International Journal of Computational Research in Science and Technology
E-ISSN: 3108-0871, Volume 1 Issue 2
Nov-Dec 2025, PP 46-54

Job Load Balancing of Edge Computing Devices by Trained Mathematical Models

Saurabh Jain[1], Mahesh kr. Sharma[2]

[1]Department of Electronic and Communication, MITRC, Alwar, Rajasthan India

[2]Information Technology Arya college of Engineering and Information Technology, Jaipur, Rajasthan, India

Abstract. Edge computing reduces latency by bringing computation closer to end devices, but the growing scale and heterogeneity of edge networks make resource management increasingly complex. Load balancing is essential for efficient resource use and low response times, yet static approaches struggle in dynamic environments. In this paper, a novel load-balancing model is proposed in which task sequences are first generated using a trained mathematical model, and each generated sequence is further optimized using genetic algorithms. Use of genetic algorithm has increases the efficiency as dynamic situation is manage by the algorithm. Experiments are conducted across various environments, and the results demonstrate that the artificial immune-based model outperforms the group search algorithm in terms of overall performance.

Keywords: Data Mining, Edge Computing, Machine learning, Job Load Balancing, Resource Utilization.

I. Introduction

To support modern Internet communities, the Internet of Everything (IoE) interconnects a vast number of intelligent communication devices and integrates them across diverse and heterogeneous networks [1]. This connectivity enables the development of many critical applications, including smart buildings, surveillance systems, target tracking, and industrial automation. A typical wireless sensor network (WSN) consists of thousands of small, low-cost sensor nodes that operate with limited processing power, communication capability, memory, and energy resources [2]. Despite these constraints, such technologies serve as valuable data sources for networked systems and continuously generate large volumes of data.

Every day, the Internet of Things (IoT) produces and collects enormous amounts of sensor data [1]. Processing this data efficiently has become a major challenge, as transmitting it to centralized cloud servers often results in increased latency and performance bottlenecks [2]. To address this issue, cloud computing and fog/edge computing have emerged as effective solutions for managing large-scale data processing in IoT environments [3].

Edge computing refers to computational resources deployed at the edge of the network, closer to data-generating devices such as sensors and mobile systems. By processing data near its source, edge computing significantly reduces latency, enhances security, lowers operational costs, and improves system reliability and scalability. For these reasons, edge computing is well suited for handling the massive data volumes generated by IoT applications [4].

However, edge nodes operate under strict resource constraints, including limited processing capacity, memory, and energy. When these limits are exceeded, edge devices can become overloaded, leading to performance degradation. Therefore, effective load-balancing mechanisms are essential in edge computing to distribute workloads evenly and prevent resource exhaustion across edge nodes.

To ensure efficient load balancing, the proposed approach incorporates detailed knowledge of edge server resources, including processing capability, memory availability, and disk capacity. By considering these factors, the system can make well-informed decisions that account for the heterogeneous nature of the infrastructure. To enhance system stability and reliability, a queue-based mechanism is employed in which each edge server maintains its own task queue. This design helps prevent task loss while minimizing resource contention among servers.

The remainder of the paper is structured as follows. Section 2 reviews related studies in the area of load balancing. Section 3 presents the proposed learning-based load-balancing approach for coordination between cloud and edge servers. Section 4 discusses the performance evaluation and experimental results. Finally, Section 5 concludes the paper and highlights key findings.

II. Literature Survey

F. Zhang et al. [5] proposed an AxTD3-based deep reinforcement learning approach to achieve effective load balancing while minimizing system latency. In their work, each edge server is assumed to host multiple virtual machines, and a workload distribution strategy is introduced to evenly balance computation across these virtual machines. The overall system is modeled as a reinforcement learning framework, where the state and action spaces are carefully analyzed. To reduce computational complexity without affecting performance, the authors simplify the state and action representations. Furthermore, they introduce models, which divide a neural network into multiple parallel subnetworks. This design significantly reduces the size of the state and action spaces, resulting in faster learning convergence.

