Kurdistan Journal of Applied Research (KJAR)
Print-ISSN: 2411-7684 | Electronic-ISSN: 2411-7706

Original Article

Check for updates

AI-Based Load Balancing Using Decision Tree Regressor for Parallel Matrix Computation in Cloud Environments

Lina Sarkawt Hiwa [a] *, **Zryan Najat Rashid** [b]

[a] Department of Information Technology, Technical College of Informatics, Sulaimani Polytechnic University, Sulaymaniyah, Iraq.
[b] Department of Computer Networks, Technical College of Informatics, Sulaimani Polytechnic University, Sulaymaniyah, Iraq.

Submitted: 25 June 2025
Revised: 10 July 2025
Accepted: 1 August 2025

*** Corresponding Author:**
lina.sarkawt.tci@spu.edu.iq

Keywords: AI-Based load balancing, Cloud computing, Parallel processing, Machine learning, Decision tree regressor.

How to cite this paper: L. S. Hiwa, Z. N. Rashid, "AI-Based Load Balancing Using Decision Tree Regressor for Parallel Matrix Computation in Cloud Environments", KJAR, vol. 10, no. 2, pp: 43-54, Dec 2025, doi: 10.24017/science.2025.2.4

Abstract: Cloud computing is an evolving technology of current information systems that supports dynamic sharing and elastic provision of resources and services. With increasing demands for computational resources, efficient workload assignment has become an important challenge. Current load balancing methods based on traditional approaches fail to suit dynamic server performance and contribute to the inefficient utilization of available resources, latency, and delays. In response to this challenge, this paper suggests an AI-driven load balancer based on a decision tree regressor to dynamically control task allocation within a parallel cloud system. The system operates to handle computationally heavy tasks, i.e., matrix multiplication, across different servers based on real-time performance measures such as Central Processing Unit (CPU) usage, memory utilization, time of execution, and networking latency. Model training was done with historical data obtained from past executions, incorporated into the web server to facilitate adaptive decision-making. It was tested experimentally with different levels of server scalability as well as matrix complexity. It was contrasted with a static, manual load balancer. All critical performance measures were found to be significantly improved by the AI-based methodology, with the total execution time reduced from 7,060 milliseconds to 1,000 milliseconds; network latency was also reduced to 5.12 ms, down from 214 ms; and the method reduced the overall use of CPU by 33% and overall use of memory by more than 85%. These findings confirm that intelligent, data-driven load balancing offers superior scalability, responsiveness, and efficiency for cloud-based parallel processing systems.

1. Introduction

Cloud computing is a foundational innovation in information technology, evolving from older models like grid, parallel, and distributed computing to deliver scalable on-demand services over the internet [1]. In the field of information technology, cloud computing is considered to be one of the most significant computing concepts of recent times. It developed as an outcome of advancements in existing computing models, including parallel, grid, distributed, and other architectures [2, 3]. Cloud computing provides three basic categories of service to users: Software as a Service, Platform as a Service, and Infrastructure as a Service. End users, who utilize cloud-based applications in everyday business, are the main consumers of Software as a Service [4, 5]. Cloud service providers are responsible for managing and protecting the storage they offer as part of their overall cloud services, which also include computing and networking. Safeguarding and ensuring the integrity of user data is of primary importance [6]. Users have access to an unlimited number of resources and services through cloud platforms. With

increasing data volumes being stored and computed in the cloud, the challenges associated with managing efficient resources have also increased [7].

Parallel computing enables the simultaneous execution of operations by decomposing large computational problems into smaller subtasks that are processed concurrently across multiple processors or nodes. This approach spans multiple levels of parallelism—including bit-level, instruction-level, data-level, and task-level—and is crucial for scaling the matrix computation workloads typical in modern cloud infrastructures [8].

Load balancing, one of the most important of these challenges, is a process used to distribute loads to different computing nodes so then no single server is overwhelmed as others remain idle [9]. Effective load balancing improves system performance through optimizing how resources are used and ensuring the timely delivery of tasks [10]. Traditional static methods often fail to adapt to runtime variability; more recent adaptive strategies leverage machine learning and optimization techniques to dynamically assess and allocate resources based on real-time system metrics [11]. Several load balancing algorithms have been developed to resolve this problem, and each attempt to distribute resources equitably and optimally to cloud tasks. Nevertheless, static methods do not satisfactorily adapt to actual server situations, especially for large systems that have to process millions of user requests concurrently [12]. Researchers are turning to machine learning (ML) and parallel computing to meet these challenges. Intelligent load balancing is provided by machine learning models, especially those with real-time prediction capability. In parallel computing, instructions are executed concurrently, breaking down complex issues into manageable subproblems for simultaneous computation [13]. The four most important types of parallelism—bit-level, instruction-level, data-level, and task-level—offer frameworks for constructing scalable, high-performance systems [14, 15]. The study of parallel computing has grown in importance in computer science and has been essential for the capture of high-performance solutions. The development of multi-core and many-core computer systems toward a larger number of cores confirms that parallelism is the preferred technique for optimizing an algorithm [16, 17].

This paper introduces a novel, lightweight AI-driven load balancing system employing a decision tree regressor, specifically designed for cloud-based matrix computation platforms. The decision tree regressor was chosen for its simplicity, fast prediction speed, and effectiveness with small to medium datasets. Decision tree regression is a supervised learning technique used to estimate continuous target variables by partitioning input feature space into hierarchical segments via training-data-driven splitting criteria. It is valued for its interpretability, computational efficiency, and suitability for small- to medium-sized datasets where real-time prediction performance is important [18, 19]. Unlike more complex models such as deep learning or genetic algorithms, decision trees are easy to interpret, require less computational power, and are well-suited for real-time decision-making in dynamic environments [20]. This makes them a practical solution for load balancing in cloud systems, where quick and efficient responses are essential.

