processes

MDPI

Article

Dynamic Load Balancing in Cloud Computing: Optimized RL-Based Clustering with Multi-Objective Optimized Task Scheduling

Ahmad Raza Khan

Department of Information Technology, College of Computer and Information Sciences, Majmaah University, Majmaah 11952, Saudi Arabia; ar.khan@mu.edu.sa

Abstract: Dynamic load balancing in cloud computing is crucial for efficiently distributing workloads across available resources, ensuring optimal performance. This research introduces a novel dynamic load-balancing approach that leverages a deep learning model combining Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to calculate load values for each virtual machine (VM). The methodology aims to enhance cloud performance by optimizing task scheduling and stress distribution. The proposed model employs a dynamic clustering mechanism based on computed loads to categorize VMs into overloaded and underloaded clusters. To improve clustering efficiency, the approach integrates Reinforcement Learning (RL) with a sophisticated Hybrid Lyrebird Falcon Optimization (HLFO) algorithm. HLFO merges the Lyrebird Optimization Algorithm (LOA) and Falcon Optimization Algorithm (FOA), enhancing the effectiveness of load balancing. A Multi-Objective Hybrid Optimization model is introduced to optimize task scheduling while considering Quality of Service (QoS) parameters, including makespan minimization, energy consumption reduction, balanced CPU utilization, efficient memory usage, and task prioritization. The implementation, conducted in Python and CloudSim, demonstrates the model's ability to effectively allocate work between virtual machines (VMs) and physical machines (PMs), resulting in improved resource utilization, shortened makespan, enhanced CPU usage, and rigorous assessments affirming its efficacy. This research addresses the complexity of dynamic load balancing in cloud environments by combining deep learning, reinforcement learning, and hybrid optimization techniques, offering a comprehensive solution to optimize cloud performance under varying workloads and resource conditions.

Keywords: cloud computing; dynamic load balancing; task scheduling; hybrid lyrebird falcon optimization; multi-objective hybrid optimization

Citation: Khan, A.R. Dynamic Load Balancing in Cloud Computing: Optimized RL-Based Clustering with Multi-Objective Optimized Task Scheduling. *Processes* **2024**, *12*, 519. https://doi.org/10.3390/pr12030519

Academic Editors: Jiaqiang E and Wei Sun

Received: 21 January 2024
Revised: 19 February 2024
Accepted: 24 February 2024
Published: 4 March 2024

1. Introduction

Load balancing is a crucial task in cloud computing that ensures optimal performance and efficient use of resources. A crucial element of cloud computing, load balancing, divides computational tasks and network traffic among several servers or virtual machines (VMs) to preserve optimal resource utilization and efficiency [1,2]. Cloud computing has grown quickly as a result of modern technology and widespread internet usage; it is currently the basis for a wide range of apps and services used by various user types [3,4]. Effective load-balancing solutions are necessary to meet the vast and diversified demand for cloud-based services while managing variable workloads and ensuring perfect user experiences [5]. A substantial pool of virtualized resources, such as processing power, storage capacity, and networking capabilities, is made available by cloud computing over the internet [6]. As the use of cloud-based apps and services increases, efficient load balancing becomes essential to overcoming problems with scalability and adaptability [7,8]. Cloud users can access internet-based resources such as software and hardware applications by connecting to servers and virtual machines (VMs) over internet networks [9]. Enhanced flexibility,

cost effectiveness, higher-quality on-demand services, and online backup capabilities to avoid database overload are just a few of the many benefits of cloud computing. A few of the several companies that make up the cloud computing infrastructure include brokers, data centers, and Cloud Information Services (CiS). Data and virtual computers are stored physically on servers housed in data centers [10]. Optimizing resource usage is crucial for attaining appropriate load balancing in cloud environments.

The dynamic and complex nature of cloud workloads means that traditional load-balancing algorithms might not be able to handle them, which is why education is looking at new approaches like hybrid metaheuristics [11,12]. Hybrid metaheuristics combine elements of several different algorithms to create a powerful problem-solving tool. By integrating the benefits of several methodologies, they aim to give almost optimal outcomes in resolving complex load-balancing problems in cloud systems [13]. Due to their robustness and ability to adjust to varying workloads, these metaheuristics strike a balance between exploration and exploitation of the search space. Hybrid metaheuristics have several advantages for load balancing in cloud computing [14]. Large user volumes and dynamic workloads are two things that cloud infrastructures have to handle. Hybrid metaheuristics ensure maximum efficiency and efficient resource allocation even during high-use periods by promptly adapting to demand variations. Efficient use of resources is another essential element of cloud load balancing [15].

This research focuses on dynamic load balancing in cloud computing, proposing a novel approach that utilizes a deep learning model incorporating CNNs and RNNs. The main goal is to enhance cloud performance by optimizing task scheduling and stress distribution among VMs. The model employs a dynamic clustering mechanism based on computed loads, categorizing VMs into overloaded and underloaded clusters. To improve clustering efficiency, the approach integrates RL with a sophisticated HLFO algorithm, combining the LOA and FOA. The overarching objective is to achieve effective load balancing, considering QoS parameters such as makespan minimization, energy consumption reduction, balanced CPU utilization, efficient memory usage, and task prioritization. Implemented in Python and CloudSim, the suggested model aims to allocate work between VMs and PMs effectively, leading to improved resource usage, shortened makespan, enhanced CPU usage, and thorough assessments affirming its efficacy.

(1) The following challenges are addressed in this research:
   - The complexity of dynamic load balancing in cloud environments;
   - A need to address conflicting goals such as makespan minimization, energy consumption reduction, and balanced resource utilization;
   - Implementation challenges associated with deep learning, reinforcement learning, and hybrid optimization algorithms;
   - Ensuring adaptability to dynamic workloads and scalability to handle larger cloud environments.

(2) The research aims to achieve the following goals:
   - Optimize task scheduling and stress distribution in cloud computing;
   - Improve cloud performance under varying workloads and resource conditions;
   - Achieve effective load balancing considering conflicting QoS parameters;
   - Enhance clustering efficiency through the integration of RL and Hybrid Lyrebird Falcon Optimization;
   - Validate the proposed model's efficacy through thorough assessments and practical implementation in Python and CloudSim.

(3) The main contributions of the paper are as follows:
   - The proposed model incorporates a sophisticated deep learning approach by combining CNNs and RNNs to compute the VM load value. This integration enhances the accuracy and efficiency of workload computations, contributing to improved decision-making in load balancing;

- To address the challenges of clustering in load balancing, the research proposes a clustering approach that combines RL with advanced hybrid optimization algorithms, specifically HLFO. This innovative method enhances clustering efficiency and accelerates the convergence to optimal solutions;
- The research contributes a Multi-Objective Hybrid Optimization model for task scheduling, considering QoS parameters such as makespan minimization, energy consumption reduction, balanced CPU utilization, efficient memory usage, and task prioritizing. This comprehensive approach ensures a holistic optimization of task allocation in the cloud environment.

The remaining parts of the document are organized as follows. A summary of relevant work on load-balancing algorithms is given in Section 2. In Section 3, the cloud computing dynamic load-balancing approach is introduced, and Section 4 contains the experiment and results analysis. A brief summary and future works are provided at the conclusion in Section 5.

2. Literature Review

Autonomous Load Balancing, developed in 2021 by Ebadifard and Babamir [16], offers a mechanism for efficiently assigning requests in a cloud context, assuring system stability, slashing response times, and boosting resource productivity. It specifically addresses the issue of inter-VM communication overheads, which is frequently disregarded in other load-balancing methods. Evaluations utilizing the CloudSim tool and comparisons with existing approaches show that it improves workload distribution and allocation by classifying requests into CPU-bound and I/O-bound kinds.

Shafiq et al. [17] concentrated on workload balancing in the Infrastructure as a Service (IaaS) architecture of cloud computing in 2021. The suggested load-balancing algorithm prioritizes VMs based on Quality of Service (QoS) task characteristics, optimizes resource allocation, and complies with Service-Level Agreement (SLA) criteria. Results show that, compared to the existing dynamic LBA, the algorithm greatly improves resource utilization, decreases execution time, and increases makespan, addressing current research gaps and issues in cloud-based systems.

A three-tier architecture made up of cloud, fog, and consumer layers was suggested by Yu et al. [18] for cloud computing load balancing in 2022. In order to balance the fog load, it introduces a real-time VM movement method that optimizes resource utilization, throughput, and reaction time. The algorithm outperforms the closest data center technique, with an 11% improvement over the dynamic reconfigure with load (DRL) method and 18% better cost outcomes and optimized response time.

A brand-new load-balancing method dubbed FIMPSO, which combines the Firefly and Improved Multi-Objective Particle Swarm Optimization (IMPSO) techniques, was presented in 2020 by Devaraj et al. [19]. FIMPSO distributes workloads in cloud computing systems effectively in order to improve resource utilization and response times. The simulation outcomes show that it performs better than previous approaches, achieving an effective average load and better task execution.

A unique Quasi-Oppositional Dragonfly Algorithm for Load Balancing (QODA-LB) was created in 2022 by Latchoumi and Parthiban [20] to effectively handle the load-balancing issue in cloud computing. The QODA-LB algorithm, which outperforms other leading algorithms in terms of load-balancing effectiveness, delivers optimal resource scheduling by utilizing three variables and the Quasi-Oppositional-Based Learning (QOBL) concept. Numerous tests show that it is more effective and has a higher rate of convergence than the traditional Dragonfly Algorithm (DA).

