional resource awareness and fuzzy pattern matching techniques. The workload was increased to 2–3 containers with higher resource requirements (100–400m CPU, 256– 1024Mi RAM) and a reduced execution time of 2 seconds to speed up the simulation process. The GA configuration was then improvised to include a population of 15 with 8 generations, utilizing a tournament-style selection method with a higher mutation rate of 15% to identify a larger search space. The fitness function was then upgraded to include a multi-objective model that balanced CPU and RAM utilization along with container counts, with strict penalties for node overloads. The Knowledge Base was improved to store a 4-tuple feature vector (container count, total CPU, total RAM, and workload size). Instead of exact matches, the system calculated the Euclidean distance between vectors, reusing solutions only if the distance fell below a similarity threshold of 100. This approach achieved a 13% cache hit rate across 100 evaluation runs.

Code 3: ML-Augmented Hybrid (ML-GA-AI) – In the final iteration or Code 3, an incremental machine learning approach was incorporated using an MLPRegressor with hidden layers (64, 32). This approach was designed to process a 7- dimensional feature vector—capturing container requirements, node capacity, and utilization ratios to predict optimal node assignments. In order to overcome the "cold start" problem, online learning strategy was incorporated, retraining the model with new data after every scheduling event. The switching logic enabled the system to activate the ML component after collecting just 20 training samples. In effect, across 100 experimental runs, the system successfully transitioned from heuristic to predictive scheduling. In terms of the actual decision-making process the ML component managed 54% of decisions, while the Knowledge Base handled 38%. However, this increased intelligence came with a trade-off; the decision time was increased to 234.58 ms due to the computational overhead of real-time feature extraction and inference.

In order to provide a strict and comparable evaluation, all the three variants utilized a similar set of infrastructure components. Caching for node lists and allocatable resources to minimize API latency was implemented using Kubernetes client, while the pod lifecycle events were managed with effective error handling. The proposed hybrid frameworks were evaluated against four standard scheduling algorithms namely Round Robin, First Fit, Best Fit, and a greedy Load Balancing approach. Performance was monitored using the Kubernetes Watch API to track phase transitions, measuring the end-to-end duration from pod creation to success or failure, with strict timeouts enforced to maintain the integrity of the experiment.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

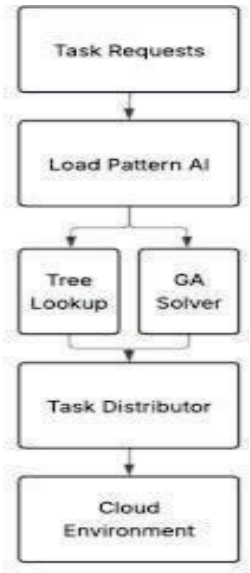

Fig. 1. Hybrid GA-AI Load Balancing Framework

IV. RESULT AND ANALYSIS

Using actual pod deployments, three iterations of the hybrid GA-AI load balancing framework were tested on a local Kubernetes cluster. Workload modeling and optimization logic were gradually improved by each of these. Decision latency, average task duration, and success rate were used to gauge performance.

Code 1 utilized a simple GA with a population of 10, 5 generations, and only container count balance in its fitness function. The only feature considered in pattern matching was the number of containers. Over 100 runs, it achieved an average task duration of 1.4 seconds with a decision time of 3.1 ms and a 100% success rate as shown in fig.2. While functional, its lack of resource awareness limited its realism.

Code 2 developed a more intelligent knowledge base using similarity matching using a GA that added CPU and RAM utilizations in its multi-objective fitness function. It had an increased population size of 15, a population of 8 generations, and a mutation rate of 15%. Workloads were "2-3" containers, needing varying resource requests (CPU: 100-400m, RAM: 256-1024Mi), with the same runtime of 30 seconds. Over 100 runs, this "Similarity Ai Hybrid" model had a reasonable average duration of 41 seconds, an increased decision latency of 23 ms and a 100% success rate as shown in fig. 3. The efficiency of the KB was only 13% (13 hits, 87 misses). This indicates a clear trade-off, since the system utilized a stricter similarity threshold of 100.0 to ensure safe resource allocation, forcing the GA to execute more frequently and hence the overhead.