In contrast to the existing approaches that rely on deep learning or theoretical optimization frameworks, our method provides real-time, interpretable, and experimentally validated task allocation based on live performance metrics. To the best of our knowledge, this is the first study to apply a decision tree regressor for parallel matrix multiplication within a real-world cloud infrastructure. The result is a scalable and efficient alternative to both static and computationally intensive AI-based load balancing methods. A decision tree is a significant learning technique that is mostly used for data analysis. It can solve both regression and classification problems, making it a versatile tool for intelligent system design [21]. The proposed scheme uses real-time metrics such as latency, execution time, memory utilization, and Central Processing Unit (CPU) usage to dynamically distribute loads between multiple servers. According to the experimental findings, this methodology significantly exceeds traditional manual balancing techniques in terms of execution speed, system efficacy, resource consumption, and network responsiveness.

The layout of this paper is structured as follows: Section 2 discusses related works on load balancing. Section 3 describes the methodology used to design and implement the AI-driven load balancer based on the decision tree regressor. Section 4 states the experimental design and the results achieved

through simulation and real-world testing. Section 5 discusses and interprets results on system performance and efficiency. Section 6 concludes the paper.

2. Related Works

The difficulty of efficient load balancing in cloud computing has been widely investigated in the current literature, especially with modern data centers under rising pressures to process massive volumes of user demands, constrained resources, and latency-critical operations. Conventional load balancing methods tend to be insufficient in dynamic environments because they fail to respond to changing server performance in real time. Since these shortcomings were identified, several researchers have explored adaptive intelligent methods, such as optimization algorithms and machine learning-based methods, for addressing these shortcomings and improving the overall performance of cloud systems. One of these methods is proposed by Singhal *et al.* [22] who proposed the Rock Hyrax load balancing algorithm for the cloud. It offers a dynamic load distribution between machines based on availability and system load, reducing overall make-span length and energy consumption. While showing an overall decrease between 10–15% in task completion time and 8–13% in energy use, the approach remains based on an established optimization scheme which may restrict its applicability across a variety of workloads and real-time scenarios.

In a parallel line of research, Zhang *et al.* [23] proposed an optimization framework based on machine learning using genetic algorithms and deep learning for scheduling resources in the cloud. Their proposed model remedies drawbacks like skewed task allocation and poor resource utilization. Although the methodology is strong and highlights better system performance, the research is still primarily theoretical, with no experimental evidence in dynamic, real-world environments with system heterogeneity and sudden spikes in load.

Similarly, Pradhan *et al*. [24] proposed Deep Reinforcement Learning with Parallel PSO (DRL-PPSO), which blended deep learning with a particle swarm optimization model to improve scheduling in cloud environments. Based on a series of simulations, the study achieved improved task handling speed, accuracy, and reward generation compared to using traditional approaches. While the DRL-PPSO model is promising, its high complexity and heavy dependence on large training sets are of concern for use in resource-restricted or high-growth environments.

Kanbar *et al.* [25] compared static and dynamic load balancing approaches, pointing to the weaknesses of static models in heterogeneous cloud environments. Their work highlights the merits of distributed and dynamic methods, especially with respect to fault tolerance and the adaptability of workload. It does mention that it is hard to implement these models based on the requirement for constant environment monitoring and the likelihood of additional system overhead.

In a similar direction, Malipatil *et al.* [26] proposed a deep reinforcement learning-based workload scheduling model that adapts dynamically to changing cloud conditions. Their system uses Deep Q-Networks (DQN) to analyze workload features such as execution time, CPU, and memory usage before making scheduling decisions. The model achieved latency reductions up to 15 ms, throughput up to 500 tasks/sec, and 92% load balancing efficiency, along with 97% Quality of Service. While highly effective, their method is relatively complex and demands significant computational resources, which may not be ideal for lightweight, real-time systems.

Albalawi *et al.* [27] proposed a dynamic scheduling framework aimed at improving task allocation and load balancing in cloud-based parallel and distributed environments. The model combines multiple metaheuristic techniques—such as sunflower whale optimization and a hybrid ant colony–genetic algorithm—to address the performance bottlenecks caused by workload variability and system heterogeneity. This aligns with the current study's focus on achieving efficient and adaptive task scheduling in real-time cloud environments. Both studies share a common objective of enhancing load distribution and optimizing resource utilization under parallel processing conditions. However, while Albalawi's framework demonstrates high performance in terms of makespan and energy efficiency, it introduces considerable algorithmic complexity and computational cost due to the nature of its multi-phase metaheuristic design.

In a related study, Mahmoud *et al*. [28] introduced the task scheduling–decision tree algorithm, which applies a supervised decision tree model for multi-objective task scheduling in heterogeneous cloud environments. The task scheduling–decision tree algorithm dynamically allocates tasks to virtual machines by evaluating system conditions, aiming to optimize makespan, load balancing, and resource utilization simultaneously. Compared to conventional scheduling techniques such as Heterogeneous Earliest Finish Time, Technique for Order Preference by Similarity to Ideal Solution – Entropy Weight Method (TOPSIS-EWM), and Q-Learning–based Heterogeneous Earliest Finish Time, Tabu Search Decision Tree achieved improvements in load distribution (by up to 59.06%), reduced makespan (by 5.21%), and enhanced overall system efficiency. Notably, the model maintains low computational complexity and interpretable structure, which addresses a major limitation of other machine-learning–based schedulers such as DRL-PPSO and DQN, whose effectiveness comes at the cost of increased model complexity and resource demands. Although task scheduling–decision tree does not specifically focus on matrix computation tasks, its general-purpose scheduling logic and decision tree–based adaptiveness make it highly applicable to cloud-based parallel processing systems, such as the one proposed in the current study.

Together, these studies show an emerging agreement regarding the need for adaptive, intelligent load balancing methods that can function under dynamic scenarios. While each of the proposed techniques possesses certain merits—their accuracy in optimization, energy efficiency, or parallel scheduling—the majority of them still suffer from important shortcomings involving scalability, real-time adaptability, or integration with heterogeneous systems. It highlights an urgent requirement for solutions that maximize task distribution while reacting adaptively to run-time performance indicators without adding too much complexity or computational overhead.

3. Materials and Methods

The purpose of this study is to design and test an AI load balancer for parallel computation tasks in cloud computing. It is intended to balance matrix computation tasks (that is, matrix multiplication) among several servers. This study employs a decision tree regressor to make informed task assignment decisions. Decision trees ask questions and develop decision rules based on the structure of the datasets that make up a problem in order to create classification and regression models that resemble trees [29]. Inquiries begin at the root node, the basic element of the tree structure, and the tree branches until they reach the leaves, the last component of the tree structure [30, 31].

This machine learning model will predict which server is most appropriate based on performance measures in real time. The AI-based load balancer was tested against a standard manual load balancer using various matrix sizes and appropriate server configurations.

3.1. System Architecture

The proposed system comprises three key components that work together to perform distributed matrix computations in a cloud environment: the client side, the web server (cloud side), and the server side.

On the client side, users initiate the process by generating two square matrices of equal dimensions (e.g., 128×128 or 512×512), filled with randomly generated numerical values to simulate computational workload. These matrices are then transmitted to the cloud system for processing, either via a web-based interface or a mobile application.

The web server, serving as the intermediary between clients and servers, manages incoming matrix multiplication requests and is responsible for task allocation. The server implements load balancing logic, which can function either manually or through an AI-based approach employing a decision tree regressor. To support optimal task distribution, the server continuously retrieves real-time performance metrics from all connected servers.

On the server side, matrix computations are executed by multiple servers deployed in a real-world environment. Two physical machines were used to host four virtual servers—two per machine. Each machine was equipped with an Intel Core i7 processor, 8 cores, 16 GB of random access memory, and

a 512 GB Solid State Drive. All servers were connected through a 1 Gbps local area network. Upon completing the computations, the results were sent back to the web server, which then returned them to the client. Figure 1 illustrates the complete system architecture, depicting the workflow from client request submission to final result delivery via the AI-based load balancer.

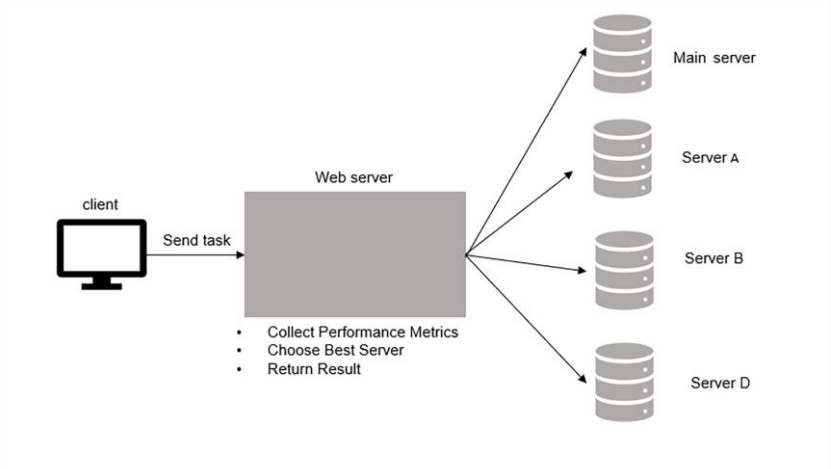

Figure 1: System architecture of the AI-based load balancing system using a decision tree regressor.

3.2. AI-Based Load Balancer

Our proposed load balancer uses a decision tree regressor, a supervised learning algorithm that predicts numerical values from a given set of input features [32]. Annotated data is referred to as a training set in SML, whereas unannotated data is the testing set. Class labels are used for annotations with discrete values, while continuous target values are used for constant numerical annotations. Regression and classification are the two categories under which SML issues belong. Predicting the discrete values that fit into a certain class is the goal of classification, a form of SML where the results have discrete values. The form of SML known as regression is obtained from labelled datasets and predicts continuous-valued results for the most recent data that has been introduced into the algorithm [33, 34]. A decision tree divides a dataset into various leaves and branches. A decision tree's nodes each have two or more branches, and each branch evaluates the value of the quality. The target value that we want to predict is represented by the leaf node. It uses this model to predict how quickly each server can accomplish a specified task [35, 36].

3.2.1. Input Features

The AI-based load balancer makes task assignment decisions based on real-time performance data collected from each server. It utilizes four key indicators as input features: CPU usage, memory usage, execution time, and network latency. CPU usage reflects the portion of a server's processing power currently in use, while memory usage indicates the percentage of random-access memory being consumed. Execution time measures how long a server takes to complete its most recent task, expressed in milliseconds for precision. Network latency captures the delay experienced when transmitting data between the server and the client. These features collectively provide a comprehensive snapshot of each server's performance, enabling the system to assess which server is best suited for incoming computational workloads.

Using these inputs, the decision tree regressor predicts the estimated execution time for each server. The server with the shortest predicted execution time is selected to handle the incoming matrix multiplication task. This ensures that computational workloads are distributed efficiently, reducing overall processing time and avoiding server bottlenecks. The model's predictive capability enables it to adapt dynamically to changing server conditions in real time.