Dong et al. [6] focused on the dynamic operational conditions of edge-layer computing nodes in IoT environments, particularly CPU and memory utilization. Based on these

real-time resource states, tasks are dynamically assigned to either edge or cloud nodes. Their optimization-based load-balancing strategy aims to minimize overall task completion time by adapting to changing system conditions.

Li et al. [7] considered both static and dynamic parameters for load balancing in edge computing systems. Static parameters include hardware characteristics such as physical memory capacity, CPU frequency, number of processor cores, and disk size, while dynamic parameters involve real-time CPU and memory usage. Using these combined metrics, their optimization-based approach categorizes edge nodes into different workload states—light, normal, or heavy—and assigns tasks to the most appropriate nodes to improve execution efficiency. Similarly, the proposed EdgeUP model in our study incorporates dynamic resource information, such as CPU and memory utilization, to support effective load-balancing decisions.

T. Li et al. [8] formulated the load-balancing scheduling problem as a multi-objective optimization task and developed a Distributional Reinforcement Learning (DRL) framework to address it. Their approach incorporates quantile regression to learn the distribution of cumulative rewards during the scheduling process, enabling more accurate decision-making. Based on this model, they proposed a dynamic load-balancing scheduling algorithm. To support training and evaluation, the authors also built a cluster-based environment capable of real-time batch job processing, which simulates realistic job arrival patterns for training the DRL-based scheduling agent.

An et al. [9] introduced a hybrid evolutionary algorithm that integrates real-time order acceptance with condition-based preventive maintenance to create robust scheduling solutions. However, most existing approaches, including this one, are primarily evaluated using standard benchmark datasets such as the Lawrence or Brandimarte instances. These datasets differ significantly from real-world industrial environments, limiting the practical applicability of the results.

A. N. Khan et al. [10] presented an enhanced learning-assisted task scheduling approach based on a Criticality and Collapse Aware Scheduling (CCAS) strategy. The proposed framework consists of two key components: a task scheduling module that considers task criticality and system collapse awareness, and an ensemble prediction model based on Gradient Boosting Decision Trees (GBDT) for proactively estimating machine utilization and task execution safety. By learning task-related features, the ensemble model provides high-level abstractions for predicting system states. Additionally, an intelligent scheduling mechanism is designed to optimize resource allocation and maximize production in resource-constrained smart manufacturing networks. Extensive experiments and comparative evaluations against the traditional Rate Monotonic (RM) algorithm demonstrate the effectiveness of the proposed method.

III. Proposed Methodology

This section explains the proposed job load-balancing model, in which the transformation of incoming job loads into an optimized job sequence is illustrated through the block diagram shown in Fig. 1. Each job is evaluated based on its fundamental resource requirements, such as processor execution time, memory usage, data transmission bandwidth, and communication cost. Using these parameters, a trained prediction

model estimates the resource demands, after which a genetic algorithm is applied to optimize the job execution sequence. Overall, the proposed framework follows three main stages: feature extraction, machine learning–based prediction, and sequence optimization.

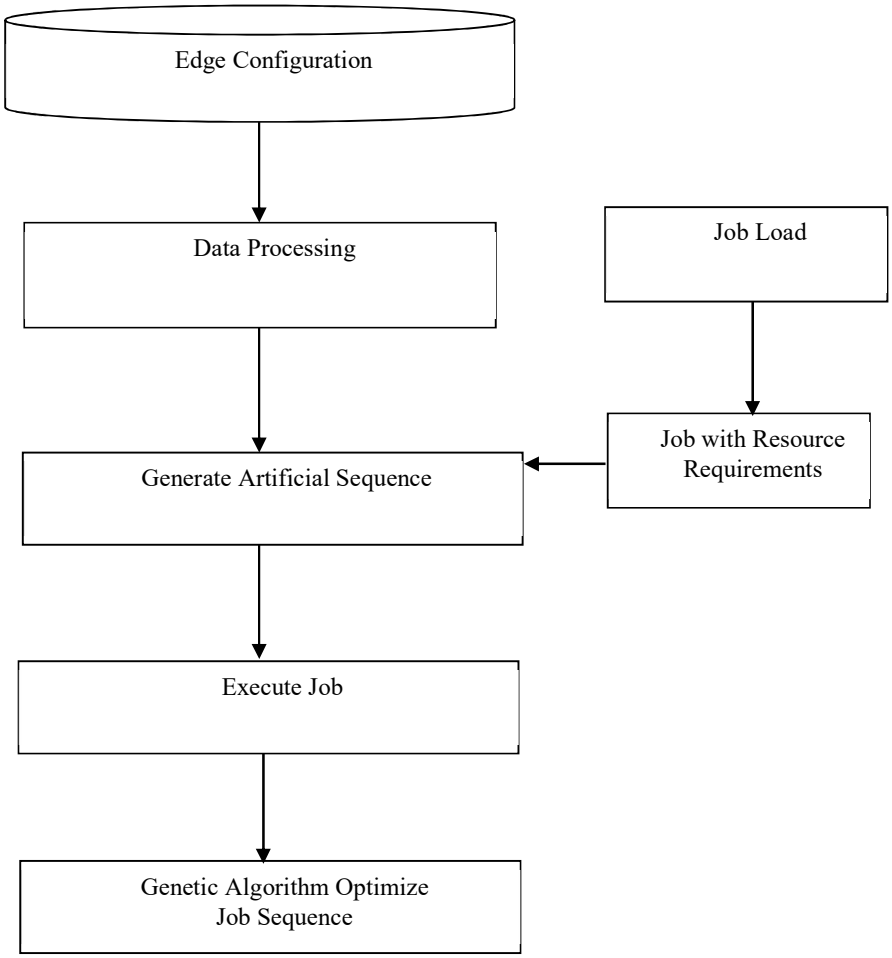

Fig. 1 Block diagram of proposed model.

Feature Collection In this study, two distinct sets of features are considered: one related to edge devices and the other related to the incoming jobs [11]. Although the features are generally common in nature, they are categorized based on their source.

Job Features: Each input job is characterized by four resource-related parameters: maximum memory requirement (Rm), processor requirement (Rp), bandwidth requirement (Rb), and communication cost (Rc).

$$J \rightarrow \{Rm, Rp, Rb, Rc\}$$

Edge Features: Similarly, each edge device is defined by its available resource limits, including maximum memory (ERm), processing capability (ERp), bandwidth (ERb), and communication cost (ERe).

$$E \rightarrow \{ERm, ERp, ERb, ERe\}$$

These job and edge parameters are combined to form the job sequence feature set (JSF), which is used for training the prediction model.

$$\mathbf{JSF} \leftarrow \{ Rm, Rp, Rb, Rc, ERm, ERp, ERb, Ere\}$$

Machine learning model

In this work, diverse job requirements are generated for IoT devices, and corresponding execution sequences are created for edge nodes. An mathematical model is trained to learn and generate effective job sequences for IoT tasks [12]. The trained mathematical model is then used to initialize the population for the Elephant Herd Optimization (EHO) process [13]. Unlike many optimization approaches that rely on random or Gaussian-based population initialization, the proposed method exploits learned job patterns from previous sequences to guide the search process more effectively.

The mathematical training input consists of IoT resource attributes such as processor usage, memory demand, bandwidth requirement, and related parameters. In addition, the model also considers the average resource availability at edge nodes. These two categories of features jointly contribute to training the model, where the input is represented by the Job Sequence Feature (JSF). The desired output, referred to as the Desired Job Sequence (DJS), corresponds to the assignment order of jobs to edge devices.

$$MM \leftarrow Mathematical\_Model(JSF, DJS)$$

Generate Elephant Using the trained mathematical model, an initial population of IoT job sequences is generated. To ensure diversity among candidate solutions, the generated job sequences are shuffled before being passed to the optimization stage.

$$JSP \leftarrow Population(IJ, E)$$

Fitness The fitness function evaluates the quality of each job sequence produced during the Elephant Herd Optimization process. In this study, the makespan of a job sequence is used as the fitness metric. A sequence with a shorter makespan is considered more efficient, as it reflects better workload distribution and reduced overall execution time. Consequently, the job sequence achieving the minimum makespan in a given iteration is selected as the best solution.

Clan Updating (Position Update)

In the Elephant Herd Optimization approach, solution updates are guided by the fitness of each elephant within the population, directing the search toward the most promising job sequence, referred to as the best solution JSb [13]. Once the leading elephant (representing the optimal job sequence) is identified, selected elements of other job sequences are randomly adjusted. This process moves weaker solutions closer to the structure of the best-performing sequence, allowing them to inherit its beneficial characteristics and gradually improve their quality.

$$JSP \leftarrow Caln\_Update(JS_b, JSP)$$

Mutation To maintain diversity and avoid premature convergence, a separation mechanism is applied in which certain jobs within a sequence are randomly exchanged with others from the job pool. This controlled randomization introduces exploration into the search process by allowing new job arrangements to emerge. The number of job exchanges may vary, typically involving one to three swaps, depending on the population structure and iteration stage.

$$JSP \leftarrow Separation(JSP)$$

Update Population The clan updating and separation operations are executed repeatedly over a predefined number of iterations. After each iteration, the fitness of the newly

generated job sequences is evaluated. If a newly generated sequence demonstrates better performance than its predecessor, it replaces the older solution in the population; otherwise, the original sequence is retained. After completing a fixed number of iterations, the job sequence with the highest fitness (best affinity) is selected as the final output of the proposed model.

IV. Evaluation Parameters

The experimental study was conducted under various environmental conditions, where the number of edge nodes and the IoT job sequences were modified. The proposed model was implemented using MATLAB software on a system equipped with a 12th-generation Intel i3 processor and 8GB of RAM. To evaluate its performance, the model was compared with PBSM, [14]. The comparison was carried out using the existing Preference-Based Stable Matching (PBSM) model proposed in [18].

Results

Table1 Makespan-based comparison of edge load balancing models.

Edges	PBSM	Proposed Model
8	158.02	156.21
16	158.47	155.34
24	159.12	154.97
32	158.05	154.72
40	156.94	155.68

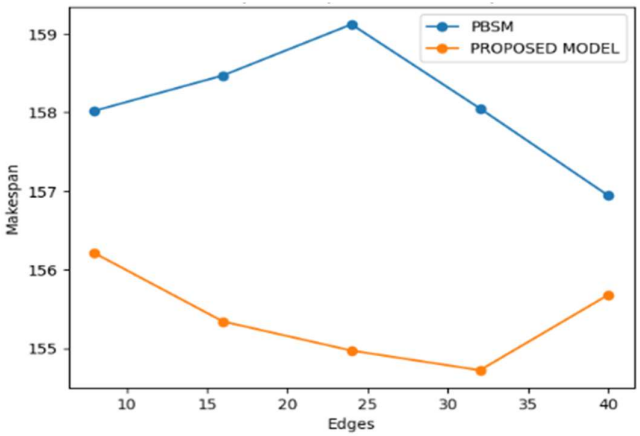

Fig. 2 Comparison of makespan parameter of different models.

Table 1 compares the makespan, i.e., the total time required to complete all tasks, under different numbers of edge nodes. As the number of edges increases, both models exhibit relatively stable makespan values. However, fig. 2 shows that Proposed Model consistently achieves lower makespan than PBSM, indicating more efficient task distribution and reduced completion time across edge nodes. This demonstrates the effectiveness of the proposed Proposed Model approach in minimizing overall execution delay.

International Journal of Computational Research in Science and Technology
E-ISSN: 3108-0871, Volume 1 Issue 2
Nov-Dec 2025, PP 46-54

Table 2 Total Flow Time-based comparison of edge load balancing models.

Edges	PBSM	Proposed Model
10	15740	15620
20	15790	15530
30	15910	15505
40	15790	15485
50	15680	15610

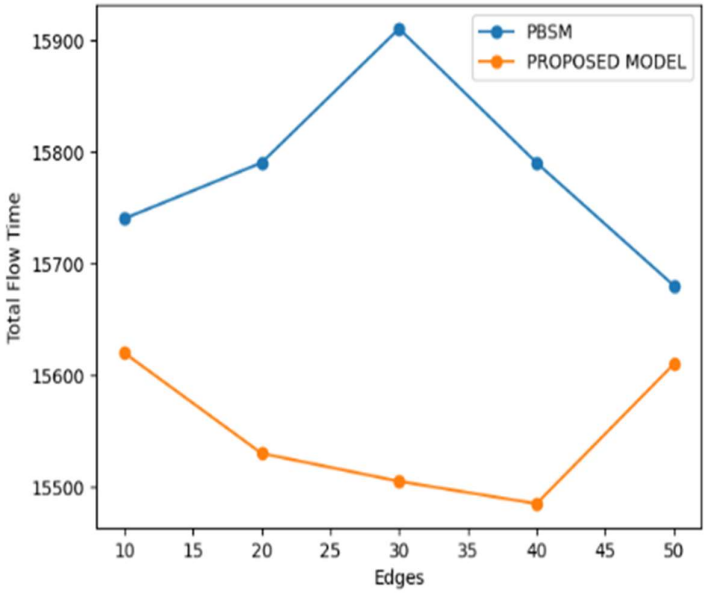

Fig. 2 Comparison of total flow time parameter of different models.

Table 2 Total flow time represents the sum of completion times of all tasks. Lower values indicate faster system responsiveness. Fig. 2 shows that Proposed Model consistently produces lower total flow time compared to PBSM, especially as the number of edges increases. This suggests that Proposed Model effectively schedules tasks with reduced waiting time, thereby improving overall system throughput.

Table 3 Edge utilization-based comparison of edge load balancing models.

Edges	PBSM (%)	Proposed Model (%)
10	94.8	95.7
20	89.5	90.1
30	84.6	86.2
40	79.4	81.3
50	71.6	74.2

Table 3, shows Edge utilization value that estimate how efficiently the available edge resources are used. As the number of edges increases, utilization gradually decreases due to workload distribution across more nodes. However, proposed model consistently achieves higher utilization than PBSM, reflecting its superior ability to evenly distribute tasks and reduce idle resources within the edge computing environment.

Table 4 Execution time-based comparison of edge load balancing models.

Edges	PBSM (s)	Proposed Model (s)
10	0.0668	0.0621
20	0.0856	0.0559
30	0.0712	0.0654
40	0.0781	0.0689
50	0.0954	0.0647

Execution time refers to the actual processing time required to execute tasks. The results demonstrate that Proposed Model generally achieves lower execution time compared to PBSM, particularly at higher edge counts. This improvement highlights the efficiency of the proposed model in reducing processing overhead and improving real-time performance in edge computing scenarios.

V. Conclusion

This work proposed a learning-assisted edge load balancing model that integrates job–edge feature collection with Elephant Herd Optimization (EHO) for efficient IoT task scheduling. Job requirements and edge resource attributes were combined into a unified Job Sequence Feature (JSF) set, enabling informed and adaptive task assignment. A trained mathematical model was used to generate high-quality initial job sequences, improving the optimization process compared to random initialization methods. Experimental evaluation demonstrated that the proposed model consistently outperforms the Preference-Based Stable Matching (PBSM) approach. The results show an average 1.8–2.7% reduction in makespan, indicating faster overall task completion. Similarly, total flow time was reduced by approximately 1.5–2.6%, reflecting improved system responsiveness and lower job waiting time. Edge utilization improved by about 1.5–3.5%, confirming more balanced workload distribution and reduced resource idleness.

References

1. G. Zhu, J. Xu, K. Huang, and S. Cui, “Over-the-air computing for wireless data aggregation in massive IoT,” IEEE Wireless Communications, vol. 28, no. 2, pp. 57–65, 2021.
2. S. Shukla, M. F. Hassan, D. C. Tran, R. Akbar, I. V. Paputungan, and M. K. Khan, “Improving latency in Internet-of-Things and cloud computing for real-time data transmission: A systematic literature review,” Cluster Computing, pp. 1–24, 2021.

3. S. Kumar, P. Tiwari, and M. Zymbler, “Internet of Things is a revolutionary approach for future technology enhancement: A review,” Journal of Big Data, vol. 6, art. no. 111, 2019.
4. U. Sivarajah, M. M. Kamal, Z. Irani, and V. Weerakkody, “Critical analysis of big data challenges and analytical methods,” Journal of Business Research, vol. 70, pp. 263–286, 2017.
5. F. Zhang et al., “Adaptive load balancing for industrial edge computing systems: An AxTD3-deep reinforcement learning approach,” IEEE Transactions on Network Science and Engineering, vol. 13, pp. 4743–4759, 2026.
6. Y. Dong, G. Xu, Y. Ding, X. Meng, and J. Zhao, “A ‘Joint-Me’ task deployment strategy for load balancing in edge computing,” IEEE Access, vol. 7, pp. 99658–99669, 2019.
7. G. Li, Y. Yao, J. Wu, X. Liu, X. Sheng, and Q. Lin, “A new load balancing strategy by task allocation in edge computing based on intermediary nodes,” EURASIP Journal on Wireless Communications and Networking, vol. 2020, art. no. 3, 2020.
8. T. Li, S. Ying, Y. Zhao, and J. Shang, “Batch jobs load balancing scheduling in cloud computing using distributional reinforcement learning,” IEEE Transactions on Parallel and Distributed Systems, vol. 35, no. 1, pp. 169–185, Jan. 2024.
9. Y. An, X. Chen, J. Zhang, and Y. Li, “A hybrid multi-objective evolutionary algorithm to integrate optimization of production scheduling and imperfect cutting tool maintenance considering total energy consumption,” Journal of Cleaner Production, vol. 268, Sep. 2020.
10. A. N. Khan, N. Iqbal, A. Rizwan, S. Malik, R. Ahmad, and D. H. Kim, “A criticality-aware dynamic task scheduling mechanism for efficient resource load balancing in constrained smart manufacturing environment,” IEEE Access, vol. 10, pp. 50933–50946, 2022.
11. R. Sumathy, S. F. Sohail, S. Ashraf, S. Y. Reddy, S. Fayaz, and M. Kumar, “Next word prediction while typing using mathematical modeling,” in Proc. 8th Int. Conf. on Communication and Electronics Systems (ICCES), Coimbatore, India, 2023, pp. 167–172.
12. M. Tishin, C. X. Mavromoustakis, and J. Mongay Batalla, “Machine learning methods in tasks load balancing between IoT devices and the cloud,” IEEE Access, vol. 12, pp. 133726–133733, 2024.
13. J. Li, H. Lei, A. H. Alavi, and G.-G. Wang, “Elephant herding optimization: Variants, hybrids, and applications,” Mathematics, vol. 8, no. 9, art. no. 1415, 2020.
14. A. Bandyopadhyay et al., “EdgeMatch: A smart approach for scheduling IoT-edge tasks with multiple criteria using game theory,” IEEE Access, vol. 12, 2023.