Code 3 (ML-Augmented) introduced an MLPRegressor to guide GA initialization and included a heuristic (kb_effectiveness_score) to decide between knowledge base reuse, ML prediction, or GA fallback. An Online Learning strategy was introduced to overcome the "cold start" problem. The ML model was triggered in the testing. Over 100 runs, the system switched to predictive mode significantly, with the ML model managing 54% of all decisions and the Knowledge Base handling 38% and GA handling the rest 8% as shown in fig 5. The decision time rose to 234.58 ms due to the overhead of feature extraction and inference, the system maintained a 100% success rate as shown in fig. 4. To validate the integration, 5-Fold Cross- Validation was deployed during the training phase. The analysis revealed a significant learning curve, with the Mean Squared Error (MSE) dropping consistently from a peak of 0.014 to stabilizing near 0.002 over 93 training batches as shown in fig. 6, proving that the system effectively learned resource-capacity relationships in real-time.

All algorithms maintained their 100% success rate, indicating system stability. Overall, Code 2(Enhanced GA- AI Hybrid with Resource Awareness) presented the best trade-off, proving that only moderate enhancements to GA especially resource-aware fitness and practical pattern matching are able to outperform both traditional heuristics and overengineered AI hybrids in real Kubernetes environments.

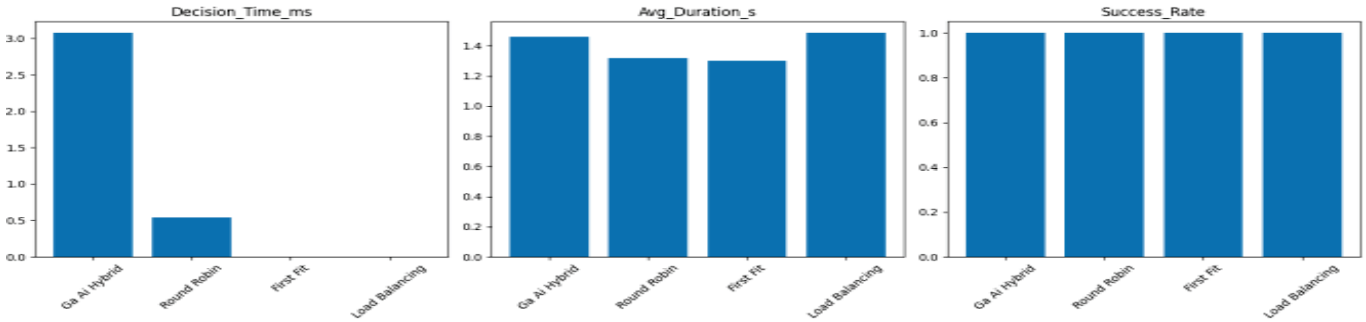

Fig. 2. Comparative metrics (decision time, duration, success rate) of the Baseline GA–AI Hybrid over 100 runs.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