A brand-new load-balancing approach for cloud computing, dubbed CMODLB, was introduced in 2021 by Negi et al. [21]. It combines supervised (artificial neural network), unsupervised (clustering), and soft computing (interval type 2 fuzzy logic system) techniques. The system uses artificial neural networks to cluster virtual machines (VMs) and uses multi-objective techniques with particle swarm optimization to schedule jobs for overloaded VMs.

To achieve load balancing among physical machines (PMs), VM migration decisions are made utilizing an interval type 2 fuzzy logic system. In comparison to previous algorithms, experimental results show enhanced performance with a notably shorter completion time and better resource utilization.

To effectively distribute work across VMs in cloud computing, Pradhan and Bisoy [22] offer LBMPSO, a load-balancing technique using modified PSO task scheduling. The algorithm uses job and resource information from the data center to reduce makespan and increase resource utilization. The performance of the system is greatly improved by LBMPSO compared to conventional approaches according to simulation results with CloudSim.

In 2022, Sefati et al. [23] published a work that focuses on employing the Grey Wolf Optimization (GWO) algorithm for load balancing in cloud computing. The GWO algorithm effectively distributes work among idle or busy nodes, optimizing the system's performance by taking resource reliability capability into account. The proposed method surpasses other techniques, according to simulation findings with CloudSim, providing lower costs, faster reaction times, and optimal solutions.

The challenges of minimizing energy consumption, SLA violations, and VM migrations in quickly expanding cloud data centers were addressed by Mapetu et al. [24] in 2021. The suggested dynamic VM-consolidation-approach-based load balancing makes use of four techniques, including VM selection based on imbalance degree for VM migration, BPSO metaheuristics for energy consumption and host shutdowns, and the Pearson correlation coefficient for SLA. The method shows promising results in effectively solving the NP-hard optimization problem through extensive simulations using real and random workloads.

The MOABCQ method, a multi-objective task-scheduling optimization strategy, was proposed by Kruekaew and Kimpan [25] for load balancing in cloud computing in 2022. Enhanced scalability is a significant benefit [26]. Conventional load-balancing strategies may not be as effective in cloud workloads due to their dynamic nature. Still, hybrid metaheuristics are designed to adjust to shifting circumstances and may respond quickly to workload variations [27,28]. The method seeks to optimize scheduling, resource utilization, and VM throughput by fusing the Artificial Bee Colony Algorithm (ABC) with the Q-learning algorithm. MOABCQ beats previous algorithms, according to experimental findings using CloudSim, in terms of lowering makespan, cost, and degree of imbalance and enhancing throughput and resource utilization. The literature review papers are given in Table 1.

Table 1. Comparative analysis of load-balancing methods in cloud computing.

Author Name and Citation	Method	Key Features	Performance Metrics
Ebadifard and Babamir [16]	Autonomous Load Balancing	Efficiently assigns requests, addresses inter-VM communication overheads	Improved workload distribution, reduced response times, boosted resource productivity
Shafiq et al. [17]	Dynamic LBA	Prioritizes VMs based on QoS task characteristics and complies with SLA criteria	Improved resource utilization, decreased execution time, increased makespan
Yu et al. [18]	Three-tier architecture	Utilizes cloud, fog, and consumer layers, introduces real-time VM movement method	11% improvement over DRL with load method, 18% greater cost outcomes, optimized response time
Devaraj et al. [19]	FIMPSO	Combines Firefly and IMPSO techniques, distributes workloads effectively	Better than previous approaches, effective average load, improved task execution
Latchoumi and Parthiban [20]	QODA-LB	Utilizes Quasi-Oppositional Dragonfly Algorithm, delivers optimal resource scheduling	More effective, higher rate of convergence than traditional DA

Table 1. *Cont.*

Author Name and Citation	Method	Key Features	Performance Metrics
Negi et al. [21]	CMODLB	Combines supervised, unsupervised, and soft computing techniques, uses artificial neural networks and interval type 2 fuzzy logic system	Enhanced performance, notably shorter completion time, better resource utilization
Pradhan and Bisoy [22]	LBMPSO	Uses modified PSO task scheduling, reduces makespan, and increases resource utilization	Greatly improved performance compared to conventional approaches
Sefati et al. [23]	GWO Algorithm	Utilizes Grey Wolf Optimization algorithm, optimizes system performance by considering resource reliability capability	Lower costs, faster reaction times, optimal solutions according to CloudSim simulations
Mapetu et al. [24]	Dynamic VM Consolidation	Uses dynamic VM-consolidation-approach-based load balancing, employs VM selection, BPSO metaheuristics, Pearson correlation coefficient	Promising results in minimizing energy consumption, SLA violations, and VM migrations through extensive simulations
Kruekaew and Kimpan [25]	MOABCQ	Multi-objective task-scheduling optimization strategy fuses ABC with Q-learning algorithm, seeks to optimize scheduling, resource utilization, and VM throughput	Beats previous algorithms in terms of lowering makespan, cost, degree of imbalance, enhancing throughput, and resource utilization according to CloudSim simulations

2.1. Problem Statement

To maintain optimal system performance and resource utilization in cloud computing, various challenges need to be addressed. One of the key issues is dealing with the diverse nature of cloud data centers, where resources possess different capacities and capabilities. Effectively allocating jobs among these varied resources is imperative to prevent overloading some and underutilizing others. Managing fluctuating workloads in the cloud environment presents a significant problem [1,7]. Workload requirements can change over time, necessitating load-balancing techniques capable of dynamically adapting resource allocation to meet shifting demands. This adaptability is crucial to avoiding performance bottlenecks during peak usage. Scalability is another challenge as cloud computing services must handle a growing number of customers and applications. Scalable load-balancing solutions are necessary to accommodate increasing demand and ensure effective job distribution across expanding infrastructure. Additionally, the adherence to Service-Level Agreements (SLAs) is crucial. Load-balancing strategies must consider SLAs between cloud providers and users to ensure that performance goals are achieved and user expectations are met. To reduce resource usage and computational complexity, load-balancing algorithms are designed with minimal overhead. Achieving efficient load balancing with minimal overhead is essential for realizing high-performance cloud computing environments. To enhance the clarity of the problem statement, it is crucial to mention the addressed problem with Quality of Service (QoS) parameters, such as those found in the energy-aware stochastic scheduler for batches of precedence-constrained jobs on a heterogeneous computing system and energy-efficient scheduling algorithms for batch-of-tasks (BoT) applications on heterogeneous computing systems. This inclusion will provide a more focused and specific understanding of the challenges at hand.

This work contributes to the topic of dynamic load balancing in cloud computing by presenting novel ideas that have not been explored before. In contrast to other methods, our model makes use of a deep learning framework to improve the load calculations' precision and flexibility. This research is unique in that it uses a dynamic clustering approach to

improve clustering efficiency and responsiveness to changing workloads. Furthermore, the use of the hybrid algorithm provides a special remedy for optimization problems. By presenting a Multi-Objective Hybrid Optimization model that optimizes task scheduling while taking into account an extensive collection of Quality of Service (QoS) characteristics, the study sets itself apart even more.

2.2. Objective Function

The objective function of the work is formulated using QoS metrics like response time, throughput, and availability.

(1) Response time

Response time is the amount of time it takes for a service to respond to a specific type of request. Response time is determined by the load intensity, which can be measured by the number of requests at once or the arrival rates (requests per second). QoS takes into account not just the average response time but also the response time percentile.

$$Avg_{RT} = \frac{T_{RT}}{N} \tag{1}$$

where Avg_{RT} is the average response time, T_{RT} denotes the total response time, and N is the total number of requests;

(2) Throughput

The throughput of a service is the maximum speed at which requests may be processed. QoS metrics include functions that describe how throughput varies with load intensity and the maximum throughput.

$$Throughput = \frac{N}{T} \tag{2}$$

where T is the total time interval;

(3) Availability

A system's or service's availability is the percentage of time it is operational and accessible to users. It is commonly expressed as an uptime percentage. High availability is crucial for mission-critical services as it minimizes downtime and service interruptions while ensuring a faultless user experience.

$$Availability = \frac{T_{Ut}}{T_{Ut} + T_{Dt}} \times 100 \tag{3}$$

where T_{Ut} is the total uptime, and T_{Dt} is the total downtime. Tasks are assigned to the chosen underloaded VMs in each cluster using the Multi-Objective Hybrid Optimization model. The best solution found by the model should serve as the basis for allocation.

The overall objective of the work is given by the following:

$$Objective\ Function = max(Avg_{RT}, Throughput, Availability) \tag{4}$$

The system aims to meet a performance threshold that is set by the maximum value of its average response time, throughput, and availability. This could be a way to ensure that the system is well balanced and does not have a single performance metric falling below a certain acceptable level.

3. Proposed Methodology

Distributing a workload over one or more servers, network ports, hard drives, or other computer resources is known as load balancing. The network infrastructure and large, powerful (and costly) computing hardware used in typical data center implementations are vulnerable to the same risks as any other physical device, such as hardware failure, outages of power or other networks, and resource constraints during periods of high demand.

By employing commodity servers to handle load balancing, cloud-based load balancing deviates from traditional theories about load-balancing design and implementation. Embracing economies of scale and new prospects presents unique problems of its own. To guarantee that no resource is idle and that all of the resources are being used effectively, load balancing is used. The load can be moved from the source nodes with more work to the destination nodes, which are comparatively less busy, to provide a balanced task distribution. Dynamic load balancing describes load balancing that is applied in real time. Depending on the manner in which the execution nodes are chosen, direct or iterative methods can be used to achieve this dynamic load distribution. Figure 1 shows the block diagram of the proposed load-balancing model.

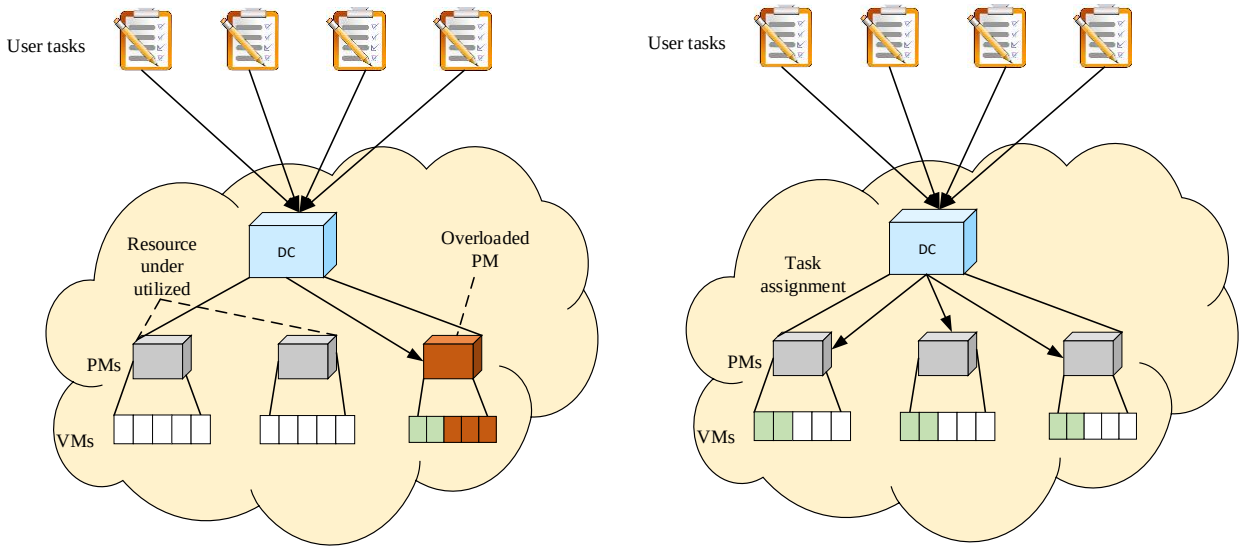

Figure 1. Block diagram for the overloaded and underloaded VMs.

3.1. Collect Virtual Machine Load Data

The proposed work focuses on load balancing among PMs and VMs in a cloud environment through hybrid supervised (with target attribute, i.e., ANN) and unsupervised (without target attribute, i.e., BOK-means clustering) machine learning techniques for efficient load calculation and VM clustering. The proposed cloud environment consists of M number of PMs as $P = \{ PM1, PM2, \ldots, PMM \}$. In each PM, L number of VMs is included as $VM = \{ VM1, VM2, \ldots, VML \}$. The cloud environment is involved with S number of user tasks, represented as $T = \{ T1, T2, \ldots, TS \}$. To balance load among M PMs and L VMs, two entities, the VM manager and cloud balancer, are involved.

Load Computation Using CNN and RNN

To compute the loads of each VM, we utilize a hybrid deep learning approach by combining CNNs and RNNs.

- CNNs are used for feature extraction from sensor data. In the context of WSN, this might involve processing spatial information. For example, if your WSN consists of sensor nodes distributed in a physical area, CNNs can be used to capture spatial patterns and relationships among nodes;
- RNNs are well suited for processing sequential data, which is often the case in WSNs. You can use RNNs to capture temporal dependencies and relationships among sensor readings over time. This is important for load computation in dynamic environments.

3.1.1. CNN

One of the most popular machine learning (ML) methods is the CNN, especially for applications requiring vision. Grid-like data may be utilized to train CNN representations,

and recent ML applications have shown considerable performance gains when using it. Convolution and pooling layers are frequently alternated with many fully connected layers in a conventional CNN configuration. This section provides a quick explanation of how various components fit into the CNN architecture; the basic CNN is shown in Figure 2.

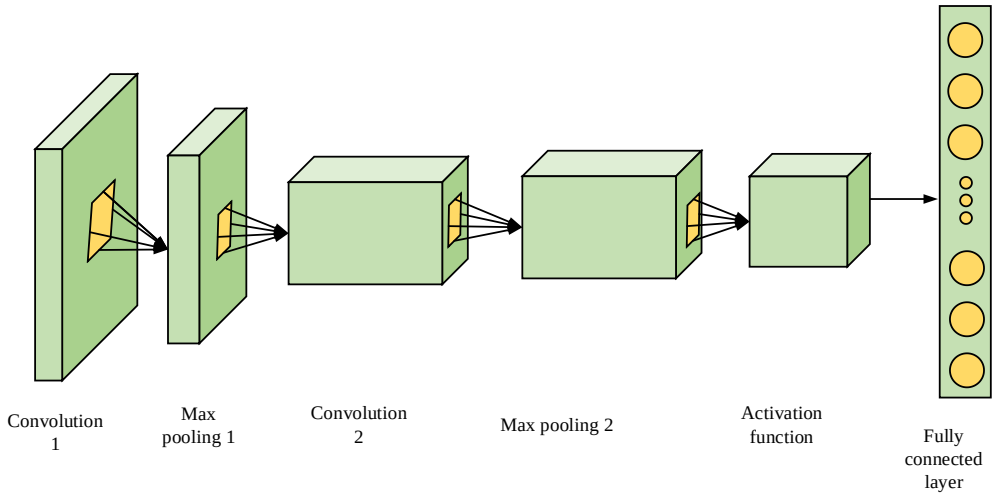

Figure 2. The basic structure of the CNN.

(1) Convolutional layer

It is composed of these kernels and uses each neuron as a convolutional kernel. Conversely, if the kernel is symmetric, convolution becomes a correlation operation. Cutting the image into distinct sections, known as receptive fields, is how the convolutional kernel works. Extracting feature motifs is made easier by dividing an image into tiny chunks. This is one way to express a convolution operation.

$$f_l^k(p,q) = \sum_c \sum_{x,y} i_c(x,y).e_l^k(u,v) \tag{5}$$

where $i_c(x,y)$ is the input data component I_c, which has been multiplied elementally by the $e_l^k(u,v)$ index of the k^{th} convolutional kernel k_1 of the l^{th} layer. Despite the possibility of representation, the resultant feature map of the k^{th} convolutional procedure is given in Equation (6).

$$F_l^k = \left[f_l^k(1,1), \ldots, f_l^k(p,q), \ldots, f_l^k(P,Q) \right] \tag{6}$$

(2) Pooling layer

Convolution operations generate feature motifs, which can appear anywhere in the picture. As long as a feature's estimated position in relation to other features is kept after extraction, the location of the feature itself is less significant. An intriguing local procedure is pooling or downsampling. It accumulates relevant data from the receptive field and creates the dominant reaction in this specific, constrained space.

$$Z_l^k = g_p\left(F_l^k\right) \tag{7}$$

Equation (7) illustrates the pooling procedure, where Z_l^k stands for the pooling feature map of the l^{th} layer for the k^{th} input feature map, F_l^k, and $g_p(.)$ specifies the kind of pooling process. With the help of the pooling approach, a combination of traits that are resilient to translational shifts and mild distortions may be retrieved. The size of the feature map is decreased to an invariant feature set, which reduces overfitting and limits the network's complexity while boosting generalization. The CNN uses several different pooling formulations, including maximal, average, overlapping, spatial pyramid, and other formulations;

(3) Activation function

The identification of complicated patterns is aided by the activation function, which also serves as a decision-making function. By selecting the right activation function, learning may be accelerated. A compressed feature map's activation mechanism is specified by Equation (8).

$$T_l^k = g_a\left(F_l^k\right) \tag{8}$$

The result of the convolution, denoted by F_l^k in the equation above, is given to the activation function g_a (.), which adds nonlinearity and produces a modified output, denoted by T_l^k for the l^{th} layer;

(4) Fully connected layer

The fully connected layer performs feature aggregation and combines the learned features from different parts of the input image. This aggregation allows the network to capture higher-level patterns and relationships between features, leading to more complex representations. In this task, the fully connected layer's output is often used to make final predictions, and the burst assembly is performed.

3.1.2. RNN

A dynamic neural network called an RNN is used to solve time series issues. In contrast to MLP, RNNs have connections between neighboring hidden neurons. Temporal correlations between distant occurrences in the time dimension can be taken into consideration through these connections, which allows the time-dependent input data in the sliding window to be successively conveyed through the hidden unit structure of a typical RNN. The fundamental formulas pertaining to the typical RNN's structure are shown here. The output of the conventional RNN's hidden unit at time t is shown in Equation (9). The loss function at time t is represented by Equation (10) and is defined as the mean squared error (MSE). The derivative of the loss Lt with respect to the weights is shown in Equation (11).

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b) \tag{9}$$

$$L_t = \frac{1}{2}\Big(u_t - p_t\Big)^2 \tag{10}$$

$$\frac{\partial L_t}{\partial W_x} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial p_t}\frac{\partial p_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_x} \tag{11}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} diag\big(\tanh'(h_j)\big)W_h \tag{12}$$

where the output vectors of hidden units at times t, j, and k are, respectively, h_t, h_k, and x_t, and the input vector is at time t. The hidden layer's input and connected weight matrices for the output are denoted by the letters W_x and W_h. The term with bias is b. The hyperbolic tangent activation function is represented by $\tanh()$, which is $\tanh(x) = \frac{1-e^{2x}}{1+e^{2x}}$. On the main diagonal, $diag()$ yields a diagonal matrix containing the vector elements. The outputs of the CNN and RNN are combined to produce the load value of each VM. Then, the VMs are grouped according to their weights.

3.2. Grouping Virtual Machines Using Reinforcement-Learning-Based Hybrid Lyrebird Falcon Optimization

The VMs are divided into overloaded and unloaded clusters based on the computed loads.

A threshold-based strategy is used to separate virtual machines (VMs) into clusters that are overloaded and unloaded according to calculated loads. A load threshold value is established. Virtual machines (VMs) that have loads over this threshold are deemed overloaded, while those that have loads below it are deemed unloaded. The threshold can

be changed in accordance with the particular needs and features of the cloud computing environment. Let us indicate that L_i is VM_i's calculated load.

The threshold points out whether identifying clusters are overcrowded or not. This is an expression for the equation that separates virtual machines (VMs) into overloaded and unloaded clusters.

$$Overload\ VMs = \{i | L_i > Load\ Threshold\} \tag{13}$$

$$Underload\ VMs = \{i | L_i \leq Load\ Threshold\} \tag{14}$$

This basic logic classifies virtual machines (VMs) according to whether their computed load exceeds or falls below the given threshold. One can define what constitutes an overloaded or unloaded condition more freely by varying the value of the load threshold. To prevent any PM from being overworked and others from being underutilized, the objective is to strike a balance in the allocation of VMs among PMs.

3.2.1. Optimized Reinforcement-Learning-Based Clustering

The modeling of the relationship between the general agent and atmosphere (in both the classic and modern RILs) is composed of an element, a set of accessible measurements A, and rewarding functions, $S \times A \rightarrow R$, as shown in Figure 3. The handler is the one who makes decisions. As it functions, the communication process ought to be conditioned. The environment or the world outside the system is the medium through which the agent communicates.

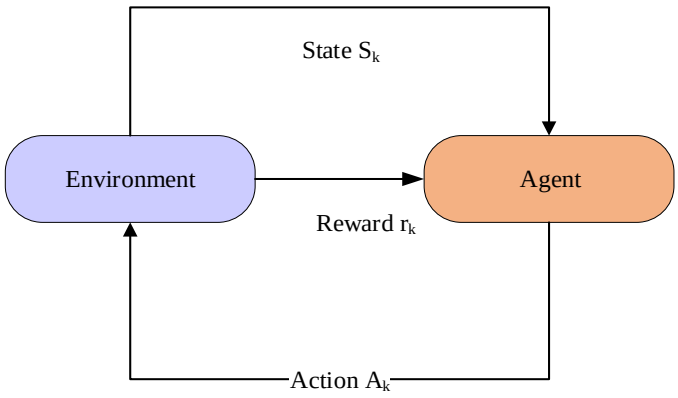

Figure 3. Basic diagram of the RL.

The interaction of the agent with the environment is called a continuous phase. Throughout the whole judgment process, the agent bases its decisions (a_k) on the current condition (s_k) of the surroundings. A newly updated state $(s_k + 1)$ environment is given to the agent for future decisions once the decision has been made. The system must then accept the decision and make the necessary modifications. The agent attempts throughout time to maximize cumulative rewards, while the environment also provides the agent with reward r_k according to choice a_k. There must be a clear input system for the agent to comprehend its optimal behaviors and regulations. When the system starts with state s and follows action a (and specific policy accordingly), it is predicted that the $Q(s,\ a)$ value function would accrue (with discounts). This is what the agent attempts to maximize via Q-learning. It is the typical RIL method in this instance. For continuous-time systems, $Q(s,\ a)$ is defined as follows:

$$Q(s,a) = E\left[\int_{t_0}^{\infty} e^{-\beta(t-t_0)} r(t)dt \middle| s_0 = s, a_0 = a\right] \tag{15}$$

The reward rate function is denoted by $r(t)$, whereas the discount rate is represented by β. Using an event-driven approach across time, Q-learning is an online reinforcement

learning adaptive strategy that lowers overhead associated with periodic updating of the RL discrete time value. Q-learning uses a continuous-time formulation of the value function $Q(s,\ a)$, which is provided in Equation (16). An equivalent approach to discrete RIL methods is used by the RL firm, which consistently adopts avaricious practices. This is where the updating rule is provided; caused by state transition, decision epoch t_{t+1} is defined as follows:

$$\begin{aligned} eQ^{(k+1)}(s_k, a_k) & \\ & \leftarrow Q^{(k)}(s_k, a_k) + \alpha.\ [\tfrac{1-e^{-\beta\tau_k}}{\beta}.r(s_k, a_k) \\ & +\max_{a'} e^{-\beta\tau_k} Q^{(k)}(s_{k+1}, a') - Q^{(k)}(s_k, a_k)] \end{aligned} \tag{16}$$

where $r(s_k, a_k)$ is the reward function; τ_k is the sojourn time where the RL is said to lie in a state s_k prior to the occurrence of the transition; and $\alpha \leq 1$ is the learning rate. $Q^{(k)}(s_k, a_k)$ is the estimation decision value at each iteration τ_k. In contrast to earlier research, this section offers a generalized RL strategy that may be used for resource allocation and other issues. The HLFO offline phase and the Q-learning phase online make up the RIL approach. The connection between the regulated action pair $(s,\ a)$ and its $Q(s,\ a)$ value function is assessed using HLFO during the offline procedure. To develop a sufficiently accurate RIL using measurement data, enough $Q(s,\ a)$ value estimations and associated $(s,\ a)$ samples must be gathered during the offline RL creation phase. Pre-processing the status transitional profile, the game playback profiles, and $Q(s,\ a)$, the estimated value for the gaming applications, are all part of this process. In order to obtain sufficient system transfer profiles and $Q(s,\ a)$ cost estimates—which can be a composite of latency, power consumption, and durability metrics for building the HLFO —the study leverages real-world task arrival points for the cloud resource allocation request. A randomly chosen policy and a progressively improved policy might come after this procedure. The conversion profiles are stored in storage D and placed in capacity ND. By using memory, parameter divergence is avoided and training is encouraged. The estimations of $Q(s,\ a)$ values and the stored state transformation profiles are used to train the HLFO. Every task's scheduling is estimated as $O(mn)$, meaning that each iteration's task scheduling requires $O\left(m^2 n\right)$ to estimate.

3.2.2. Clustering Based on Hybrid Lyrebird Falcon Optimization Algorithm

Falcons have unique and complex hunting behaviors, using both clear, clearly identifiable techniques and more elaborate, convoluted tactics while chasing and capturing their prey.

- Step 1

  During the initial phase, the FOA factors and the governing limits are initialized;

- Step 2

Following this, the motion and location of falcons are altered according to the supplied values, which is represented as follows:

$$y = \begin{bmatrix} y_{1,1} & \cdots & y_{1,V} \\ \vdots & \vdots & \vdots \\ y_{A,1} & \cdots & y_{A,D} \end{bmatrix} \tag{17}$$

where y is the falcon location regarding the total applicants A for every dimension V. The speed is generated randomly within the v_{Max} and v_{Min} limits.

$$v_{Max} = 0.1 Up_{li} \tag{18}$$

$$v_{Min} = -v_{Max} \tag{19}$$

where Up_{li} is the upper limit in every measurement;

- Step 3

Evaluation and identification of the global and personal optimal conditions for each falcon are carried out. The health value associated with the problem is obtained, generating the vector. Determination of the fitness value at each iteration uses Equation (20).

$$OF = \begin{bmatrix} of_1 \\ \vdots \\ of_n \end{bmatrix} \tag{20}$$

where the objective fitness is denoted by OF. At that point, each falcon's best individual location is represented by *xbest*, and the individual best is fixed as *gbest*;

- Step 4

During that period, the optimal individual is designated as I_b, and the optimal position for each falcon is denoted as y_b. Two random elements (Q_B, Q_C) are created with a normal distribution on every bird of attack to study the association between awareness and leap probability. The main probability considered where Q_B is smaller than the falcon is as follows:

$$y_t = y_{t-1} + v_{t-1} + M_t r\,(y_{b,\,t-1} - y_{t-1}) + O_t r(I_{b,\,t-1} - y_{t-1}) \tag{21}$$

y_{t-1} and v_{t-1} are present location and the falcon's motion. $M_t r$ and $O_t r$ are cognitive rate and social.

If Q_B exceeds B (adaptive prob), the jump is compared to Q_C. If Q_C exceeds C (dive prob), the falcon selects one prey (y_{ch}) and performs its hunting evolution using logarithmic twisting expressed as follows:

$$y_t = y_{t-1} + |(y_{ch} - y_{t-1}|\exp(ze)\cos(2\pi e) \tag{22}$$

where y_t is a new position, z is accurate observation logarithmic twisting that equates to 1, and e is an irregular value in the range $[-1, 1]$, which indicates how close the falcon is to its actual target;

- Step 5

Since the lyrebird is supposed to randomly escape to one of these safe zones, this is hybridized with the FOA to include the hunting behavior. The randomness of the lyrebird's escape makes it harder for predators to anticipate its movements, increasing its chances of survival. By incorporating elements of the falcon's hunting, which relies on surprise and agility, the lyrebird could further confuse and outmaneuver its pursuers. Common aspects of a falcon's hunting are spotting and taking advantage of weaknesses in its target. By combining this with its escape plan, the lyrebird may be able to draw predators into situations where they are vulnerable and increase the likelihood of escape.

Using Equation (22), a new location is determined for each LOA member based on the lyrebird displacement modeling conducted in this phase. Subsequently, Equation (23) states that this new location takes the place of the corresponding member's prior position if the value of the goal function is enhanced.

$$y_t = y_{t-1} + rand\cdot(SSA_t - I_t.y_{t-1}) \tag{23}$$

In this instance, SSA_t represents the safe area that has been chosen; y_{t+1} signifies the new position that has been based on the escaping strategy of the proposed LOA; *rand* is the random numbers from the interval [0, 1]; and I_t is the number that has been randomly selected as 1 or 2. When Q_B is lower than Q_C, the fitness of the picked prey is compared to the falcon's fitness, and this condition is expressed as follows:

$$y_n = y_{t-1} + v_{t-1} + f_c r(y_{ch} - y_{t-1}) \tag{24}$$

3.3. Task Scheduling Using a Multi-Objective Hybrid Optimization Model

After forming the VM clusters, analysis of each cluster is performed to identify the underloaded VMs. Underloaded VMs are VMs with spare capacity that can handle additional tasks. A Multi-Objective Hybrid Optimization model that takes into account each of the specified objectives by HLFO is constructed. Finding the optimal VMs to assign work to while concurrently considering numerous objectives should be the goal of the model. To improve user experience and adhere to Service-Level Agreements (SLAs), QoS measurements including response time, throughput, and availability are used.

4. Result and Discussion

The experimental assessment of the proposed load-balancing technique in a cloud system environment is described in this part. There are several subsections in this section. The suggested cloud environment's specifics are given in the simulated environment, and each important matrix is discussed in the performance matrices. The comparative analysis section compares the proposed approach with current scheduling and load-balancing techniques. The experimental setup is given in Table 2.

Table 2. Experimental setup.

Experimental Setup	Description
Software	Python 3.11.1
Simulation Toolkit	CloudSim 4.0
Cloud Environment Type	Simulated
VMs	10 to 50
PMs	1
Number of Tasks	100 to 500

4.1. Performance Metrics

(1) Makespan

Makespan is the amount of time required to complete each task or activity in a system. It is highly important in job management and scheduling when the goal is to minimize the total amount of time needed to complete all tasks. Minimize makespan to ensure efficient resource allocation and Service-Level Agreements;

(2) Energy Consumption

The fundamental objective of energy consumption is to reduce the overall power utilization of a cloud infrastructure. Sustainability and cost effectiveness rely on it. There are several tactics to optimize energy consumption, including server consolidation, workload distribution, and dynamic resource scaling;

(3) CPU Utilization

CPU utilization measures how well CPUs are used in cloud computing environments. CPU utilization balancing ensures that no PMs or VMs are overworked while others remain idle. Performance bottlenecks may be avoided and hardware resource efficiency can be maximized with appropriate CPU use;

(4) Memory Utilization

Memory utilization assesses how efficiently PMs and VMs use their memory resources. Reducing memory usage imbalances helps avoid memory bottlenecks, which can lead to system slowdowns. An effective utilization of RAM improves overall performance and system responsiveness;

(5) Task Prioritization

The process of prioritizing tasks involves determining which jobs or activities inside the system are most vital or significant. High-priority activities are certain to receive the necessary resources and to be completed on schedule, even during periods of extreme workload. When it comes to managing important business operations or customer requests, for example, setting priorities is essential to meeting needs and maintaining service quality.

4.2. Overall Performance of the Proposed Model by Varying the Task Count

Table 3 presents a comparative analysis of four models—Reinforcement Learning (RL), LOA, FOA, and HLFO (proposed)—across different task counts, highlighting key performance metrics. The makespan column indicates the time taken to complete tasks, with RL showing a makespan of 482 units for 100 tasks. Energy consumption is depicted in the RL model as 51.62 units for the same task count. Balanced CPU utilization, measuring evenness in CPU usage, is exemplified by RL with a value of 0.0168. Optimized memory usage, indicating memory efficiency, is shown as 5313.61 units for RL with 100 tasks. Task prioritization, denoted by numerical values, is presented as 5930 for RL with 100 tasks. The table facilitates a comprehensive understanding of each model's performance under varying task counts, encompassing factors such as time efficiency, energy consumption, CPU utilization balance, memory optimization, and task prioritization strategies.

Table 3. Comparison of the performance for various task counts.

		Task				
Model	**Task**	**Makespan**	**Energy Consumption**	**Balanced CPU Utilization**	**Optimized Memory Usage**	**Task Prioritization**
RL	100	482	51.61574154	0.0168	5313.6104	5930
	200	1091	66.6182419	0.020866	5404.4675	29,800
	300	1788	81.97394104	0.021016	5473.8971	44,850
	400	2902	84.13317268	0.0935	5619.417494	49,800
	500	2457	96.9864314	0.01256	5721.105324	52,750
LOA	100	506	66.15810703	0.02424	6029.2364	5950
	200	1097	69.63142331	0.01474	6380.0291	30,900
	300	1769	74.86529995	0.02096	6573.920833	45,850
	400	2154	75.98325648	0.04096	6879.920833	50,697
	500	2365	78.73125647	0.06096	6943.920833	53,954
FOA	100	601	79.85247	0.03542	6125.3697	6025
	200	1165	80.36542	0.056487	6596.32	42,238
	300	1874	82.95423	0.02465	6685.68	63,375
	400	2187	84.74295	0.025463	6896.74	64,481
	500	2396	86.98452	0.036214	7098.32	66,598
HLFO (proposed)	100	299	42.72476182	0.003376	4893.0531	6950
	200	1013	45.17180384	0.004606	4915.504375	50,900
	300	1546	48.27371066	0.001246222	5085.768622	70,850
	400	1972	55.41229177	0.012035	5241.274244	80,800
	500	2015	61.9782497	0.009344	5383.952204	81,750

(1) Makespan

The provided table compares the performance of different models (RL, LOA, FOA, and HLFO) across various task counts based on several metrics. The "Makespan" values represent the total time taken to complete tasks in each scenario, shown in Figure 4. In the RL model, makespan increases from 482 (for 100 tasks) to 2457 (for 500 tasks), indicating a proportional rise in completion time with an increasing task load. The LOA and FOA follow similar trends, with makespan values escalating as the number of tasks grows. Notably, the proposed HLFO model stands out with significantly lower makespan values across all task counts (e.g., 299 for 100 tasks), showcasing its superior efficiency in task completion compared to the other models. The HLFO model's consistently lower makespan suggests its potential for optimizing task scheduling and resource allocation, resulting in faster task completion times and improved overall system efficiency;

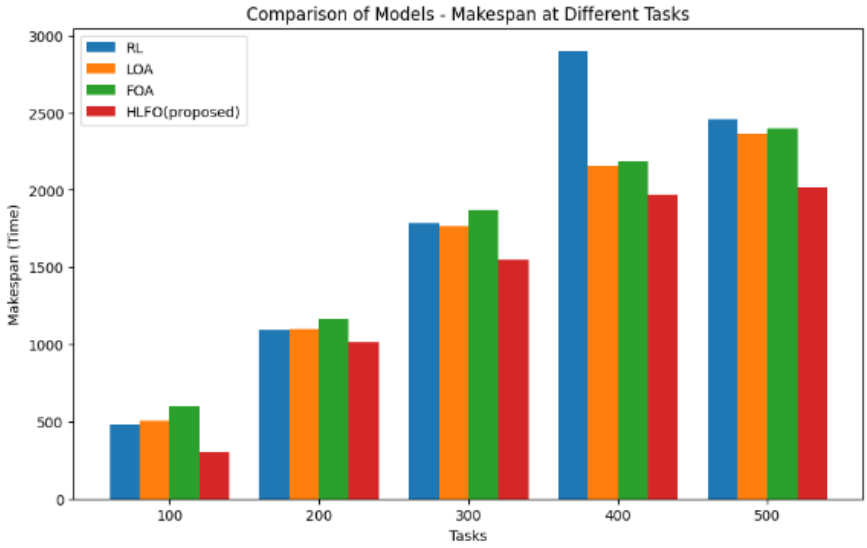

Figure 4. Comparison of the makespan by varying task.

(2) Energy consumption

The "Energy Consumption" values in the table depict the amount of energy consumed by various models (RL, LOA, FOA, and HLFO) in executing tasks across different counts. For RL, the LOA, and the FOA, there is a discernible upward trend in energy consumption as the task count increases, indicating an increased energy demand for handling larger workload scenarios, shown in Figure 5. Specifically, RL's energy consumption rises from 51.62 for 100 tasks to 96.99 for 500 tasks. In contrast, the proposed HLFO model consistently exhibits lower energy consumption values across all task counts, underscoring its potential for energy-efficient task execution. For instance, HLFO achieves a notably lower energy consumption of 42.72 for 100 tasks, positioning it as a promising model for scenarios prioritizing energy conservation in task scheduling and optimization;

(3) Balanced CPU utilization

The "Balanced CPU Utilization" values in the table gauge the efficiency of computational resource allocation among different models (RL, LOA, FOA, and HLFO) across varying task counts. For RL, there is a noticeable increase in balanced CPU utilization as the task count rises, implying a more efficient distribution of computational load with larger workload scenarios, shown in Figure 6. The LOA exhibits a fluctuating trend in balanced CPU utilization, while the FOA maintains relatively consistent values. Remarkably, the proposed HLFO model consistently achieves remarkably low balanced CPU utilization values, such as 0.003376 for 100 tasks, suggesting an exceptionally balanced distribution of CPU workload;

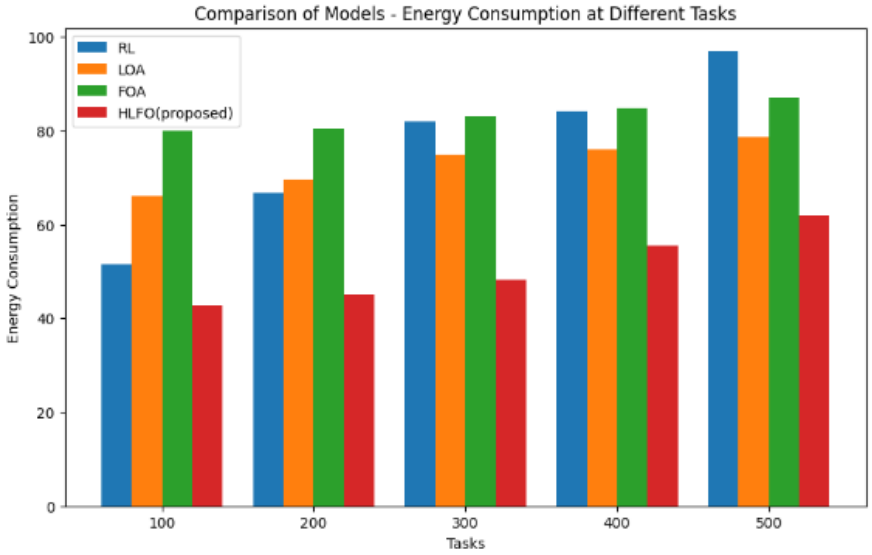

Figure 5. Comparison of the energy consumption by varying task.

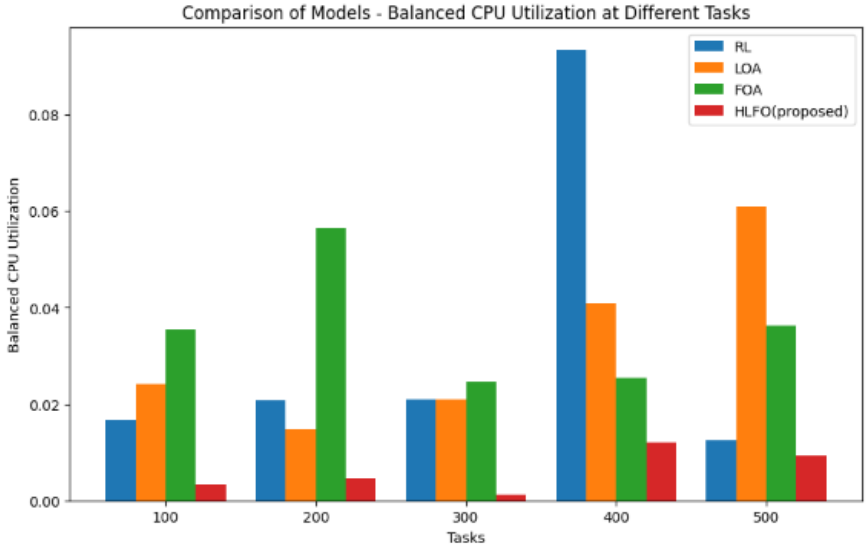

Figure 6. Comparison of the balanced CPU utilization by varying task.

(4) Optimized memory usage

The "Optimized Memory Usage" values in the table signify the efficiency of memory allocation among different models (RL, LOA, FOA, and HLFO) across varying task counts. In the cases of RL, the LOA, and the FOA, there is a consistent increase in optimized memory usage as the task count rises, reflecting a proportional allocation of memory resources to accommodate larger workload scenarios, shown in Figure 7. Conversely, the proposed HLFO model consistently demonstrates lower optimized memory usage values across all task counts, such as 4893.05 for 100 tasks, indicating a more resource-efficient approach to memory utilization. These findings suggest that the HLFO model may excel in optimizing memory resources even in scenarios with fewer tasks, potentially contributing to enhanced system efficiency and responsiveness by avoiding unnecessary memory allocation compared to other models;

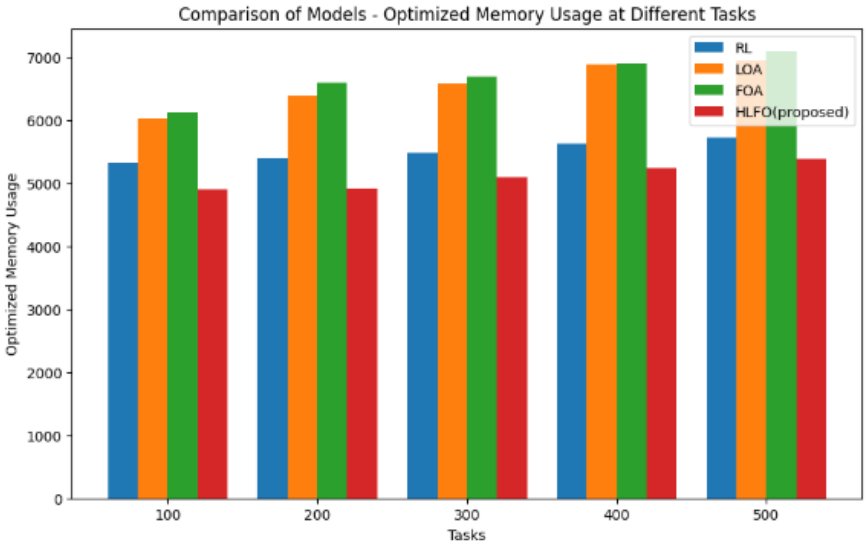

Figure 7. Comparison of the optimized memory usage by various tasks.

(5) Task prioritization

The combined examination of "Optimized Memory Usage" and "Task Prioritization" values in the table offers a holistic perspective on how different models (RL, LOA, FOA, and HLFO) manage memory resources and prioritize tasks across various workload scenarios, shown in Figure 8. For RL, the LOA, and the FOA, there is an observable increase in optimized memory usage as the task count rises, indicative of proportional memory allocation. While explicit "Task Prioritization" values are not provided, they would complement insights into how effectively these models prioritize tasks for optimal memory utilization. In contrast, the proposed HLFO model consistently demonstrates lower optimized memory usage values across task counts, suggesting a more efficient approach to memory management. The missing "Task Prioritization" values for HLFO, if available, would provide further understanding of how task prioritization strategies contribute to its optimized memory usage.

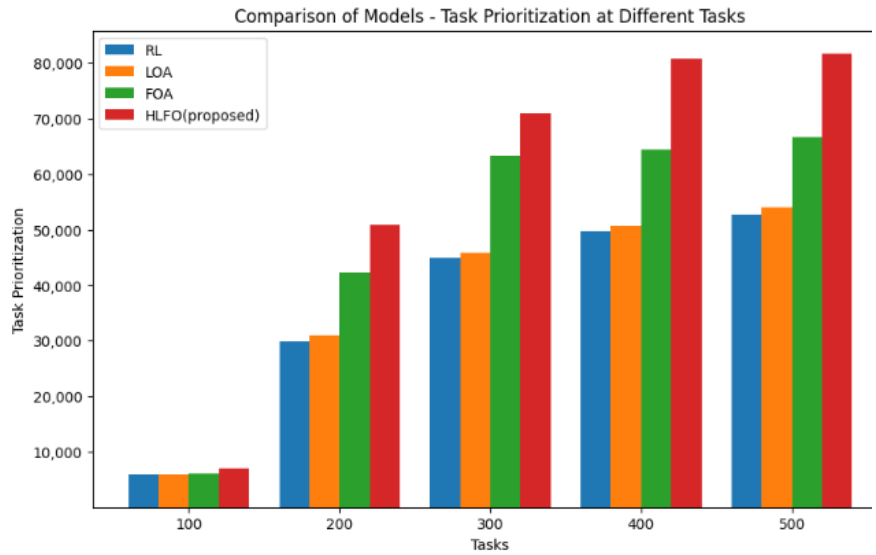

Figure 8. Comparison of the task prioritization by varying task.

4.3. Overall Performance of the Proposed Model by Varying the Task Count

The provided table compares the performance of different models (RL, LOA, FOA, and HLFO) across various virtual machine (VM) counts based on several metrics. Each row corre-

sponds to a specific model, and the columns represent the VM count, makespan, energy consumption, balanced CPU utilization, optimized memory usage, and task prioritization values.

As the number of VMs increases from 10 to 50, RL shows an increase in makespan (from 65 to 75), energy consumption (from 6.05 to 13.06), and balanced CPU utilization (with fluctuations). Optimized memory usage increases from 6019.16 to 7025.12. Task prioritization values also increase, indicating a preference for specific tasks. The LOA exhibits similar trends with increasing VM count. Makespan rises from 67 to 73, energy consumption increases from 7.05 to 12.01, and balanced CPU utilization fluctuates. Optimized memory usage increases from 6123.24 to 7084.62. Task prioritization values also show an upward trend. The FOA displays a trend of increasing makespan (from 68 to 75), energy consumption (from 8.37 to 13.87), and balanced CPU utilization. Optimized memory usage rises from 6236.85 to 7498.36. Task prioritization values also increase, indicating changing task priorities with higher VM counts. HLFO consistently demonstrates lower makespan values (from 43 to 37) and energy consumption (from 4.39 to 5.60) compared to other models. Balanced CPU utilization and optimized memory usage vary with VM count. Notably, HLFO exhibits a distinctive decrease in optimized memory usage from 5012.29 to 1642.64 as VM count increases, and task prioritization values show variability.

(1) Makespan

The makespan, as indicated in Table 4, serves as a crucial metric for evaluating the efficiency of various computational models under different VM count scenarios, shown in Figure 9. It represents the total time required for a given model to complete its tasks. In the case of the Reinforcement Learning (RL) model, the makespan starts at 65 units for 10 VMs and gradually increases to 75 units for 50 VMs. Similarly, the Learning Automata Optimization (LOA) and Firefly Optimization Algorithm (FOA) models exhibit increasing makespan values with higher VM counts, reaching 73 units and 75 units, respectively, for 50 VMs. Notably, the proposed HLFO model stands out with significantly lower makespan values across the board. For instance, it achieves a makespan of 43 units for 10 VMs and maintains a relatively low value of 37 units even with 50 VMs. This suggests that the HLFO model excels in completing tasks more efficiently compared to the RL, LOA, and FOA approaches, making it a promising approach for optimizing computational workloads;

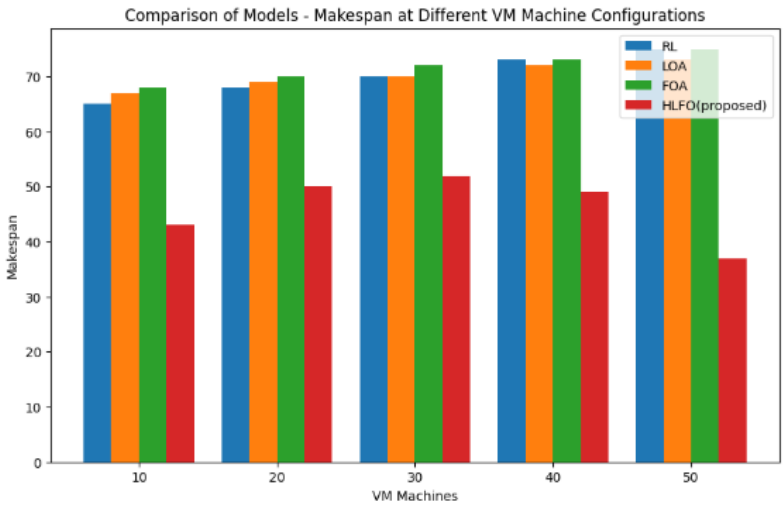

Figure 9. Comparison of the makespan by varying VM count.

(2) Energy consumption

Energy consumption, as shown in Table 4, quantifies the energy utilized by computational models across different VM count scenarios, shown in Figure 10. In the RL model, energy consumption ranges from 6.045057552 units for 10 VMs to 13.06201347 units for 50 VMs. Similarly, the LOA and FOA models exhibit increasing energy consumption with

higher VM counts, peaking at 12.00505755 and 13.86954712 units, respectively, for 50 VMs. Notably, the proposed HLFO model demonstrates more favorable energy consumption. For instance, it consumes 4.387614347 units for 10 VMs and maintains low values, like 5.598419936 units for 50 VMs;

Table 4. Comparison of the performance for various VM counts.

Model	VM Machine	Makespan	Energy Consumption	Balanced CPU Utilization	Optimized Memory Usage	Task Prioritization
RL	10	65	6.045057552	0.01	6019.16	45
	20	68	8.516720472	0.06	6570.69	55
	30	70	9.262013472	0.091	6666.24	60
	40	73	12.26201347	0.0125	6854.25	64
	50	75	13.06201347	0.0134	7025.12	65
LOA	10	67	7.045057552	0.0214	6123.24	76
	20	69	8.025057552	0.06475	6663.31	86
	30	70	9.015057552	0.01096	6786.52	89
	40	72	11.00505755	0.02163	6892.34	91
	50	73	12.00505755	0.032564	7084.62	93
FOA	10	68	8.369854127	0.03856	6236.85	77
	20	70	9.65487296	0.079856	6758.36	87
	30	72	10.857463	0.0269	6874.64	90
	40	73	12.9685765	0.03684	7236.98	92
	50	75	13.86954712	0.045489	7498.36	95
HLFO (proposed)	10	43	4.387614347	0.006	5012.29	80
	20	50	3.843706576	0.04475	4818.76	88
	30	52	5.373256934	0.0141	4619.01	94
	40	49	5.193933356	0.004475	3523.84	96
	50	37	5.598419936	0.0105	1642.64	97

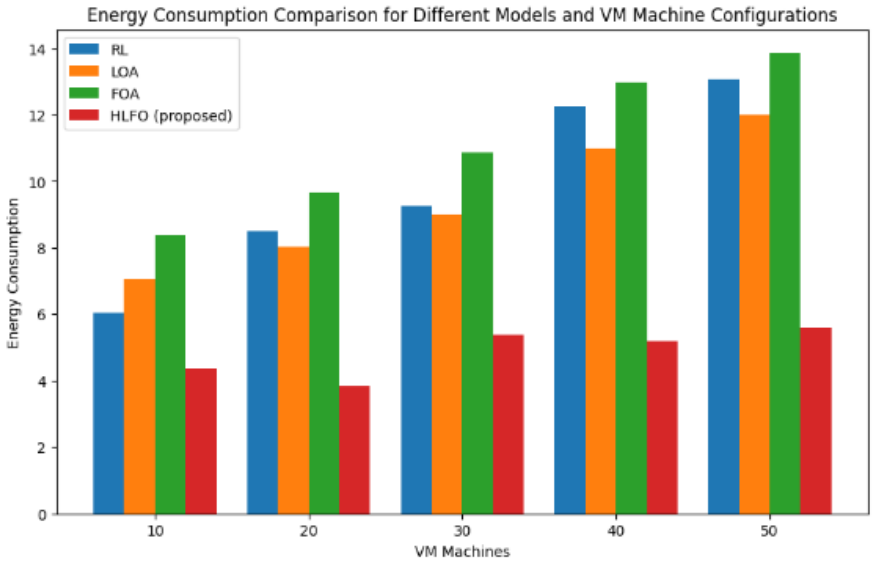

Figure 10. Comparison of the energy consumption by varying VM count.

(3) Balanced CPU utilization

In the case of the RL model, the balanced CPU utilization is denoted by values such as 0.01 for 10 VMs and 0.0134 for 50 VMs. Both the LOA and FOA models exhibit increasing balanced CPU utilization values with higher VM counts, reaching 0.032564 for the LOA and 0.045489 for the FOA for the 50 VMs scenario, shown in Figure 11. The proposed Hierarchical Learning Firefly Optimization (HLFO) model showcases notably low balanced CPU utilization across VM counts. For instance, it achieves values like 0.006 for 10 VMs and 0.0105 for 50 VMs. This suggests that the HLFO model efficiently distributes its processing load, maintaining a balance between CPUs and potentially avoiding overloading specific processors;

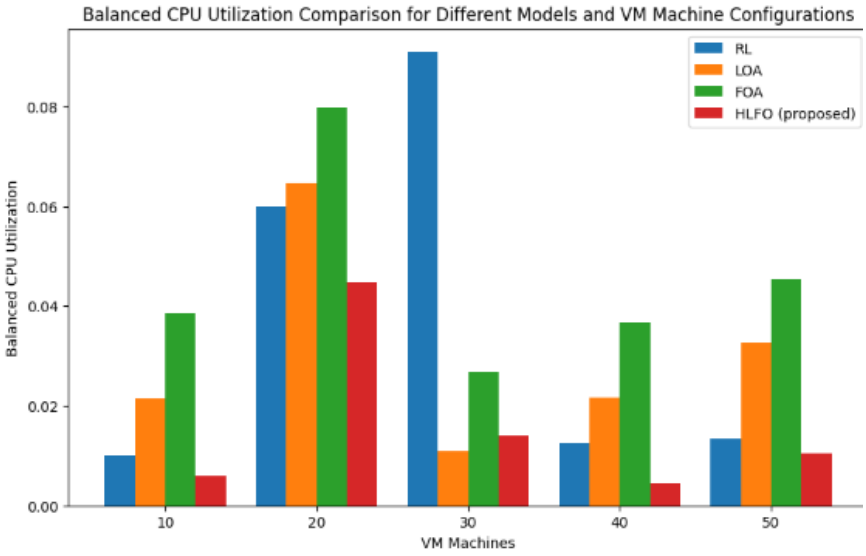

Figure 11. Comparison of the balanced CPU utilization by varying VM count.

(4) Optimized memory usage

Outlined in Table 4 is a measure of how efficiently computational models utilize system memory when performing tasks on different numbers of VMs. In the RL model, for instance, optimized memory usage is represented by values like 6019.16 for 10 VMs and 7025.12 for 50 VMs. The LOA and FOA models also show increasing optimized memory usage values as the VM count rises, with the LOA reaching 7084.62 and the FOA reaching 7498.36 for the 50 VMs scenario, shown in Figure 12. The proposed HLFO model demonstrates competitive optimized memory usage, with values such as 5012.29 for 10 VMs and 1642.64 for 50 VMs. Lower values of optimized memory usage suggest more efficient utilization of available memory resources, indicating that the HLFO model manages memory in a way that is conducive to effective task execution;

(5) Task prioritization

In the RL model, task prioritization is denoted by values like 45 for 10 VMs and 65 for 50 VMs. Similarly, the LOA and FOA models exhibit increasing task prioritization values with higher VM counts, reaching 93 for the LOA and 95 for the FOA in the 50 VMs scenario, shown in Figure 13. The proposed HLFO model showcases competitive task prioritization, with values such as 80 for 10 VMs and 97 for 50 VMs. Higher task prioritization values indicate a model's ability to effectively order and execute tasks, potentially leading to improved overall system performance.

The comparative analysis of four models—RL, LOA, FOA, and the HLFO—reveals nuanced performance variations across different task counts, shown in Table 3. RL demonstrates competitive performance in terms of makespan, energy consumption, balanced CPU utilization, optimized memory usage, and task prioritization for 100 tasks. Table 4

extends the analysis to the impact of increasing VM counts on the RL, LOA, FOA, and HLFO models. RL and the LOA exhibit rising makespan, energy consumption, and balanced CPU utilization with higher VM counts, while the FOA displays similar trends with varying task prioritization values. Notably, HLFO consistently outperforms other models, showcasing lower makespan and energy consumption. However, HLFO exhibits distinctive fluctuations in optimized memory usage and task prioritization values with increasing VM counts. In summary, the comprehensive evaluation of these models provides valuable insights into their performance under varying task and VM counts. RL stands out in specific metrics, while HLFO consistently demonstrates superior makespan and energy consumption. The proposed HLFO model showcases its effectiveness in achieving optimal system performance, balancing trade-offs across multiple performance metrics in cloud environments.

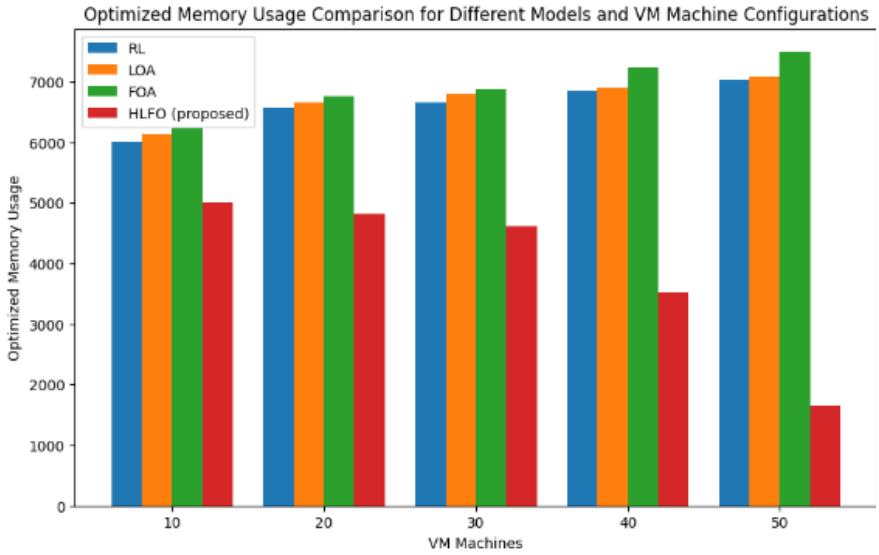

Figure 12. Comparison of the optimized memory usage by varying VM count.

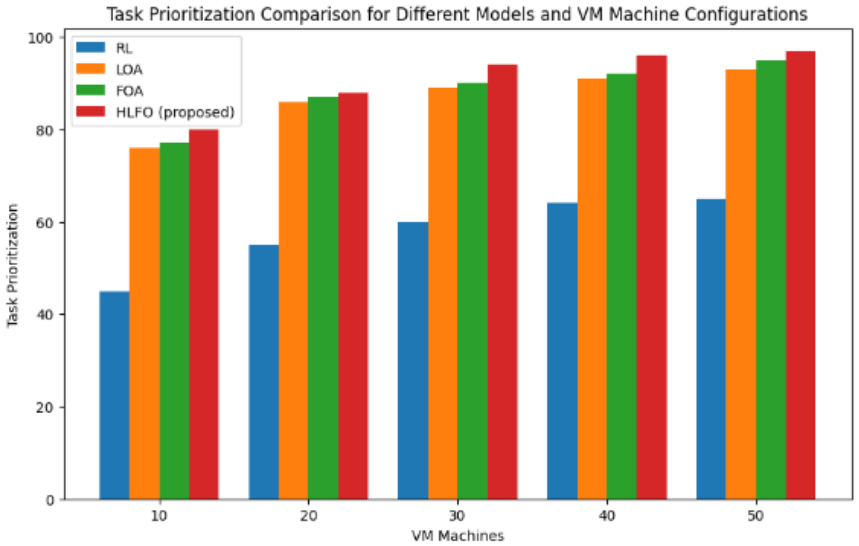

Figure 13. Comparison of the task prioritization by varying VM count.

5. Conclusions

In conclusion, this research introduces a pioneering dynamic load-balancing approach that addresses the imperative need for efficient task scheduling and stress distribution in cloud computing. The utilization of a deep learning framework, integrating CNNs and

RNNs, showcases a sophisticated methodology to compute and categorize VMs based on their loads into overloaded and underloaded clusters. The enhancement of clustering efficiency is achieved through the integration of RL with advanced hybrid optimization algorithms, exemplified by the HLFO. The proposed Multi-Objective Hybrid Optimization model optimizes task scheduling by considering QoS parameters, such as makespan minimization, energy consumption reduction, balanced CPU utilization, efficient memory usage, and task prioritization. Implemented in Python and CloudSim, the model demonstrates its capability to effectively allocate workloads between VMs and PMs, resulting in improved resource utilization, shortened makespan, enhanced CPU usage, and comprehensive assessments validating its efficacy. This research contributes a significant stride toward the refinement of dynamic load-balancing techniques, offering a promising avenue for advancing cloud performance in modern computing environments.

In future work, adaptive techniques for dynamically tuning the parameters of the deep learning, reinforcement learning, and hybrid optimization components will be developed. This could involve self-adjusting algorithms that optimize their own performance based on evolving workload characteristics.

Funding: The author extends the appreciation to the Deanship of Postgraduate Studies and Scientific Research at Majmaah University for funding this research work through the project number (R-2024-985).

Data Availability Statement: All the data is collected from the simulation reports of the software and tools used by the authors. Authors are working on implementing the same using real world data with appropriate permissions.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Dong, Y.; Xu, G.; Ding, Y.; Meng, X.; Zhao, J. A 'Joint-Me' Task Deployment Strategy for Load Balancing in Edge Computing. *IEEE Access* **2019**, *7*, 99658–99669. [CrossRef]
2. Maswood, M.M.S.; Rahman, M.R.; Alharbi, A.G.; Medhi, D. A Novel Strategy to Achieve Bandwidth Cost Reduction and Load Balancing in a Cooperative Three-Layer Fog-Cloud Computing Environment. *IEEE Access* **2020**, *8*, 113737–113750. [CrossRef]
3. Dong, Y.; Xu, G.; Zhang, M.; Meng, X. A High-Efficient Joint 'Cloud-Edge' Aware Strategy for Task Deployment and Load Balancing. *IEEE Access* **2021**, *9*, 12791–12802. [CrossRef]
4. Souravlas, S.; Anastasiadou, S.D.; Tantalaki, N.; Katsavounis, S. A Fair, Dynamic Load Balanced Task Distribution Strategy for Heterogeneous Cloud Platforms Based on Markov Process Modeling. *IEEE Access* **2022**, *10*, 26149–26162. [CrossRef]
5. Mondal, S.; Das, G.; Wong, E. A Game-Theoretic Approach for Non-Cooperative Load Balancing among Competing Cloudlets. *IEEE Open J. Commun. Soc.* **2020**, *1*, 226–241. [CrossRef]
6. Zhang, F.; Wang, M.M. Stochastic Congestion Game for Load Balancing in Mobile-Edge Computing. *IEEE Internet Things J.* **2021**, *8*, 778–790. [CrossRef]
7. Shojafar, M.; Canali, C.; Lancellotti, R.; Abawajy, J. Adaptive Computing-Plus-Communication Optimization Framework for Multimedia Processing in Cloud Systems. *IEEE Trans. Cloud Comput.* **2020**, *8*, 1162–1175. [CrossRef]
8. Zhao, D.; Mohamed, M.; Ludwig, H. Locality-Aware Scheduling for Containers in Cloud Computing. *IEEE Trans. Cloud Comput.* **2020**, *8*, 635–646. [CrossRef]
9. Zhang, F.; Deng, R.; Zhao, X.; Wang, M.M. Load Balancing for Distributed Intelligent Edge Computing: A State-Based Game Approach. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 1066–1077. [CrossRef]
10. Liu, C.; Li, K.; Li, K. A Game Approach to Multi-Servers Load Balancing with Load-Dependent Server Availability Consideration. *IEEE Trans. Cloud Comput.* **2021**, *9*, 1–13. [CrossRef]
11. Annie Poornima Princess, G.; Radhamani, A.S. A hybrid meta-heuristic for optimal load balancing in cloud computing. *J. Grid Comput.* **2021**, *19*, 21. [CrossRef]
12. Pang, S.; Li, W.; He, H.; Shan, Z.; Wang, X. An EDA-GA Hybrid Algorithm for Multi-Objective Task Scheduling in Cloud Computing. *IEEE Access* **2019**, *7*, 146379–146389. [CrossRef]
13. Rehman, A.U.; Ahmad, Z.; Jehangiri, A.I.; Ala'Anzy, M.A.; