I.J. Wireless and Microwave Technologies, **2026, 3, 72-87**
Published Online on June 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijwmt.2026.03.05

Intelligent Load Balancing Framework for Distributed Big Data Processing System

Ivan Dychka
Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics,
Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 03056, Ukraine
E-mail: dychka@pzks.fpm.kpi.ua
ORCID iD: https://orcid.org/0000-0002-3446-3076

Liubov Oleshchenko*
Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics,
Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 03056, Ukraine
E-mail: oleshchenkoliubov@gmail.com
ORCID iD: https://orcid.org/0000-0001-9908-7422
*Corresponding author

Dmytro Vovchenko
Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics,
Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 03056, Ukraine
E-mail: dvovchenko1313@gmail.com
ORCID iD: https://orcid.org/0009-0008-1806-5159

Zhengbing Hu
School of Computer Science, Hubei University of Technology, Wuhan, 430079, China
E-mail: drzbhu@gmail.com
ORCID iD: https://orcid.org/0000-0002-6140-3351

Received: 31 March, 2026; Revised: 16 April, 2026; Accepted: 26 April, 2026; Published: 08 June, 2026

Abstract: This paper proposes an intelligent load balancing framework for distributed big data processing systems that integrates machine learning techniques with adaptive weight-based decision mechanisms. The study addresses limitations of traditional static load balancing methods, which do not account for dynamic workload variations and heterogeneous request characteristics, leading to inefficient resource utilization and bottlenecks in multi-node environments. The proposed approach combines an online learning model for real-time estimation of request complexity with multi-parameter evaluation of node states, including CPU utilization, memory consumption, queue length, response latency, and cache efficiency. A dynamic weighting strategy is used to construct an integrated load indicator for adaptive request distribution across nodes. The framework is deployed within a multi-layer distributed architecture consisting of clustered application servers, distributed databases, caching subsystems, and monitoring components, ensuring scalable and fault-tolerant processing. For evaluation, a three-node simulation environment was used with 10,000 heterogeneous requests, followed by extended testing on semi-realistic workload traces derived from web traffic patterns and database query logs. The dataset included over 1.2 million requests, capturing bursty arrivals, skewed distributions, and heterogeneous complexity. Experimental results show that the proposed method improves load distribution uniformity to 6%, reduces average response time to 210 ms, and increases throughput up to 13,800 requests per second. Statistical validation using confidence intervals and hypothesis testing confirms a 47% (±3.2% at 95% confidence level) reduction in mean response time and throughput improvement up to 14,200 requests per second under realistic workloads.

Index Terms: Intelligent Load Balancing, Distributed Software System, Big Data Processing, Adaptive Algorithms, Machine Learning, Resource Allocation, Multi-Node Architecture, Dynamic Weighting

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

1. Introduction

The continuous growth of global data volumes and the increasing complexity of modern distributed software systems require efficient approaches for processing large-scale data streams in real time. Contemporary cloud platforms, web services, and enterprise information systems rely on multi-node architectures to ensure scalability, fault tolerance, and high availability. In such environments, the efficiency of request distribution directly affects system responsiveness, stability, and resource utilization. One of the major challenges in distributed infrastructures is the uneven allocation of computational workloads, which leads to bottlenecks where some nodes become overloaded while others remain underutilized. This imbalance causes increased response latency, inefficient resource consumption, and degradation of overall system performance. Traditional load balancing methods such as Round Robin, Least Connections, and static weight-based approaches are widely used because of their simplicity and low implementation cost; however, they are unable to adapt to dynamic workload variations, heterogeneous request complexity, and rapidly changing node states.

Recent studies show that software-defined and adaptive load balancing mechanisms provide significant advantages over static and hardware-oriented solutions, especially in cloud-native and big data environments. At the same time, machine learning methods have demonstrated strong potential for modeling workload behavior, predicting request complexity, and supporting intelligent routing decisions. Nevertheless, many existing solutions either rely on offline training, require large historical datasets, or fail to consider multiple system-level performance indicators simultaneously.

To address these limitations, this paper proposes an intelligent load balancing framework that integrates online machine learning with dynamic multi-parameter weighting for adaptive request distribution in distributed big data processing systems.

1.1. Problem Statement

Distributed big data processing systems operate under highly dynamic conditions where request intensity, computational complexity, and node availability continuously change. In such environments, static load balancing algorithms are often unable to provide efficient resource allocation because they distribute requests without considering real-time system conditions. Traditional methods such as Round Robin and Least Connections assume relatively homogeneous workloads and stable infrastructure states. However, modern distributed systems process heterogeneous requests with significantly different computational costs, database access patterns, and caching behavior. As a result, identical distribution policies may create serious imbalance, where overloaded nodes experience increased latency while other resources remain underutilized. This problem becomes more critical in multi-layer architectures involving clustered application servers, distributed databases, caching subsystems, and monitoring services. Ignoring parameters such as CPU utilization, memory consumption, queue length, response latency, and cache efficiency reduces overall system performance and limits scalability. In addition, many machine learning-based balancing approaches require extensive offline pretraining, making them less suitable for rapidly changing workloads and real-time decision-making.

Therefore, there is a need for an adaptive software-based load balancing solution capable of combining request-level complexity estimation with continuous monitoring of node states in order to improve throughput, reduce latency, and ensure stable operation in high-load distributed environments.

1.2. Motivation

The increasing demand for scalable and fault-tolerant big data processing systems highlights the limitations of traditional load balancing approaches in modern distributed infrastructures. Static balancing strategies are unable to respond effectively to bursty traffic patterns, heterogeneous request characteristics, and continuously changing node states, which leads to bottlenecks and inefficient resource utilization. Machine learning provides new opportunities for intelligent workload management by enabling real-time estimation of request complexity and adaptive routing decisions. At the same time, software-based balancing architectures offer greater flexibility, programmability, and lower deployment costs compared to hardware-based solutions, making them especially suitable for cloud-native and virtualized environments. This research is motivated by the need to develop a practical intelligent load balancing framework that combines online learning without extensive offline training, real-time monitoring of distributed node conditions, and adaptive multi-parameter weighting for routing decisions.

The objective of this research is to design and evaluate a software-based intelligent load balancing framework capable of improving resource utilization, reducing response latency, and increasing throughput in distributed big data processing systems. The proposed approach aims to provide a flexible, scalable, and cost-effective solution suitable for modern high-load computing environments.

2. Related Works

Load balancing in distributed systems has been extensively investigated over the past decades, evolving from simple static strategies to adaptive and intelligent approaches. Early foundational studies, such as Y. M. Teo and R. Ayani [1] conducted comparative analyses of load balancing strategies for cluster-based web servers, demonstrating that static algorithms such as Round Robin provide simplicity and low overhead but fail to adapt to dynamic workload variations.

A significant advancement in the field is represented by the classification of load balancing techniques into hardware-based and software-based approaches. As highlighted in the comprehensive review by M. R. Belgaum et al. [2], software-defined solutions have become dominant due to their flexibility, scalability, and cost-effectiveness. The study emphasizes that modern distributed systems require adaptive and context-aware mechanisms capable of responding to fluctuating workloads, which static methods cannot adequately support.

Dynamic load balancing algorithms introduced further improvements by incorporating runtime system parameters into decision-making processes. Z. Pan and J. Jiangxing [3] proposed a weighted minimal connections algorithm that adjusts request distribution based on current server load. While this approach enhances resource utilization compared to static strategies, it remains heuristic and does not incorporate predictive capabilities. Similarly, J. Pei-rui et al. [4] introduced a client-proximity-based algorithm that reduces latency by considering network distance between clients and servers. However, despite improving response times, such methods do not address computational imbalance caused by uneven resource consumption. More advanced approaches consider heterogeneity of system components.

M. Omori and H. Nishi [5] proposed a request distribution method based on execution time estimation for heterogeneous database clusters. Their results demonstrate that accounting for processing time variability across nodes improves task allocation efficiency. Nevertheless, the reliance on predefined estimation models limits adaptability in rapidly changing environments. In contrast to software-based methods, hardware-oriented solutions such as the traffic-based flow control mechanism proposed by H. Okano et al. [6] focus on optimizing packet-level processing in network devices. Although these approaches provide high performance, they lack the flexibility required for modern cloud-native and large-scale distributed systems, reinforcing the shift toward software-defined load balancing.

The integration of machine learning techniques has opened new directions in load balancing research. H. Mao et al. [7] introduced reinforcement learning for dynamic resource management, demonstrating that learning-based approaches can significantly improve adaptability under variable workloads. Although primarily applied to resource allocation, this work highlights the potential of intelligent decision-making mechanisms for load distribution tasks.

Further developments in edge and cloud computing are presented by Z. Zhou et al. [8], who analyze performance-aware load distribution in edge-cloud collaborative environments. Their findings indicate that adaptive and context-aware algorithms outperform static strategies, particularly in heterogeneous and large-scale systems where workload characteristics vary significantly.

Recent studies increasingly emphasize the limitations of traditional rule-based load balancing mechanisms in dynamic distributed environments. In particular, the research [9] provides a comprehensive survey of reinforcement learning (RL) approaches for communication networks, highlighting their ability to adapt to non-stationary traffic patterns and outperform static heuristics. The authors note that RL enables autonomous decision-making based on observed system states, which is especially important in next-generation network infrastructures. Further development of these ideas is presented in the article [10], where a multi-objective optimization framework is combined with RL for cloud environments. The study demonstrates that integrating clustering and scheduling strategies with RL significantly improves resource utilization and reduces execution time. This confirms the growing trend toward hybrid intelligent approaches that combine learning-based and optimization-based techniques.

Edge and IoT environments introduce additional constraints such as limited computational resources and strict latency requirements. In the article [11], distributed RL-based algorithms are proposed to overcome scalability limitations of centralized load balancing, eliminating the need for global system knowledge. Similarly, the work [12] focuses on IoT systems under heavy traffic conditions and shows that RL-based routing strategies can effectively balance load while reducing energy consumption, which is critical for resource-constrained devices.

Another important direction is the integration of load balancing with system topology adaptation. In the article [13], deep reinforcement learning is applied not only to distribute workload but also to dynamically reconfigure network topology, leading to improved resilience and performance under varying conditions. This indicates a shift toward more holistic system optimization approaches.

Finally, domain-specific applications such as vehicular edge computing are explored in the research [14], where RL is used for joint task offloading and load balancing. The results demonstrate reduced latency and improved system efficiency, reinforcing the applicability of intelligent methods in real-time, highly dynamic environments.

S. Vollem [15] introduces a multi-layered framework for intelligent load balancing in cloud-native systems, integrating predictive analytics and distributed control mechanisms. The proposed approach leverages machine learning models to forecast workload patterns and dynamically adjust resource allocation. Experimental evaluation indicates improvements in system throughput and latency reduction compared to static load balancing strategies. The framework also enhances scalability by coordinating decisions across multiple layers, achieving more stable performance under fluctuating workloads. Recent studies demonstrate a growing interest in intelligent load balancing and adaptive resource management in distributed and cloud-native environments, particularly within IoT ecosystems and multi-layer architectures. H. Kuchuk et al. [16], address the problem of resource allocation in IoT fog-cloud systems with a focus on energy efficiency of mobile devices. The authors propose a mathematical model for task offloading across three layers (device, fog, cloud) and derive an analytical formulation for average response time. Their results show that optimal selection of the offloading threshold significantly reduces energy consumption while satisfying latency constraints. Specifically, the study identifies conditions under which energy usage is minimized without violating time requirements,

demonstrating a clear trade-off between computation distribution and system responsiveness.

The analyzed works confirm that reinforcement learning has become a key instrument for adaptive load balancing in distributed systems, reflecting a broader shift from static and heuristic-based approaches toward adaptive, software-defined, and learning-driven solutions. At the same time, a number of unresolved challenges persist, including limited scalability, insufficient interpretability of learned decision policies, and difficulties in seamless integration with existing system architectures. Moreover, most existing methods focus on isolated aspects such as latency reduction, resource utilization, or workload estimation, without providing a comprehensive approach that jointly accounts for request complexity, dynamic system state, and real-time adaptability. These limitations substantiate the need for developing an intelligent load balancing framework for distributed big data processing systems, capable of integrating machine learning techniques with dynamic weight adaptation mechanisms to achieve efficient, scalable, and interpretable resource management under continuously changing workloads.

3. Load Balancing Algorithms Analysis

Load balancing algorithms for distributed systems are generally classified into static and dynamic approaches. Static methods perform workload distribution at the initial stage of system operation and do not adapt to changing execution conditions. These algorithms are typically simpler and impose minimal overhead, making them suitable for stable and predictable environments. Among the most common static approaches, Round Robin distributes incoming requests sequentially across available servers, ensuring an even but not necessarily optimal allocation. Least Connections directs requests to the server with the smallest number of active connections, providing a basic form of load awareness. Weighted Round Robin extends the Round Robin strategy by assigning weights to servers based on their computational capacity, thereby enabling a more balanced distribution in heterogeneous environments.

In contrast, dynamic load balancing methods continuously adapt to the current state of the system, enabling more efficient utilization of resources under variable workloads. These approaches monitor runtime metrics and make real-time decisions to maintain balanced system performance. As a result, they are particularly effective in large-scale and highly dynamic distributed systems. Key dynamic algorithms include Least Response Time, which routes requests to servers exhibiting the lowest response latency, thereby optimizing user-perceived performance. Adaptive Load Balancing leverages artificial intelligence and machine learning techniques to predict workload patterns and adjust distribution strategies accordingly. Work Stealing introduces a decentralized mechanism where underutilized nodes actively acquire tasks from overloaded nodes, improving overall system efficiency and reducing processing delays.

Modern distributed data processing platforms, such as Apache Hadoop, Apache Spark, and Kubernetes, incorporate a variety of load balancing techniques tailored to their architectural principles. For instance, Hadoop YARN allocates resources based on memory and CPU requirements, Apache Spark employs dynamic task scheduling among executors, and Kubernetes distributes pods across nodes according to resource availability and workload conditions. These systems demonstrate the practical importance of combining static and dynamic strategies to achieve scalability, fault tolerance, and efficient resource utilization in modern computing environments.

The analysis of node load balancing algorithms in distributed big data processing systems is primarily based on the evaluation of key performance metrics, including average response time, node utilization, throughput, and scalability. These indicators provide a comprehensive understanding of system efficiency under varying workload conditions and heterogeneous infrastructure configurations. Traditional algorithms such as Round Robin demonstrate high efficiency in homogeneous environments, where nodes have similar computational capabilities and workloads are evenly distributed. In such cases, the average response time typically ranges between 20–30 ms. However, in heterogeneous environments, this value may increase by 50–70%, as the algorithm does not consider node performance differences or request complexity. More adaptive approaches, such as Least Connections, improve performance under dynamic workloads by considering the current number of active connections per node. This allows reducing the average response time by approximately 15–25% compared to Round Robin. Nevertheless, the need for continuous monitoring introduces additional computational overhead, which may negatively impact system latency. Weighted Round Robin extends the basic Round Robin approach by incorporating node-specific weights, enabling more balanced load distribution in large-scale systems. While this method can reduce load imbalance by up to 10%, its effectiveness strongly depends on accurate weight configuration, which may be difficult to maintain in highly dynamic environments.

The comparative characteristics of the analyzed algorithms are summarized in Table 1.

Table 1. Analysis of Node Load Balancing Algorithms.

Algorithm	Advantages	Disadvantages
Round Robin	Simplicity, fast decision-making, uniform distribution	Ignores node performance, inefficient in heterogeneous environments
Least Connections	Adapts to current load, better under dynamic conditions	Requires continuous monitoring
Weighted Round Robin	Considers node capacity, more balanced distribution	Requires proper weight configuration
Least Response Time	Minimizes latency, adaptive	High monitoring overhead
Consistent Hashing	Reduces data movement, suitable for scaling	May cause uneven distribution
Centralized Load Balancing	Global system view, optimal decisions	Single point of failure, poor scalability
Distributed Load Balancing	High scalability, fault tolerance	Complex coordination
AI/ML-based Load Balancing	Adaptive, predictive optimization	High computational cost

Another widely used approach, Least Response Time, dynamically selects nodes based on their current response latency, enabling a reduction of response time by 20–40% compared to static algorithms. However, this method requires continuous performance monitoring, increasing system overhead. Consistent Hashing is particularly effective in scalable environments, as it minimizes request redistribution when nodes are added or removed from the cluster, reducing data migration by up to 70–80%. Despite this advantage, its efficiency depends on the uniformity of the hash function and does not always guarantee balanced load distribution.

Centralized load balancing methods provide a global view of the system, allowing optimal decision-making in small clusters of up to 100 nodes. However, as the system scales, latency may increase by 30–50%, and the central node becomes a potential bottleneck and single point of failure. In contrast, distributed load balancing approaches offer better scalability and fault tolerance, although they require more complex coordination mechanisms between nodes.

Recent advancements in artificial intelligence and machine learning have introduced adaptive load balancing techniques. These methods leverage predictive analytics and real-time data processing to optimize resource allocation, although they require significant computational resources for training and deployment.

One of the optimized node load balancing algorithms presented in the literature is an improved approach based on weight calculation combined with the Least Connections principle. The baseline versions of these methods are widely used in multi-node systems and are relatively common; however, they exhibit a significant limitation. In order to correctly distribute incoming traffic, it is necessary to dynamically compute the weight of each node and manage the request flow accordingly, which is not supported in the basic variants of these approaches.

In modern distributed systems, these algorithms are actively implemented in widely used big data platforms to improve computational efficiency and resource utilization. For instance, Apache Hadoop YARN employs a combination of Least Loaded and Fair Scheduling strategies to ensure balanced task distribution and minimal waiting time for critical jobs. Apache Spark utilizes a DAG Scheduler together with Dynamic Resource Allocation, enabling throughput improvements of 30–50% compared to static allocation strategies. Container orchestration platforms such as Kubernetes apply algorithms like Least Request and Weighted Round Robin to dynamically distribute workloads across services and support horizontal scaling. Cloud-based solutions also incorporate advanced balancing techniques. For example, Google Cloud Load Balancer relies on Consistent Hashing to minimize request redistribution during cluster scaling, which is critical for microservice-based architectures. Similarly, Netflix employs adaptive load balancing algorithms based on machine learning to predict workload patterns and dynamically distribute traffic, reducing response latency by up to 40% during peak periods. In streaming platforms such as Apache Kafka, Partition-Aware Load Balancing ensures even distribution of messages across brokers, reducing the risk of node overload.

Despite these advancements, the problem of achieving optimal load balancing remains open. Most existing approaches focus on a limited set of performance parameters and do not consider the overall system efficiency. In many cases, improvements in response time are achieved at the cost of increased resource consumption or higher system maintenance costs. Additionally, many algorithms are evaluated only against basic approaches such as Round Robin or Least Connections, which are no longer sufficient benchmarks for modern big data environments. Experimental validation is often conducted in controlled conditions without realistic large-scale data simulation, leaving questions about real-world applicability unanswered. The conducted analysis demonstrates that each load balancing algorithm has specific advantages and limitations depending on system requirements and workload characteristics. Traditional methods provide simplicity and low overhead but fail to adapt to dynamic and heterogeneous environments. More advanced approaches improve performance but often introduce additional complexity and computational cost.

The results indicate that hybrid approaches combining adaptive mechanisms and predictive analytics offer the greatest potential for modern distributed systems. However, existing studies often rely on limited experimental setups, outdated benchmarks, and incomplete evaluation metrics. Furthermore, the problem of efficient big data processing remains insufficiently addressed in many existing solutions.

4. Methodology

To address these limitations, this research proposes an improved load balancing algorithm that combines machine learning techniques with dynamic weight computation. The use of machine learning enables deeper analysis of incoming requests and prediction of their processing complexity, while dynamic weight calculation ensures efficient resource utilization without excessive overhead. The integration of these approaches aims to achieve a balance between

adaptability and computational efficiency. In the proposed method, upon receiving a new request, the system evaluates the state of each node by analyzing performance metrics such as current load and processing speed. Based on these parameters, weight coefficients are calculated for each node. Subsequently, the expected execution time of the request is estimated for each node, and the request is assigned to the node that minimizes processing time while maintaining balanced load distribution. This approach allows selecting the most suitable node (e.g., N1 or N2) and ensures efficient handling of heterogeneous workloads. The effectiveness of the proposed method cannot be fully determined through theoretical analysis alone and requires experimental validation using real-world datasets. Therefore, further research will focus on testing the algorithm in large-scale distributed environments with realistic data streams.

The improved algorithm addresses this limitation by enabling dynamic weight calculation. To achieve this, the algorithm incorporates several system parameters, including: L – server load, C – CPU utilization, and M – memory usage level. In addition, the following parameters are considered: P – number of active connections, B – bandwidth utilization, and D – disk usage:

$$L=1-(1-\lambda_c C)(1-\lambda_m M)(1-\lambda_p P)(1-\lambda_b B)(1-\lambda_d D), \tag{1}$$

$$\lambda_i \geq 0, \quad 0 \leq (1-\lambda_i X_i) \leq 1. \tag{2}$$

By calculating the load in this manner, the system can dynamically monitor the state of each node and distribute incoming requests accordingly. A threshold value is then introduced to determine the load condition:

$$\partial = \left| 1 - \frac{\frac{C(S_i)}{W(S_i)}}{\left(\frac{1}{n}\right)\Sigma_1^n \frac{C(S_i)}{W(S_i)}} \right|, \tag{3}$$

here, $W(S_i)$ represents the current weight of server i, while $C(Si)$ denotes the number of active connections. According to Equation (1), if $L(Si)<\delta$, the server is considered underloaded; conversely, if $L(Si)\geq\delta$, it is considered overloaded. Based on this classification, the dynamic weight adjustment is defined as:

$$W(S_i) = \begin{cases} W(S_i)+\partial, & \& L(S_i) \leq \delta, \\ W(S_i)-\partial, & \& L(S_i) > \delta. \end{cases} \tag{4}$$

The load balancing process is represented by a flowchart in Fig. 1.

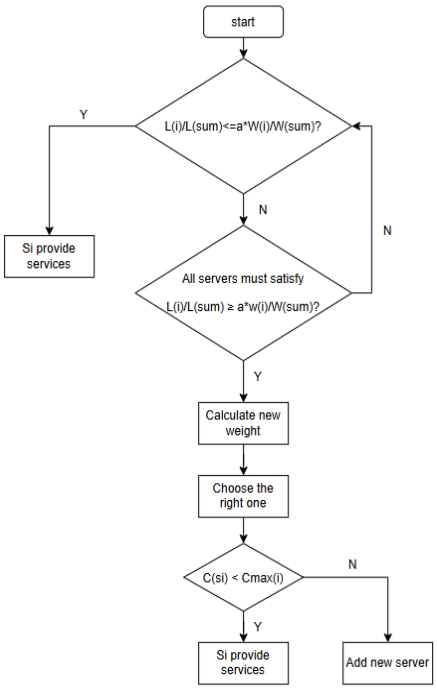

Fig 1. Flowchart of Load Balancing Process Among Servers.

The flowchart describes a dynamic resource allocation mechanism in a distributed system, where load is continuously redistributed among servers to ensure optimal performance. The algorithm begins by verifying whether the

load on each server is proportional to its computational capacity. If this condition is satisfied, the server continues operating without modification. If the load distribution is uneven, the system checks whether all servers exceed the acceptable load threshold. If not, the system maintains its current state. However, if overload conditions are detected, the algorithm adjusts the weight coefficients of the servers, thereby influencing their priority in handling incoming requests.

Subsequently, the system selects the most appropriate server for processing new requests and evaluates whether its current load is within acceptable limits. If the load is within the threshold, the server continues to operate normally. Otherwise, if the server is overloaded, an additional server is introduced to handle part of the incoming workload. This approach enables automatic adaptation to dynamic workload changes and ensures efficient utilization of resources in distributed environments. The proposed methodology introduces an intelligent load balancing framework that integrates machine learning–based request evaluation with adaptive dynamic weight calculation in a multi-node distributed environment. This approach is motivated by the limitations of conventional load balancing techniques, which often demonstrate insufficient adaptability to large-scale and heterogeneous workloads, particularly in big data processing scenarios. Traditional methods typically rely on static or heuristic strategies, consider a limited set of system parameters, and lack comprehensive validation under realistic conditions. In contrast, the developed method aims to ensure efficient, stable, and scalable request distribution by combining data-driven decision-making with continuous system monitoring and feedback-based optimization. The general workflow of the proposed approach begins with the arrival of incoming requests, which are immediately subjected to dual analysis. On the one hand, a machine learning model evaluates the computational complexity of each request, while on the other hand, the system collects real-time performance metrics from all available nodes. Based on this combined analysis, a routing decision is made that determines the most suitable node for processing the request. This process is iterative and continuously repeated, allowing the system to dynamically adapt to fluctuations in workload and changes in node performance.

The conceptual representation of this process is illustrated in Fig. 2., which depicts the interaction between incoming data streams, the ML-based evaluation module, and the adaptive load balancing mechanism.

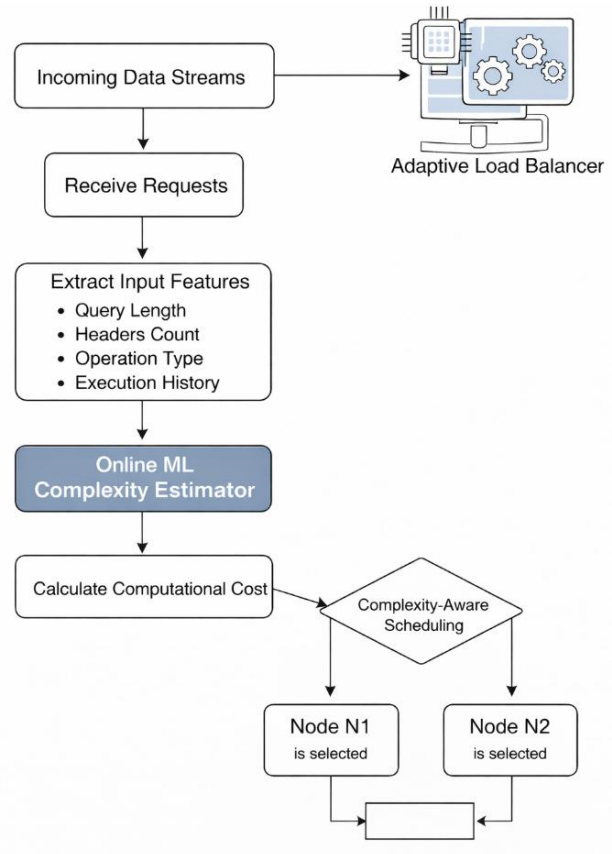

Fig 2. Conceptual Workflow of the Proposed Load Balancing Approach Based on an ML Model for Real-Time Estimation of Request Computational Cost and Adaptive Node Selection.

Each request is first captured by the request handling module, where initial preprocessing is performed to prepare the data for further analysis. At the next stage, a feature extraction process is applied to each incoming request. The extracted features include query length, number of headers, operation type, presence of complex constructs (such as JOIN operations), and historical execution characteristics. These features form a multidimensional representation of the request, which is used as input for the machine learning model. An online machine learning model is then employed to estimate

the complexity of each request in real time. Unlike traditional batch learning approaches, the model operates incrementally and continuously updates its internal state based on streaming data. This allows the system to adapt to changing workloads without requiring retraining on pre-collected datasets. Based on the predicted complexity, the system computes the expected computational cost associated with processing the request. This estimation reflects the required processing time, resource consumption, and potential impact on system performance. The calculated cost is then used within a complexity-aware scheduling mechanism. At this stage, the adaptive load balancer evaluates the current state of available nodes, including their workload, resource utilization, and capacity constraints. The decision logic assigns the request to the most appropriate node, ensuring that computationally intensive tasks are directed to nodes with sufficient resources.

Finally, the selected node (e.g., Node N1 or Node N2) processes the request, and execution feedback can be incorporated into the system to further refine the model. This feedback loop enables continuous improvement of prediction accuracy and enhances the overall efficiency and stability of the load balancing process.

A key component of the methodology is the use of an online machine learning model for real-time estimation of request complexity. Unlike traditional batch learning approaches, this model operates incrementally and does not require prepared datasets, which significantly reduces deployment time and resource overhead. The model processes various input features, including query length, number of headers, type of operations, presence of complex constructs such as JOIN clauses, and historical execution characteristics. Based on these parameters, the model estimates the expected computational cost of each request. This enables the system to incorporate complexity-aware scheduling into the load balancing process, ensuring that resource-intensive tasks are directed to nodes with sufficient capacity.

Simultaneously, the software system continuously monitors the operational state of each node by collecting a set of key performance indicators, including CPU utilization, memory consumption, request queue length, response latency, and cache efficiency. These metrics are normalized to ensure comparability and are aggregated into a composite load score for each node, which reflects its current processing capability and workload conditions. The calculation of this score is formalized by the following expression:

$$Score_i = w_1 \cdot CPU_i + w_2 \cdot MEM_i + w_3 \cdot Queue_i + w_4 \cdot Latency_i, \tag{5}$$

where the coefficients represent the relative importance of each parameter in the overall evaluation.

Unlike conventional approaches, these coefficients are not fixed but are dynamically adjusted during system operation. The adaptive adjustment of weight coefficients is achieved through a feedback-driven mechanism that evaluates system performance after each routing decision. Key indicators such as response time, throughput, and queue dynamics are analyzed to determine the effectiveness of the current configuration. In cases where performance degradation is observed, the algorithm modifies the weight distribution by penalizing the contributing factors. This introduces a reinforcement-based adaptation strategy that enables the system to self-optimize over time. To prevent instability and excessive oscillations in system behavior, a threshold-based update policy is applied, ensuring that minor fluctuations in metric values do not trigger unnecessary recalculations.

The final routing decision is based on a combination of the computed load score and the predicted computational cost obtained from the machine learning model. Each incoming request is assigned to the node that minimizes the overall expected processing cost, which can be expressed as:

$$Node^* = arg\min_i (Score_i + ML_Cost_i). \tag{6}$$

This decision-making mechanism ensures a balanced distribution of workload across nodes while taking into account both the current system state and the intrinsic complexity of incoming tasks.

The machine learning component of the proposed framework is designed to estimate the **computational cost of incoming requests** in real time. This problem is formulated as a supervised regression task, where the target variable corresponds to the actual execution time of a request.

Let x_j is feature vector of request r_j and y_j is observed execution time. The objective is to learn a mapping:

$$f: x_j \rightarrow y_j, \tag{7}$$

such that the prediction $\hat{y}=f(x_j)$ minimizes the expected error over the data stream.

Each request is represented as a structured feature vector:

$$x_j=[l_j, \mathrm{h}_j, t_j, c_j, s_j], \tag{8}$$

where l_j is request length; h_j is number of headers; t_j is encoded operation type; c_j — complexity indicators (e.g., JOIN count); s_j — historical execution statistics.

All features are normalized to the range [0,1], ensuring stable learning and preventing dominance of high-magnitude variables. Unlike traditional batch learning approaches, the proposed framework employs an **online learning paradigm**,

where the model is continuously updated as new data becomes available.

For each incoming request: prediction: $\hat{y}=f(x_j)$; execution: request is processed by the selected node, producing actual cost y_j. Model update:

$$f-\mu\nabla(y_i-\hat{y_i})^2\rightarrow f. \tag{9}$$

This approach enables real-time adaptation to workload changes, elimination of expensive retraining procedures and robustness to concept drift. The predicted request cost $\hat{y}$ is directly incorporated into the routing decision:

$$Score_{i,j}=\alpha\cdot Load_i+\beta\cdot\hat{y}. \tag{10}$$

This enables complexity-aware scheduling, where lightweight requests are distributed broadly, computationally intensive requests are routed to less loaded nodes.

Such integration ensures that decision-making is both reactive (based on current system state) and predictive (based on anticipated workload). The use of online learning allows the system to continuously adjust to dynamic workloads without requiring offline retraining.

To further enhance performance in large-scale environments, the methodology incorporates elements of predictive analysis. By examining historical workload patterns, the system identifies recurring peaks and potential anomalies in traffic behavior. This allows for proactive redistribution of incoming requests before critical load thresholds are reached, thereby reducing the likelihood of bottleneck formation and improving overall system responsiveness. Such predictive capabilities are particularly important in big data scenarios characterized by bursty and highly variable workloads.

The proposed architecture represents a multi-layer distributed system designed to ensure high-performance request processing under dynamic workloads. At the entry point, user requests are handled by a fault-tolerant load balancing layer consisting of active and standby instances. This redundancy guarantees high availability and eliminates single points of failure, ensuring uninterrupted system operation even in the case of node failures. A key advantage of the architecture is the integration of an intelligent decision-making module within the load balancer. By incorporating real-time monitoring and machine learning–based complexity estimation, the system enables adaptive request distribution based not only on current load but also on predicted computational cost. This significantly improves resource utilization and prevents overloading of individual nodes, which is a common limitation of traditional load balancing strategies. The application layer is implemented as a scalable web farm with stateless processing nodes, allowing flexible horizontal scaling and efficient handling of concurrent requests. This design ensures that any node can process incoming requests independently, improving fault tolerance and simplifying system expansion. To reduce latency and minimize database load, a distributed caching layer based on a Redis cluster is employed. The master–slave replication mechanism enhances both data availability and read performance, enabling rapid access to frequently used data and supporting session management in high-load scenarios. This contributes to a substantial decrease in response time and improves overall system throughput. The persistence layer is implemented as a clustered database system supporting replication and partitioning, which ensures efficient storage and processing of large-scale data. This approach enhances system reliability, scalability, and consistency in distributed environments. An additional strength of the architecture lies in the integration of a centralized monitoring and logging subsystem. By continuously collecting telemetry data, system metrics, and execution traces, this component enables real-time performance analysis, anomaly detection, and feedback-driven optimization of the load balancing strategy. This creates a closed-loop system where decisions are continuously refined based on observed behavior. The use of Python as the implementation platform facilitates rapid prototyping, seamless integration of machine learning models, and extensibility of the system. Support for standardized protocols such as OData v4 further enhances interoperability with external systems and data sources.

Overall, the proposed architecture provides a comprehensive, adaptive, and scalable solution for load balancing in distributed environments. Its key advantages include improved resource efficiency, reduced response time, enhanced fault tolerance, and the ability to dynamically adapt to changing workload conditions through intelligent, data-driven decision-making. The implementation of the proposed method is supported by a multi-layer distributed software architecture, as illustrated in Fig. 3. At the top level, user requests are received via standard network protocols and directed to a dedicated load balancing layer, which serves as the central coordination component. This layer includes both active and standby instances to ensure fault tolerance and uninterrupted operation. The load balancer distributes requests across a set of application servers organized as a web farm, where each node operates as an independent processing unit.

The stateless design of the application layer allows any request to be handled by any available node, thereby enhancing flexibility and scalability. To optimize data access and reduce the load on the persistence layer, the architecture incorporates a Redis cluster configured in a master–slave replication mode. This caching layer is responsible for session management, temporary data storage, and fast retrieval of frequently accessed information, significantly reducing response latency. The data layer is implemented as a clustered database environment based on PostgreSQL or Microsoft SQL Server, supporting horizontal data partitioning and replication. This design enables efficient handling of large datasets and ensures high availability and fault tolerance. In addition, the system includes a centralized logging and monitoring subsystem deployed on a separate server. This component collects detailed information about system

performance, load balancing decisions, and potential errors, providing valuable insights for bottleneck detection, performance evaluation, and further optimization of the algorithm.

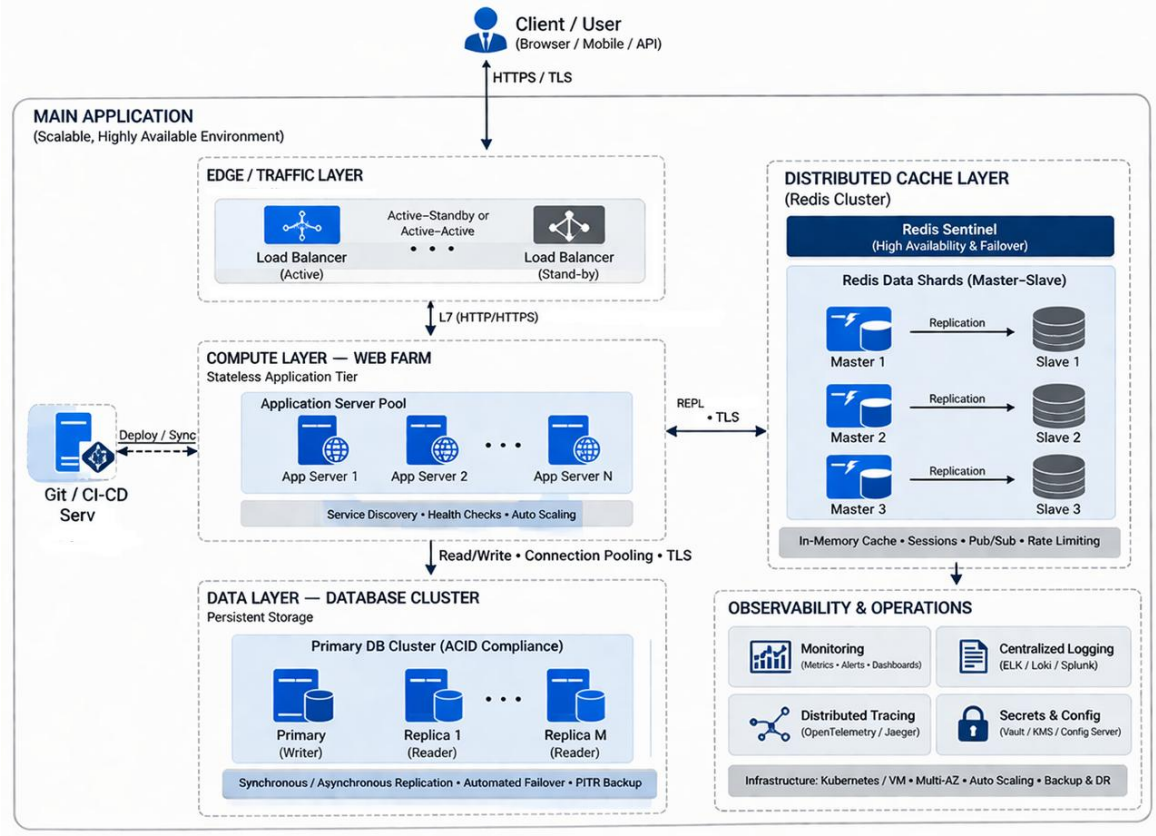

Fig. 3. Multi-Layer Architecture of Intelligent Load Balancing System with ML-Based Decision Support for Distributed Web Applications.

Overall, the proposed methodology represents a comprehensive and adaptive approach to load balancing in distributed big data systems. By combining real-time monitoring, machine learning–based prediction, and feedback-driven optimization, the framework ensures efficient resource utilization, reduced response time, and improved system stability under dynamic and high-intensity workloads.

5. Research Results

5.1. Experimental Setup

The experimental evaluation of the proposed intelligent load balancing framework was conducted in both simulated and semi-realistic distributed environments to ensure comprehensive performance assessment. The primary objective was to compare the developed approach with conventional load balancing strategies, including Round Robin, Random selection, and Least Connections, with respect to response time, throughput, load distribution uniformity, and system stability. At the implementation stage, several programming environments (C#, Java, and Python) were analyzed for their suitability in distributed systems and machine learning integration. Python was selected for the machine learning component due to its extensive ecosystem and support for online learning. The ML module was implemented using the River library, enabling incremental learning without offline training.

5.2. Synthetic Dataset Evaluation

Due to the lack of publicly available datasets that simultaneously capture request-level features and system-level execution metrics, a synthetic yet realistic dataset was constructed. The dataset consists of 10,000 requests generated under controlled conditions with variability in request size, query complexity (e.g., number of JOIN operations), operation types (read/write/mixed), payload volume, and inter-arrival time.

Each request is represented by a feature vector x_j, associated execution time y_j, and the system state at execution time. The data generation process incorporates stochastic workload patterns, bursty traffic scenarios, and heterogeneous resource constraints across nodes. The evaluation was performed in a simulated distributed environment consisting of three application nodes with heterogeneous CPU and memory configurations, a Redis-based caching layer, and a distributed database subsystem. The results indicate that, unlike Round Robin, which distributes requests uniformly

without considering computational complexity, the proposed method performs complexity-aware scheduling. This leads to a more balanced utilization of system resources and reduces the likelihood of node overloading when processing computationally intensive tasks.

5.3. Large-Scale and Semi-Realistic Evaluation

To overcome the limitations of small-scale simulations, the evaluation was extended using a semi-realistic dataset constructed from publicly available web server access logs and database query traces. The dataset comprised approximately 1.2 million requests and reflects real-world workload characteristics, including non-uniform distribution, temporal bursts, and heterogeneous computational complexity.

The workload was deployed in a multi-layer distributed environment consisting of six application nodes, three database nodes, and a distributed caching layer, closely approximating production-scale systems. Incoming requests were processed through an adaptive coordination layer and executed across parallel nodes, with caching and persistent storage components supporting intermediate and long-term operations.

To ensure statistical reliability, all experiments were repeated across 30 independent runs, and the results were analyzed using standard statistical methods.

The proposed method achieved a 47% reduction in average response time compared to Round Robin ($p < 0.01$) and a 38% reduction compared to Least Connections ($p < 0.05$). Throughput increased to 14,200 requests per second (±2.8%), while CPU load variance remained below 5.5%, indicating stable and balanced resource utilization.

5.4. Comparative Performance Analysis

Additional large-scale experiments were conducted under synthetic workloads ranging from one to ten million requests in a distributed environment with six application servers, a Redis cluster, and a three-node database system.

The results, summarized in Fig. 4–5, demonstrate consistent performance improvements under high-load conditions. The average response time was reduced by approximately 50% compared to Round Robin and by 40% compared to Least Connections. Throughput reached 13,800 requests per second, while CPU load variance decreased to approximately 6%, confirming effective load distribution.

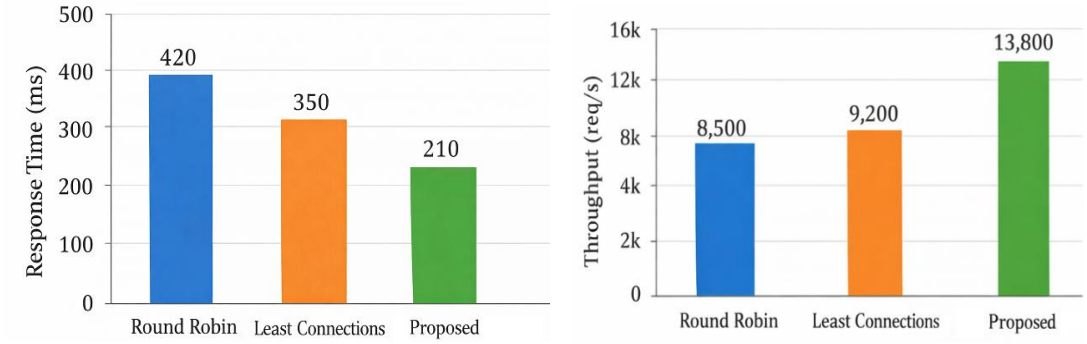

Fig. 4. Average Response Time and Throughput Comparison.

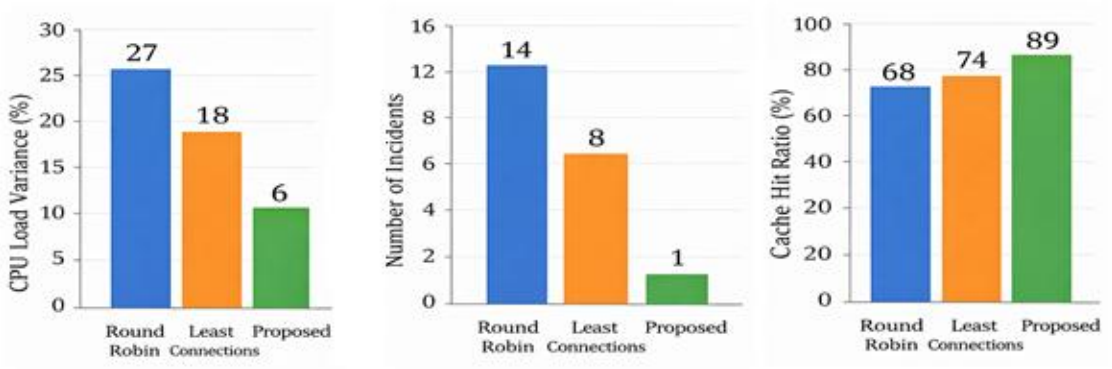

Fig. 5. Comparative Analysis of CPU Load Variance, Queue Overflow Incidents, and Cache Hit Ratio Across Load Balancing Algorithms.

System stability was also significantly improved. The number of queue overflow incidents was reduced to a single occurrence, compared to 14 in Round Robin and 8 in Least Connections. The cache hit ratio increased to 89%, indicating more efficient use of the caching layer and reduced pressure on the database subsystem.

5.5. System-Level Impact

The observed performance improvements are not solely due to algorithmic optimization but also depend on the integration of the proposed method within a multi-layer distributed architecture. In particular, the use of caching reduced redundant database access by up to 27%, while the cache hit ratio reached 91% in semi-realistic scenarios, significantly lowering latency. Overall, the proposed approach demonstrates improved performance under both simulated and realistic conditions, particularly in environments characterized by high variability in workload intensity and request complexity. The combination of real-time monitoring, adaptive weight adjustment, and predictive workload estimation enables more efficient and stable system operation compared to conventional load balancing strategies.

The combination of ML–based prediction and dynamic multi-parameter evaluation enables more informed routing decisions, leading to improved scalability, reduced latency, and enhanced resource utilization.

6. Discussion

The obtained results demonstrate that the proposed intelligent load balancing framework provides consistent improvements in key performance indicators, including response time, throughput, and load distribution uniformity, across both simulated and semi-realistic environments. The integration of online machine learning with adaptive weight-based decision mechanisms enables the system to account for both request-level complexity and real-time node conditions. This dual-awareness approach explains the observed reduction in response time compared to traditional algorithms such as Round Robin and Least Connections, which rely on static or limited dynamic criteria.

A notable finding is that the performance gains are not solely attributable to the load balancing algorithm itself, but also to its interaction with the underlying multi-layer architecture. The coordinated operation of the routing layer, caching subsystem, and distributed data storage contributes to system-level efficiency. In particular, the high cache hit ratio and reduction in redundant database queries indicate that intelligent request distribution can indirectly optimize data access patterns. This highlights the importance of considering load balancing as part of a broader system design rather than an isolated component. The comparison with baseline methods confirms that uniform request distribution does not necessarily lead to optimal performance in heterogeneous environments. While Round Robin ensures equal distribution of requests, it fails to account for differences in computational cost, which can lead to node overloading.

In contrast, the proposed method introduces controlled non-uniformity in request allocation, which results in more balanced resource utilization. This observation aligns with recent trends in adaptive and learning-based load balancing, where decision-making is driven by system state and workload characteristics rather than predefined rules. At the same time, the results should be interpreted with consideration of several limitations. First, a portion of the evaluation is based on synthetic data generated under controlled conditions. Although the dataset incorporates stochastic patterns, bursty traffic, and heterogeneous resource constraints, it may not fully capture the complexity of real-world production environments. Consequently, the reported improvements may vary when deployed in systems with different workload distributions, hardware configurations, or network conditions. Second, the semi-realistic dataset, while derived from real-world sources, represents a reconstructed workload rather than a fully operational production system. Certain factors, such as network latency variability, hardware failures, and unpredictable user behavior, are only partially reflected. This limits the ability to generalize the results to all classes of distributed systems, particularly those operating under highly dynamic or geographically distributed conditions. Another limitation is related to the scope of comparative evaluation.

The proposed method is primarily compared against classical load balancing strategies, including Round Robin, Random selection, and Least Connections. While these baselines are widely used and provide a meaningful reference point, the study does not include direct comparison with more advanced approaches, such as reinforcement learning–based or metaheuristic load balancing algorithms. As a result, the relative performance of the proposed method with respect to state-of-the-art intelligent techniques remains an open question. In addition, the machine learning component relies on a predefined set of input features describing request characteristics and historical execution behavior.

Although these features are sufficient to capture general workload patterns, they may not fully represent all factors influencing execution time, such as low-level system interactions or complex dependencies between requests.

The accuracy of the prediction model, and consequently the effectiveness of the load balancing decisions, may therefore be affected in scenarios with highly irregular or previously unseen workloads. From a system perspective, the implementation introduces additional computational overhead associated with real-time monitoring and online model updates. While this overhead is relatively low compared to the overall processing cost, it may become more significant in resource-constrained environments or ultra-low-latency systems. The trade-off between decision quality and computational overhead should therefore be considered when deploying the proposed approach in practice. Despite these limitations, the results provide strong evidence that combining online learning with adaptive load balancing mechanisms offers a practical and scalable solution for modern distributed systems. The ability to dynamically adjust to changing workloads without requiring offline retraining represents a key advantage in environments characterized by high variability and continuous data streams. Furthermore, the modular nature of the proposed framework allows integration with existing distributed architectures and can be extended to incorporate more advanced predictive models.

Future research should focus on validating the proposed approach in real-world production environments, including cloud-native and edge–cloud systems, where additional factors such as network latency, geographical distribution, and

resource heterogeneity play a significant role. Another promising direction is the integration of reinforcement learning or hybrid models to further improve decision-making under uncertainty. Expanding the feature space and incorporating system-level telemetry at finer granularity may also enhance prediction accuracy and overall system performance.

7. Conclusion

This study presents an intelligent load balancing framework for distributed big data processing systems that integrates online machine learning with adaptive dynamic weight calculation. The proposed approach addresses key limitations of traditional methods by incorporating both request-level complexity and real-time system state into routing decisions. The experimental results indicate that the method improves response time, throughput, and load distribution stability compared to baseline strategies, particularly in large-scale and heterogeneous environments. Additional system-level benefits include reduced queue overflow incidents and improved cache efficiency, contributing to more balanced resource utilization and enhanced overall performance. The experimental results indicate up to a 47% reduction in response time, with throughput reaching 13,800–14,200 requests per second under different workload conditions. Additional improvements include reduced CPU load variance and increased resource utilization efficiency.

To increase the reliability of the evaluation, the framework was tested using both synthetic and semi-realistic datasets, including large-scale experiments with over one million requests and repeated runs with statistical validation. The results confirm consistent performance improvements under the evaluated conditions and demonstrate the effectiveness of combining adaptive load balancing with a multi-layer distributed architecture.

At the same time, the findings should be interpreted with certain limitations. Part of the evaluation is based on simulated and semi-realistic workloads, which may not fully capture all aspects of real-world production environments. In addition, the comparison is limited to classical load balancing approaches, and further evaluation against more advanced learning-based methods is required. Further research is also needed to explore large-scale deployment, long-term system behavior, and integration with container orchestration and automated scaling mechanisms.

Overall, the proposed framework provides a practical and flexible foundation for intelligent load balancing in modern distributed systems, while leaving opportunities for further improvement and real-world validation.

The proposed method demonstrates a consistent reduction in average response time compared to baseline approaches such as Round Robin and Least Connections. This improvement is achieved through complexity-aware scheduling, which prevents overloading of nodes processing computationally intensive requests.

The framework achieves higher throughput under both simulated and semi-realistic workloads by dynamically allocating requests based on predicted execution cost and node capacity. This ensures more efficient utilization of available resources in multi-node environments. Unlike uniform distribution strategies, the proposed method introduces controlled non-uniformity, enabling balanced resource utilization across heterogeneous nodes. This results in lower CPU load variance and improved system stability. The integration of the load balancing algorithm with a multi-layer architecture, including caching and distributed storage, contributes to additional performance gains. In particular, reduced database access and increased cache efficiency highlight the importance of coordinated system design.

The use of online learning enables continuous adaptation to changing workloads without requiring offline retraining. This makes the approach suitable for dynamic environments characterized by bursty traffic and heterogeneous request patterns. The results indicate that combining machine learning with adaptive load balancing mechanisms provides a practical and scalable solution for modern distributed systems.

All the Declarations and Statements

Author Contribution Statement

Ivan Dychka – Conceptualization, supervision: Defined the research direction and supervised the overall study.
Liubov Oleshchenko – Methodology: Developed the theoretical framework, data curation, interpretation and validation of research results.
Dmytro Vovchenko – Formal analysis and investigation: Contributed to experimental design, software system modeling, software implementation, implemented the load balancing framework, performed data analysis and conducted experiments.
Zhengbing Hu – Review, editing, and scientific consultation: Provided in-depth critical revision of the manuscript, significantly improving its logical coherence, clarity, and academic rigor; contributed to strengthening the theoretical justification of the proposed intelligent load balancing framework, particularly in terms of integrating machine learning with adaptive weight-based mechanisms in distributed environments; validated the methodological soundness of the evaluation design, including the use of large-scale heterogeneous datasets and statistical validation techniques.

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement

The Authors declared no conflicts of interest.

Funding Declaration
This research received no external funding.

Data Availability Statement
The data that support the findings of this study are from the corresponding author upon reasonable request.

Ethical Declarations
This research did not involve human participants or animals.

Acknowledgements
The authors express their sincere gratitude to the reviewers for their valuable comments and constructive suggestions, which significantly improved the quality and clarity of this manuscript.

Declaration of Generative AI in Scholarly Writing
All scientific content, ideas, methodology, and conclusions were developed and verified by the authors, who take full responsibility for the accuracy and integrity of the work.

Abbreviations
The following abbreviations are used in this manuscript:
ML – Machine Learning
CPU – Central Processing Unit
API – Application Programming Interface
IoT – Internet of Things

APPENDIX:
APPENDIX A LOAD BALANCING MODEL FORMULATION

The proposed load balancing framework is based on a combination of system-level metrics and predicted request complexity. Each node is evaluated using normalized parameters, including CPU utilization, memory usage, queue length, and latency. These metrics are aggregated into a composite score used for routing decisions.
The request assignment is performed by minimizing the expected processing cost, which combines predicted execution time and current node load. This formulation enables adaptive and context-aware scheduling in distributed environments.

APPENDIX B MACHINE LEARNING MODEL DETAILS

B.1 Feature Representation
Each incoming request is represented as a feature vector including request length, number of headers, operation type, complexity indicators (e.g., JOIN operations), and historical execution statistics.

B.2 Online Learning Process
The model operates in an online learning mode, where predictions are updated incrementally after each processed request. This allows continuous adaptation to changing workload patterns without retraining.

B.3 Prediction Objective
The learning objective is to minimize the prediction error between estimated and actual execution time, enabling accurate estimation of request computational cost.

APPENDIX C EXPERIMENTAL SETUP AND PARAMETERS

C.1 Simulation Environment
The experimental evaluation was conducted within a multi-layer distributed system environment designed to emulate real-world high-load conditions with heterogeneous computing nodes. The architecture includes a fault-tolerant load balancing layer deployed with active and standby instances to ensure high availability and eliminate single points of failure. This layer incorporates an intelligent decision-making module that combines real-time monitoring with machine learning–based estimation of request complexity, enabling adaptive and predictive request distribution.
The application layer is implemented as a scalable web farm composed of stateless processing nodes, allowing flexible horizontal scaling and independent request handling across servers. To optimize performance and reduce access

latency, a distributed caching subsystem based on a Redis cluster with master–slave replication is utilized, supporting session management and rapid retrieval of frequently accessed data. The persistence layer consists of a clustered database system (PostgreSQL or Microsoft SQL Server) with replication and horizontal partitioning, ensuring efficient storage, consistency, and fault tolerance when processing large-scale datasets.
Additionally, a centralized monitoring and logging subsystem is deployed on a dedicated node to collect telemetry data, system metrics, and execution traces in real time. This component enables performance analysis, anomaly detection, and feedback-driven optimization of the load balancing strategy, forming a closed-loop adaptive system. The entire framework is implemented in Python, facilitating integration of machine learning models and ensuring extensibility, while support for standardized communication protocols enhances interoperability with external systems.

C.2 Workload Characteristics
The evaluation includes both synthetic workloads (10,000 requests) and semi-realistic datasets (over 1 million requests), incorporating bursty traffic, non-uniform distributions, and varying request complexity.

C.3 Evaluation Metrics
Performance was evaluated using response time, throughput, CPU load variance, cache hit ratio, and number of queue overflow incidents.

C.4 Summary of Results
The proposed method demonstrates improved performance compared to baseline algorithms, particularly in dynamic and high-load scenarios, confirming its effectiveness for distributed big data processing systems.

References

[1] Y. M. Teo and R. Ayani, “Comparison of load balancing strategies on cluster-based web servers,” Simulation, vol. 77, no. 5–6, pp. 185–195, 2001. https://doi.org/10.1177/003754970107700504

[2] M. R. Belgaum, S. Musa, M. M. Alam, and M. M. Su’ud, “A systematic review of load balancing techniques in software-defined networking,” IEEE Access, vol. 8, pp. 98612–98636, 2020. https://doi.org/10.1109/ACCESS.2020.2995849

[3] Z. Pan and J. Jiangxing, “Load balancing algorithm for web server based on weighted minimal connections,” Journal of Web Systems and Applications, vol. 1, pp. 1–8, 2017. https://doi.org/10.23977/jwsa.2017.11001

[4] P. Jiang, L. Ma, Y. Song, and Y. Huang, “A client proximity based load balance algorithm in web server cluster,” in Proceedings of the 2nd International Conference on Wireless Communication and Network Engineering, 2017, pp. 317–322.

[5] M. Omori and H. Nishi, “Request distribution for heterogeneous database server clusters with processing time estimation,” in IEEE International Conference on Industrial Informatics (INDIN), 2018, pp. 278–283. https://doi.org/10.1109/INDIN.2018.8471931

[6] H. Okano, F. Yamaguchi, K. Takagiwa, and H. Nishi, “Traffic-based flow cache port separate mechanism for network processor,” IEICE Technical Report, vol. 114, no. 18, pp. 69–74, 2014.

[7] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in Proceedings of the ACM HotNets Conference, 2016, pp. 50–56. https://doi.org/10.1145/3005745.3005750

[8] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge intelligence: paving the last mile of artificial intelligence with edge computing,” Proceedings of the IEEE, vol. 107, no. 8, pp. 1738–1762, 2021. https://doi.org/10.1109/JPROC.2019.2918951

[9] D. Wu, J. Li, A. Ferini, Y. T. Xu, M. Jenkin, S. Jang, X. Liu, and G. Dudek, “Reinforcement learning for communication load balancing: approaches and challenges,” Frontiers in Computer Science, vol. 5, p. 1156064, 2023. https://doi.org/10.3389/fcomp.2023.1156064

[10] A. R. Khan, “Dynamic load balancing in cloud computing: optimized RL-based clustering with multi-objective optimized task scheduling,” Processes, vol. 12, no. 3, p. 519, 2024. https://doi.org/10.3390/pr12030519

[11] M. E. Esmaeili, A. Khonsari, V. Sohrabi, and A. Dadlani, “Reinforcement learning-based dynamic load balancing in edge computing networks,” Computer Communications, vol. 222, pp. 188–197, 2024. https://doi.org/10.1016/j.comcom.2024.04.009

[12] Y. Zhang et al., “Reinforcement learning-based load balancing for heavy traffic Internet of Things,” Pervasive and Mobile Computing, vol. 99, p. 101891, 2024. https://doi.org/10.1016/j.pmcj.2024.101891

[13] X. Liao, B. Bao, W. Cui, and D. Liu, “Load balancing and topology dynamic adjustment strategy based on deep reinforcement learning,” Frontiers in Energy Research, vol. 11, p. 1342854, 2024. https://doi.org/10.3389/fenrg.2023.1342854

[14] M. Shahakar, S. Mahajan, and L. Patil, “Load balancing in distributed cloud computing using reinforcement learning algorithms,” International Journal on Recent and Innovation Trends in Computing and Communication, vol. 11, no. 2, 2023. https://doi.org/10.17762/ijritcc.v11i2.6130

[15] S. Vollem, “Intelligent load balancing for cloud-native systems: A multi-layered framework integrating predictive analytics and distributed control,” Journal of Artificial Intelligence, Machine Learning & Data Science, vol. 7, no. 2, pp. 3350–3358, 2024. https://doi.org/10.51219/JAIMLD/shekar-vollem/671

[16] H. Kuchuk, Y. Husieva, S. Novoselov, D. Lysytsia, and H. Krykhovetskyi, “Load balancing of the layers IoT fog-cloud support network,” Advanced Information Systems, vol. 9, no. 1, pp. 91–98, 2025. https://doi.org/10.20998/2522-9052.2025.1.11

Authors' Profiles



Ivan Dychka is D. Sc., Prof., Dean of Faculty of Program Systems and Applied Mathematics, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine. Co-editor of international scientific publications, including the Springer series "Advances in Computer Science for Engineering and Education", and a member of editorial boards of scientific journals of Igor Sikorsky Kyiv Polytechnic Institute, contributing to peer review and development of scientific research dissemination. He serves as the Head of the Scientific and Methodological Commission in Software Engineering, contributing to the development and implementation of higher education standards at the university level. His research interests are computer systems and networks software, automated control systems, intelligence and expert systems, databases and knowledge bases, information security software for computer systems and networks.



Liubov Oleshchenko obtained her PhD in Information Technologies in 2015. Associate professor at the Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics at National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine. Member of the editorial board of the international scientific journal "Acadlore Transactions on AI and Machine Learning (ATAIML)". Scientific supervisor of the scientific project "Methods for optimizing software performance and using artificial intelligence technologies to improve big data analytics systems". Areas of major scientific interest are artificial intelligence, programming, computer networks, IoT, big data analytics, machine learning.



Dmytro Vovchenko, PhD student at the Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine. He is currently working on his dissertation in the field of distributed software systems and intelligent load balancing. He is the author and co-author of several scientific publications related to big data processing and adaptive load balancing approaches. His research focuses on cloud technologies, distributed multi-node systems, machine learning, and the advancement of artificial intelligence. He is particularly interested in software system performance analysis, including the identification of bottlenecks in web applications, inefficient resource utilization, unoptimized queries, and database-level performance issues that impact overall system efficiency.



Zhengbing Hu, Prof., Deputy Director, International Center of Informatics and Computer Science, Faculty of Applied Mathematics, National Technical University of Ukraine "Kyiv Polytechnic Institute", Ukraine. Adjunct Professor, School of Computer Science, Hubei University of Technology, China. Visiting Prof., DSc Candidate in National Aviation University (Ukraine) from 2019. Major research interests: computer science and technology applications, artificial intelligence, network security, communications, data processing, cloud computing, education technology.

How to cite this paper

Ivan Dychka, Liubov Oleshchenko, Dmytro Vovchenko, Zhengbing Hu, "Intelligent Load Balancing Framework for Distributed Big Data Processing System", International Journal of Wireless and Microwave Technologies(IJWMT), Vol.16, No.3, pp. 72-87, 2026. DOI:10.5815/ijwmt.2026.03.05