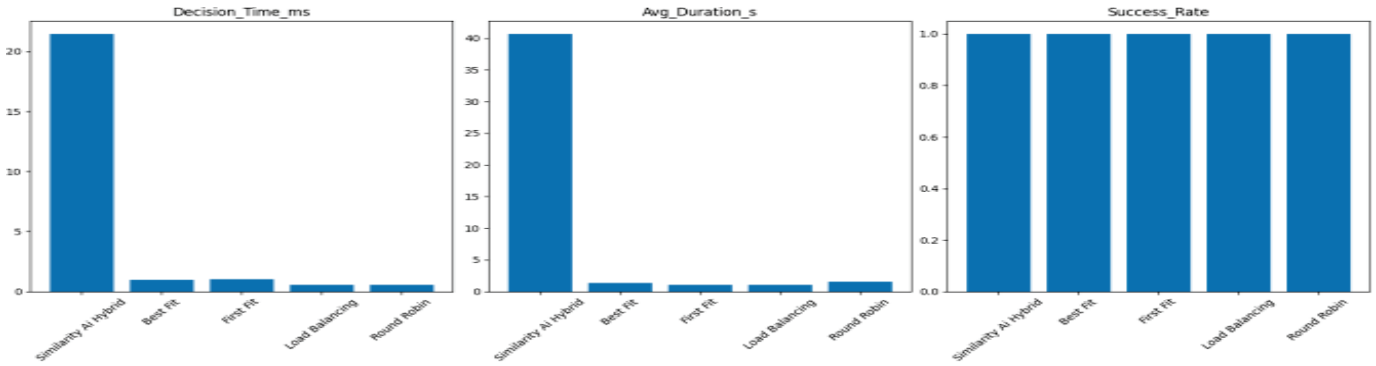

Fig. 3. Comparative metrics (decision time, duration, success rate) of the GA-AI with Resource awareness and similarity matchmaking over 100 runs.

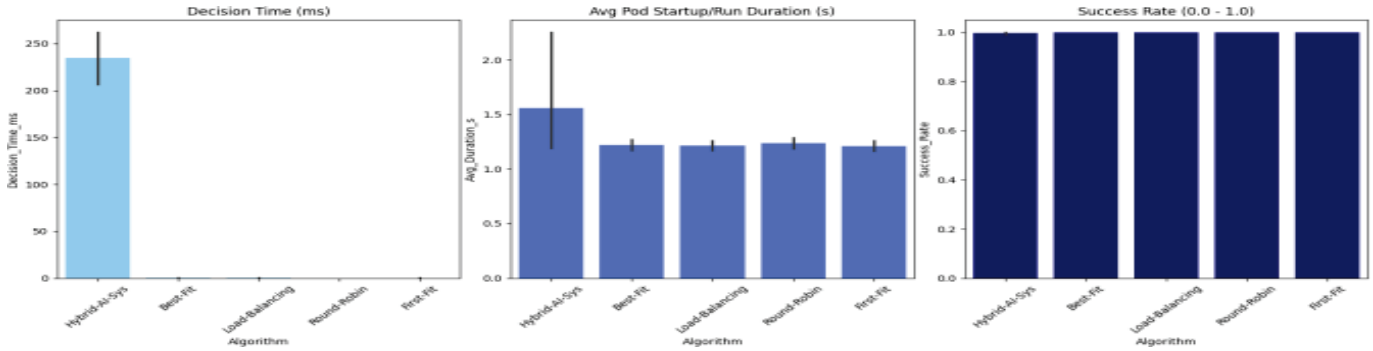

Fig. 4. Comparative metrics (decision time, duration, success rate) of the ML-Augmented Hybrid over 100 runs.

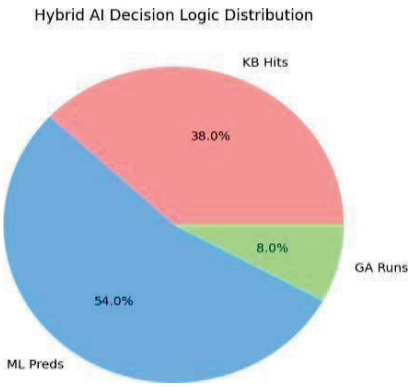

Fig. 5. Hybrid AI Decision Logic Distribution

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

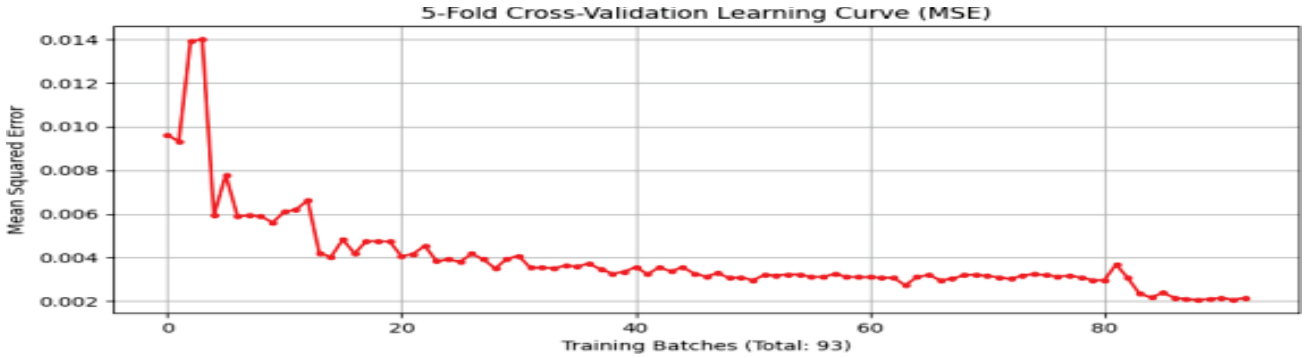

Fig. 6. Validation of the predictive ml model susing 5-Fold Cross-Validation

V. CONCLUSION

The three hybrid GA-AI implementations' experimental evaluation revealed a clear balance between complexity and utility. Although baseline GA-AI hybrid lacked resource awareness, the original model achieved low decision overhead by validating against a lightweight knowledge base for schedule reuse. The second implementation, known as "Similarity-AI-Hybrid," was equipped with similarity matching and multi-objective fitness. The computational costs of "fuzzy" pattern matching were demonstrated by the second implementation, which had a low 13% knowledge base schedule reuse rate and a high decision overhead (23 ms) while achieving a task duration of 41 seconds. Similarly, a third iteration that used machine learning also experienced higher latency (234.58 ms), but successfully utilized Online Learning to automate 54% of scheduling decisions via predictive modeling.

In conclusion, even though resource awareness has benefits, it was found that adding advanced intelligence either through a machine learning technique or a similarity search increases decision latency to high levels (20 – 235 ms). The most reliable and practical way to smartly distribute the work is to combine a Genetic Algorithm that knows system's resources with a fast, simple system that reuses past solutions, while ML integration remains viable for complex scenarios where decision automation outweighs latency costs.

VI. FUTURE WORK

For the purpose of improving the robustness, scalability and usability of the system, several strategic improvements have been identified that can be implemented in the future. First, the current testing should be expanded to include real-world managed Kubernetes services, such as EKS and GKE, specifically using heterogeneous node pools to evaluate and validate performance in diverse environments. This infrastructure improvement or expansion must also include advanced resource modeling to support accelerators like GPUs and topology-aware placement. To solve the high decision latency observed in the ML-augmented model, the framework will need to adopt to lightweight neural architectures and hardware accelerators to reduce the overhead, while also exploring Deep Reinforcement Learning (DRL) for dynamic state management. In this regard, the optimization scope will be expanded to include a multi- objective fitness function that accounts for energy consumption, network latency, and SLA compliance. Finally, strict scalability testing should be conducted with over 100 concurrent pods and dynamic task arrival patterns to ensure the stability of the system under high-demand scenarios.

Link: https://colab.research.google.com/drive/1kutaa- o8R48zL_pLU7fTMYvtFcs5ccdn?usp=sharing

VII. REFERENCE

[1] M. Vachhani, K. Patel, Z. Patel, M. Patel, and D. Garg, “Enhancing cloud computing efficiency: Dynamic and predictive resource allocation and load balancing strategies,” in *Proc. Int. Conf. IoT Based Control Netw. Intell. Syst. (ICICNIS)*, 2024, pp. 16–20. doi: 10.1109/ICICNIS64247.2024.10823380.

[2] R. Kalaiselvi, N. S. N, D. R, J. M, S. K. R. C, and M. Suresh, “AI powered load balancer in cloud computing,” in *Proc. 3rd Int. Conf. Adv. Electr. Electron. Commun. Comput. Autom. (ICAEECA)*, 2025, pp. 1–6. doi: 10.1109/ICAEECA63854.2025.11012236.

[3] S. K. Bothra, D. P. Bhatt, and A. K. Sharma, “A novel hybrid genetic algorithm for deadline based cost optimization in cloud environment,” in *Proc. 4th Int. Conf. Ubiquitous Comput. Intell. Inf. Syst. (ICUIS)*, 2024, pp. 281–286. doi: 10.1109/ICUIS64676.2024.10866630.

[4] B. Sahu, S. K. Swain, Sudheer Mangalampalli, and S. Mishra, “Multiobjective Prioritized Workflow Scheduling in Cloud Computing Using Cuckoo Search Algorithm,” *Applied Bionics and Biomechanics*, vol. 2023, pp. 1–13, Jul. 2023, doi: https://doi.org/10.1155/2023/4350615.

[5] I. Agarwal, S. Gupta, and Ravi Shankar Singh, “A novel hybrid algorithm for workflow scheduling in cloud,” *International Journal of Cloud Computing*, vol. 12, no. 6, pp. 605–620, Jan. 2023, doi:https://doi.org/10.1504/ijcc.2023.134648.

[6] S. Mogal and V. Sonaje, “Dynamic load balancing in cloud computing environments: a genetic algorithm approach,” *Comput. Electr. Eng.*, vol. 113, p. 109321, 2024.

[7] S. K. Bothra, S. Singhal, and H. Goyal, “Hybrid JAYA algorithm for workflow scheduling in cloud,” *Naučno- tehničeskij Vestnik Informacionnyh Tehnologij, Mehaniki i Optiki*, vol. 23, no. 2, pp. 313–322, Apr. 2023, doi: https://doi.org/10.17586/2226-1494-2023-23-2-313-322.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

[8] A. Sharma and K. K. Sharma, "Cloud Computing: Hybrid Load Balancing Algorithm Proposal," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 10s, pp. 859–864, 2023.

[9] Q.-H. Chen and C.-Y. Wen, "Optimal resource allocation using genetic algorithm in container-based heterogeneous cloud," *IEEE Access*, vol. 12, pp. 12345– 12356, 2024.

[10] K. Singh and P. Kumar, "Dynamic load balancing in cloud computing using AI techniques," *Journal of Cloud Computing*, vol. 12, no. 1, pp. 45-59, 2023.

[11] K. Singh and P. Kumar, "Dynamic load balancing in cloud computing using AI techniques," *Journal of Cloud Computing*, vol. 12, no. 1, pp. 45-59, 2023.

[12] R. Padhy and S. K. Pani, "Optimizing task load distribution in cloud environments via dynamic genetic algorithm," in *Proc. IEEE Int. Conf. Emerg. Smart Technol. Comput. Commun. (ESIC)*, 2024, pp. 1–6.

[13] J. Liu and Q. Zhao, "AI-powered cloud performance management: A framework for adaptive load balancing," *IEEE Access*, vol. 11, pp. 22333-22345, 2024.

[14] S. Patel and M. Verma, "AI-based optimization in dynamic load balancing for cloud computing," *International Journal of Computer Applications*, vol. 182, no. 2, pp. 78-85, 2023.

[15] Y. Wu, H. Tang, and L. Zhou, "Optimization algorithms for dynamic workload distribution in cloud environments," *Cloud Computing Journal*, vol. 16, no. 3, pp. 200-218, 2023.

[16] F. Jamal and T. Siddiqui, "An optimized algorithm for resource utilization in cloud computing based on the hybridization of meta heuristic algorithms," *Int. J. Inf. Technol.*, pp. 1–10, 2023.

[17] A. R. Khan, "Dynamic Load Balancing in Cloud Computing: Optimized RL-Based Clustering with Multi- Objective Optimized Task Scheduling," *Processes*, vol. 12, no. 3, 2024. doi: 10.3390/pr12030519.

[18] L. I. Qian and W. Xue, "Modified Artificial Bee Colony Algorithm for Load Balancing in Cloud Computing Environments," *Int. J. Adv. Comput. Sci. Appl.*, vol. 15, no. 5, 2024.

[19] S. Sobhanayak, "MOHBA: multi-objective workflow scheduling in cloud computing using hybrid BAT algorithm," *Computing*, Apr. 2023, doi: https://doi.org/10.1007/s00607-023-01175-9.

[20] G. Kaur and M. Kalra, "Cost effective hybrid genetic algorithm for scheduling scientific workflows in cloud under deadline constraint," *International Journal of Advanced Intelligence Paradigms*, vol. 24, no. 3/4, p. 380, 2023, doi: https://doi.org/10.1504/ijaip.2023.129185.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)