The decision tree regressor was trained using historical data from prior matrix computation tasks conducted on local servers. The training dataset included actual measurements for CPU and memory usage, execution time, and network latency, along with the corresponding task completion durations.

Training was performed offline using a general-purpose machine learning framework, such as Scikit-learn in Python. Once the model was trained, it was embedded into the web server to perform real-time predictions and guide load balancing decisions during system operations. Figure 2 illustrates a simplified representation of the decision tree structure, showing how the model evaluates the performance indicators to select the optimal server for each task.

.

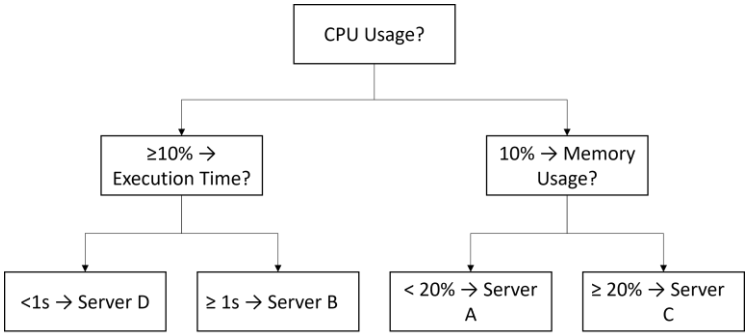

Figure 2: Simplified decision tree used by the AI-based load balancer to determine the optimal server based on CPU usage, execution time, and memory usage.

3.3. Manual Load Balancer

To evaluate the effectiveness of the proposed AI-based load balancer compared to the manual load balancing approach, a series of controlled experiments were conducted using varying system configurations and testing conditions. The core computational task selected for these experiments was matrix multiplication, involving the multiplication of two square matrices. Matrix sizes of 128×128 and 512×512 were used throughout the testing due to their computational intensity and common application in parallel processing environments.

The experimental environment was configured to simulate realistic performance conditions across a variety of server setups. Multiple servers were used, each representing different levels of CPU and memory utilization to closely mirror real-world cloud computing scenarios. This setup enabled the testing of the load balancing system under both uniform and heterogeneous resource availability.

Key performance indicators were recorded to assess the efficiency and responsiveness of each load balancing method. The metrics included CPU usage, memory usage, execution time, and network latency for each server, as well as aggregated measurements for the system as a whole. Task distribution data was also collected to evaluate how evenly computational workloads were allocated across available servers, providing insight into load balancing fairness and resource utilization. Execution time was measured in milliseconds to capture the precision of task completion, while network latency represented the communication delay between clients and servers.

The system workflow begins when a client initiates a matrix multiplication request. The web server then collects real-time performance data from all connected servers. Depending on the selected balancing method, the system either assigns the task to the next server in a fixed sequence (manual approach) or uses the decision tree regressor to determine the server with the most favorable performance conditions (AI-based approach). Once the selected server completes the task, the result is sent back to the web server and then returned to the client. Throughout this process, all relevant performance metrics are logged for analysis and comparison.

The experimental setup included four servers identified by their roles and unique IP addresses: the Main server, Server A, Server B, and Server D. These servers were connected via a local area network, forming a distributed environment suitable for parallel processing. The use of multiple matrix sizes and varying system conditions allowed for a comprehensive evaluation of the performance, scalability, and adaptability of the AI-based load balancing approach compared to its manual counterpart.

4. Results

This section presents the performance outcomes of two load balancing techniques implemented in a parallel cloud environment for matrix multiplication tasks: a manual (static) load balancer and an AI-based load balancer using a decision tree regressor. Experiments were conducted using varying matrix sizes and different numbers of server. Performance was evaluated based on six key metrics: task distribution, CPU usage, memory usage, execution time, network latency, and scalability. The experimental setup involved four servers—the Main server, Server A, Server B, and Server D—each connected through a common network and assigned a unique IP address. This setup enabled the effective evaluation of load balancing performance under different system conditions.

4.1. Task Distribution

Task distribution is the process of allocating incoming workloads or requests to different servers within the network. Effective distribution helps prevent individual servers from becoming overloaded while others remain underutilized, ensuring optimal system performance and resource usage [37].

In the case of the AI-based load balancer, the allocation of tasks across servers is fairly balanced. The highest percentage of tasks, 32.08%, was assigned to 192.168.33.3:5004 (Server_D), while the lowest, 6.19%, was allocated to 192.168.33.3:5000 (Main Server). This distribution reflects the decision-making process of the AI-based system, which relies on real-time performance metrics such as CPU usage, memory availability, and network latency. The main server, which also manages task coordination and system control functions, was intentionally assigned fewer computational tasks to maintain its responsiveness. This design choice supports the system's logic, ensuring that computational loads are directed to servers with greater availability and fewer responsibilities.

By contrast, the manual load balancer employs a static task allocation strategy, distributing workloads to servers in a predetermined sequence without accounting for current system conditions. This approach often leads to inefficiencies, as some servers may become overloaded while others remain underused due to the absence of dynamic performance evaluation during the task assignment.

4.2. Best Server for Task

In the AI-based load balancing approach, Server_D (http://192.168.33.3:5004) is identified by the system as the optimal server for executing the task. This selection is based on real-time performance metrics, including CPU utilization, memory consumption, execution time, and network latency. By evaluating these factors, the AI-based system dynamically determines which server is best suited to handle the workload under the current conditions, thereby maximizing efficiency and minimizing delays.

In contrast, the manual load balancer selects Server_B for the same task. However, this selection is made without evaluating the server's current performance state. Because the manual method lacks real-time adaptive capabilities, it cannot consistently select the most suitable server, potentially resulting in inefficiencies and imbalanced resource utilization.

4.3. Server Performance Metrics

4.3.1. AI Load Balancer Results

The performance of the AI-based load balancer is summarized in Table 1, which presents the per-server metrics across CPU usage, memory usage, execution time, and network latency.

CPU usage across servers ranged from 3.9% to 20.5%, resulting in a total system CPU utilization of 43.09%. This distribution indicates that the AI-based load balancer achieved a well-balanced workload among the servers, avoiding both underutilization and excessive strain on any single node.

Memory usage remained consistently low on all servers, with individual usage values ranging from 7.28512% to 7.29654%. The overall memory consumption across the system totaled 29.17%, demonstrating efficient resource utilization and minimal overhead.

Execution time varied between 0.1 milliseconds and 1,000.9 milliseconds across servers, with a total execution time of 1,001.2 milliseconds. This reflects the system's ability to perform high-speed processing, particularly under the guidance of the AI-based load balancer.

Network latency was also significantly minimized, ranging from 1.02056 milliseconds to 2.00496 milliseconds per server. The total latency recorded was only 5.12361 milliseconds, indicating that the AI-based method effectively reduced communication delays and enhanced responsiveness within the cloud environment.

Table 1: Per-server performance metrics using the AI Load Balancer.

Server	CPU Usage (%)	Memory Usage (%)	Execution Time (ms)	Network
Server_D	4.8	7.28512	0.1	1.02056
Server_B	20.5	7.29256	0.1	1.05189
Server_A	14.7	7.29409	0.1	1.04620
Main_Server	3.9	7.29654	1,000.9	2.00496

4.3.2. Manual Load Balancer Results

In the manual load balancing setup, CPU usage across individual servers ranged from 9.8% to 22.6%, with the total system CPU usage reaching 64.40%. This indicates a significantly higher processing load compared to the AI-based approach, suggesting the less efficient distribution of tasks across the system.

Memory consumption was considerably higher and more variable, ranging from 51.3% to 60.4% per server. The total memory usage reached 223.40%, reflecting poor memory management and indicating that some servers were overburdened while others may have had available resources go unused.

Execution time in the manual approach varied between 41.5 milliseconds and 59.5 milliseconds, with a total execution time of 7,064.5 milliseconds. This is substantially longer than the 1,001.2 milliseconds recorded under the AI-based load balancer, highlighting the impact of static task allocation on processing efficiency.

Network latency under the manual load balancing method ranged from 42 milliseconds to 67 milliseconds per server, with a total latency of 214.00 milliseconds. This is significantly higher than the 5.12361 milliseconds recorded in the AI-based setup. These findings demonstrate that the AI-based load balancer substantially outperforms the manual method in terms of execution speed and communication efficiency, as further illustrated in figure 3.

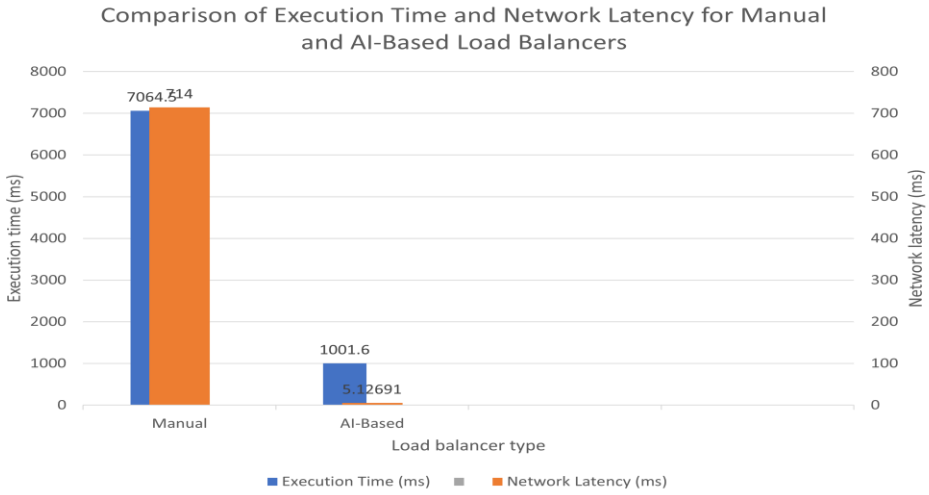

Figure 3: Bar chart comparing execution time and network latency for AI-based and manual load balancing.

4.4. General Findings

The AI load balancer enhances resource consumption and produces better results because it can choose the most effective server as well as dynamically balance workloads and efficiently control CPU

and memory consumption, ensuring neither overuse nor underutilization, while also greatly increasing task processing execution speed. Additionally, AI-based load balancing enhances system response times and speeds up data transfer, which decreases network latency. Manual load balancing uses more resources than is necessary, which results in inefficient allocation. All things considered, AI-based load balancing demonstrated consistently better performance under varying workloads because it improves system performance.

4.5. Theoretical Outcomes

The AI load balancer enables faster task processing with lower latency while optimizing CPU and memory use to improve system performance. Without human intervention, it can adapt dynamically to growing workloads because of its power-efficient scaling features. It lowers hardware needs through improved resource management, which results in cost savings. The AI model adjusted effectively to workload changes, minimizing delays during test runs, which increases system reliability. However, manual load balancing is impacted by a higher system load, higher network latency, slower reaction times because of longer execution durations, and increased CPU and memory consumption. Additionally, it lacks the scalability required to effectively manage large, unpredictable workloads. Additionally, manual load balancing ineffectively distributes work, which wastes resources with particular servers. As shown in Table 2, the AI-based load balancer significantly outperforms the manual approach across all performance metrics.

Table 2: Comparative performance analysis of the manual and AI-based load balancing methods.

Metric	Manual Load Balancer	AI Load Balancer	Optimal Choice	Reasoning
Task Distribution	Static, unbalanced	AI adapts dynamically	AI (More adaptive)	AI helps balance the workload, thus minimizing bottlenecks.
CPU Usage	Min: 9.8% Max: 22.6% Total: 64.40%	Min: 3.9% Max: 20.5% Total: 43.09%	AI (Better efficiency)	AI keeps the CPU from being overloaded or underutilized, keeping it stable.
Memory Usage	Min: 51.3% Max: 60.4% Total: 223.40%	Min: 7.28512% Max: 7.29654% Total: 29.17%	AI (Optimized memory)	AI effectively handles memory load without consuming an excessive amount of memory.
Execution Time	Min: 1,569.7 ms Max: 1,946.5 ms Total: 7.0645s	Min: 0.1 ms Max: 1,001.1 ms Total: 1.001600s	AI (Much faster)	AI reduces execution time by enhancing server selection.
Network Latency	Min: 42 ms Max: 67 ms Total:214.00ms	Min: 1.02056 ms Max: 2.0049ms Total:5.12361ms	AI (Lower latency)	AI reduces latency by going about the most efficient way to select the best server.
Resource Utilization	Inefficient, leading to idle or overloaded servers	Optimized, balancing server workloads	AI (Better utilization)	AI contributes to the overall efficiency and performance of the system.
Scalability	Limited	High	AI (Highly scalable)	AI works well for dynamic workloads because it can adjust in real time.

5. Discussion

The results clearly demonstrate that the AI-powered load balancer, driven by a decision tree regressor, delivers consistently better performance than the traditional manual method across all measured criteria. This improvement stems from its ability to make intelligent, data-driven decisions in real time, dynamically adjusting task allocation according to the live operational status of each server. By continuously monitoring CPU usage, memory consumption, and network conditions, the system can anticipate performance bottlenecks before they occur, reassign workloads accordingly, and maintain a balanced distribution of resources. This adaptability not only ensures optimal resource utilization but also enhances system stability, even when workloads fluctuate unpredictably.

One of the most striking results is the dramatic reduction in execution time—from 7,060 milliseconds to just 1,000 milliseconds—demonstrating the effectiveness of predictive task routing in minimizing delays and preventing overloads. By identifying the server with the shortest expected completion time for each incoming task, the AI system significantly reduces latency and increases throughput. These results align with the findings of Singhal *et al.* [22], who achieved similar gains in energy efficiency and makespan reduction in dynamic environments, although their approach lacked the transparency and interpretability of decision tree-based models, which offer clear reasoning for each decision made.

Beyond processing speed, the system also achieved substantial improvements in resource balancing. CPU and memory usage were spread more evenly across servers, bringing overall memory usage down from nearly 223% to just 29.17%. This reduction is critical, as it helps avoid resource saturation—a common cause of performance degradation and unexpected crashes in high-demand cloud environments. Comparable improvements have been observed in prior studies, such as the work by Pradhan *et al.* [15], where deep reinforcement learning enhanced resource handling; however, their approach required higher computational overhead and was more complex to implement.

Network performance also benefited significantly, with latency decreasing from 214 milliseconds to slightly over 5 milliseconds. This improvement is particularly valuable for latency-sensitive applications, such as real-time analytics, online gaming, and financial transactions, where even small delays can impact user experience or operational accuracy. These results are consistent with the performance gains seen in delay-aware scheduling models like the LEWIS framework [24] , which emphasize incorporating network delay into task allocation decisions.

Perhaps equally important is the system's scalability. As infrastructure size increased, performance remained consistent, addressing one of the main limitations of static scheduling approaches in heterogeneous and growing cloud environments. This directly responds to the challenges highlighted by Kanbar and Faraj [25], who noted that traditional methods often struggle to adapt when system configurations change or when workloads scale up rapidly.

Taken together, these findings strongly indicate that a lightweight, interpretable AI model such as the decision tree regressor is not only capable of delivering high processing speeds and operational efficiency but also excels in adaptability, scalability, and ease of deployment. Its combination of simplicity, low computational cost, and strong predictive performance makes it a practical choice for real-time cloud computing contexts, particularly where transparency, reliability, and cost-effectiveness are as important as raw speed.

6. Conclusions

This study demonstrates that the effectiveness and responsiveness of cloud computing systems can be significantly enhanced through the use of an AI-based load balancing technique, specifically one driven by a decision tree regressor. By assigning matrix computation tasks in real time based on current server conditions, the proposed approach improves resource utilization, reduces task completion time, and minimizes network latency. These findings reinforce the value of integrating intelligent algorithms into modern parallel cloud environments to optimize performance.

The primary contribution of this work is the development of a lightweight, data-driven load balancer capable of dynamically adapting to varying workloads and server conditions. This adaptive mechanism surpasses traditional static methods by improving fault tolerance, increasing throughput, and ensuring more reliable system behavior under real-time demands. The system's ability to predict optimal server assignment based on live performance metrics confirms the potential of machine learning in building scalable and efficient distributed infrastructures.

Despite these advantages, the study was limited by the scope of its testing, specifically the focus on matrix multiplication tasks and a fixed set of server configurations. Future research should explore the performance of alternative machine learning models, such as Random Forest, Gradient Boosting, and Support Vector Regression, to identify optimal trade-offs between accuracy, computational overhead, and inference speed. Incorporating online or adaptive learning techniques could further enhance the model's responsiveness to dynamic cloud conditions. Additionally, extending the application of

this method to other computational tasks beyond matrix multiplication would offer insights into its generalizability across diverse cloud-based workloads.

Author contributions: Lina Sarkawt Hiwa: Conceptualization, Methodology, Writing – original draft. **Zryan Najat Rashid:** Supervision, Investigation, Review & editing.

Data availability: The data are available upon reasonable request by the authors.

Conflicts of interest: The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this study.

Funding: The authors did not receive support from any organization for the submitted work.

References

[1] T. Khan, W. Tian, and R. Buyya, "Machine learning (ML)-centric resource management in cloud computing: A review and future directions," *arXiv preprint arXiv:2105.05079*, May 2021. [Online]. Available: http://arxiv.org/abs/2105.05079

[2] D. A. Shafiq, N. Z. Jhanjhi, and A. Abdullah, "Load balancing techniques in cloud computing environment: A review," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 3910–3933, Jul. 2022, doi: 10.1016/j.jksuci.2021.02.007.

[3] R. Islam *et al.*, "The future of cloud computing: Benefits and challenges," *International Journal of Communications, Network and System Sciences*, vol. 16, no. 04, pp. 53–65, 2023, doi: 10.4236/ijcns.2023.164004.

[4] R. Younis, M. Iqbal, K. Munir, M. A. Javed, M. Haris, and S. Alahmari, "A comprehensive analysis of cloud service models: IaaS, PaaS, and SaaS in the context of emerging technologies and trend," in *5th International Conference on Electrical, Communication and Computer Engineering, ICECCE 2024*, Institute of Electrical and Electronics Engineers Inc., 2024. doi: 10.1109/ICECCE63537.2024.10823401.

[5] N. S. Aldahwan and M. S. Ramzan, "Descriptive literature review and classification of community cloud computing research," Scientific Programming, vol. 2022, 2022, doi: 10.1155/2022/8194140.

[6] P. Goswami, N. Faujdar, S. Debnath, A. K. Khan, and G. Singh, "Investigation on storage level data integrity strategies in cloud computing: classification, security obstructions, challenges and vulnerability," *Journal of Cloud Computing*, vol. 13, no. 1, Art. 45, Dec. 2024, Springer Science and Business Media Deutschland GmbH, doi: 10.1186/s13677-024-00605-z.

[7] F. Khoda Parast, C. Sindhav, S. Nikam, H. Izadi Yekta, K. B. Kent, and S. Hakak, "Cloud computing security: A survey of service-based models," *Comput Secur*, vol. 114, Mar. 2022, doi: 10.1016/j.cose.2021.102580.

[8] N. Chauhan **et al.**, "A systematic literature review on task allocation and performance management techniques in cloud data center," *Computer Systems Science and Engineering*, vol. 48, no. 3, pp. 571–608, 2024, doi: 10.32604/csse.2024.042690.

[9] K. Mishra and S. K. Majhi, "A state-of-art on cloud load balancing algorithms," *International Journal of Computing and Digital Systems*, vol. 9, no. 2, pp. 201–220, Mar. 2020, doi: 10.12785/IJCDS/090206.

[10] A. K. Moses, A. J. Bamidele, O. R. Oluwaseun, S. Misra, and A. A. Emmanuel, "Applicability of MMRR load balancing algorithm in cloud computing," *International Journal of Computer Mathematics: Computer Systems Theory*, vol. 6, no. 1, pp. 7–20, 2021, doi: 10.1080/23799927.2020.1854864.

[11] N. Devi, K. Rajalakshmi, R. Parthasarathy, and G. Kousalya, "A systematic literature review for load balancing and task scheduling techniques in cloud computing," *Artificial Intelligence Review*, vol. 57, no. 10, Oct. 2024, doi: 10.1007/s10462-024-10925-w.

[12] Z. S. Ageed, M. R. Mahmood, M. S. Sadeeq, M. B. Abdulrazzaq, and H. I. Dino, "Cloud computing resources impacts on heavy load parallel processing approaches," *IOSR Journal of Computer Engineering (IOSR JCE)*, vol. 22, no. 3, Ser. IV, pp. 30–41, May–June 2020, doi: 10.9790/0661-2203043041.

[13] K. Rajammal and M. Chinnadurai, "Dynamic load balancing in cloud computing using predictive graph networks and adaptive neural scheduling," *Scientific Reports*, vol. 15, no. 1, Dec. 2025, doi: 10.1038/s41598-025-97494-2.

[14] Z. A. Aziz, D. N. Abdulqader, A. B. Sallow, and H. K. Omer, "Python parallel processing and multiprocessing: a review," Academic Journal of Nawroz University, vol. 10, no. 3, pp. 345–354, Aug. 2021, doi: 10.25007/ajnu.v10n3a1145.

[15] M. H. Kashani and E. Mahdipour, "Load balancing algorithms in fog computing," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1505–1521, Mar. 2023, doi: 10.1109/TSC.2022.3174475.

[16] M. Vijayaraj, R. M. Vizhi, P. Chandrakala, L. H. Alzubaidi, K. Muzaffar, and R. Senthilkumar, "Parallel and distributed computing for high-performance applications," E3S Web of Conferences, vol. 399, Art. no. 04039, 10 pp., Jul. 2023, doi: 10.1051/e3sconf/202339904039

[17] S. Dahake and R. Y. Nagpure, "Research paper on basic parallel processing," *IOSR Journal of Engineering (IOSR-JEN)*, vol. 2, pp. 77–83, presented at the 2nd National Conference on Recent Trends in Computer Science and Information Technology, Nagpur, India, 2019. [Online]. Available: https://www.iosrjen.org/Papers/Conf.19021-2019/Volume-2/14.%2077-83.pdf [Accessed: Jun. 2, 2025].

[18] H. Blockeel, L. Devos, B. Frénay, G. Nanfack, and S. Nijssen, "Decision trees: from efficient prediction to responsible AI," *Frontiers in Artificial Intelligence*, vol. 6, 2023, doi: 10.3389/frai.2023.1124553.

[19] I. D. Mienye, Y. Sun, and Z. Wang, "Prediction performance of improved decision tree-based algorithms: A review," in *Procedia Manufacturing*, Elsevier B.V., 2019, pp. 698–703. doi: 10.1016/j.promfg.2019.06.011.

[20] A. A. Mahamat *et al.*, "Decision tree regression vs. gradient boosting regressor models for the prediction of hygroscopic properties of Borassus fruit fiber," *Applied Sciences (Switzerland)*, vol. 14, no. 17, Sep. 2024, doi: 10.3390/app14177540.

[21] J. Singh Kushwah, A. Kumar, S. Patel, R. Soni, A. Gawande, and S. Gupta, "Comparative study of regressor and classifier with decision tree using modern tools," *Mater Today Proc*, vol. 56, pp. 3571–3576, Jan. 2022, doi: 10.1016/j.matpr.2021.11.635.
[22] S. Singhal *et al.*, "Energy-efficient load balancing algorithm for cloud computing using rock hyrax optimization," *IEEE Access*, vol. 12, pp. 48737–48749, 2024, doi: 10.1109/ACCESS.2024.3380159.
[23] Y. Zhang, B. Liu, Y. Gong, J. Huang, J. Xu, and W. Wan, "Application of machine learning optimization in cloud computing resource scheduling and management," *Applied and Computational Engineering*, vol. 64, no. 1, pp. 9–14, May 2024, doi: 10.54254/2755-2721/64/20241359.
[24] A. Pradhan, S. K. Bisoy, S. Kautish, M. B. Jasser, and A. W. Mohamed, "Intelligent decision-making of load balancing using deep reinforcement learning and parallel PSO in cloud environment," *IEEE Access*, vol. 10, pp. 76939–76952, 2022, doi: 10.1109/ACCESS.2022.3192628.
[25] A. B. Kanbar and K. Faraj, "Modern load balancing techniques and their effects on cloud computing," *Journal of Hunan University Natural Sciences*, vol. 49, no. 7, pp. 37–43, Jul. 2022, doi: 10.55463/issn.1674-2974.49.7.5.
[26] A. R. Malipatil, D. Gulyamova, A. Saravanan, *et al.*, "Energy-efficient cloud computing through reinforcement learning-based workload scheduling," *International Journal of Advanced Computer Science and Applications*, vol. 16, no. 4, **Art. no. 64**, May 2025. doi: 10.14569/IJACSA.2025.0160464.
[27] N. S. Albalawi, "Dynamic scheduling strategies for cloud-based load balancing in parallel and distributed systems," *Journal of Cloud Computing*, vol. 14, no. 1, Dec. 2025, doi: 10.1186/s13677-025-00757-6.
[28] H. Mahmoud, M. Thabet, M. H. Khafagy, and F. A. Omara, "Multiobjective task scheduling in cloud environment using decision tree algorithm," *IEEE Access*, vol. 10, pp. 36140–36151, 2022, doi: 10.1109/ACCESS.2022.3163273.
[29] B. T. Jijo and A. M. Abdulazeez, "Classification based on decision tree algorithm for machine learning," *Journal of Applied Science and Technology Trends*, vol. 2, no. 1, pp. 20–28, Jan. 2021, doi: 10.38094/jastt20165.
[30] B. K. Gacar and İ. D. Kocakoç, "Regresyon analizleri mi karar ağaçları mı? [Regression analyses or decision trees?]," *Celal Bayar Üniversitesi Sosyal Bilimler Dergisi*, pp. 251–260, Dec. 2020, doi: 10.18026/cbayarsos.796172.
[31] A. Deva Kumari, J. Prem Kumar, V. S. Prakash, and Divya K. S., "Supervised learning algorithms: A comparison," Kristu Jayanti Journal of Computational Sciences, vol. 1, no. 1, pp. 1–12, Nov. 2020
[32] I. D. Mienye and N. Jere, "A survey of decision trees: Concepts, algorithms, and applications," *IEEE Access*, vol. 12, pp. 86716–86727, 2024, doi: 10.1109/ACCESS.2024.3416838.
[33] S. H. Shetty, S. Shetty, C. Singh, and A. Rao, "Supervised machine learning: Algorithms and applications," in *Fundamentals and Methods of Machine and Deep Learning: Algorithms, Tools, and Applications*, Wiley, 2022, pp. 1–16. doi: 10.1002/9781119821908.ch1.
[34] Z. Ren, S. Wang, and Y. Zhang, "Weakly supervised machine learning," *CAAI Transactions on Intelligence Technology*, vol. 8, no. 3, pp. 549–580, Sept. 2023, doi: 10.1049/cit2.12216.
[35] T. W. Gyeera, A. J. H. Simons, and M. Stannett, "Regression analysis of predictions and forecasts of cloud data center KPIs using the boosted decision tree algorithm," *IEEE Trans Big Data*, vol. 9, no. 4, pp. 1071–1085, Aug. 2023, doi: 10.1109/TBDATA.2022.3230649.
[36] Priyanka and D. Kumar, "Decision tree classifier: a detailed survey," *International journal of information and decision sciences*, vol. 12, no. 3, pp. 246–269, Jul. 2020, doi: 10.1504/IJIDS.2020.108141.
[37] D. D. Vecliuc, F. Leon, and D. Logofătu, "A comparison between task distribution strategies for load balancing using a multiagent system," *Computation*, vol. 10, no. 12, Dec. 2022, doi: 10.3390/computation10120223.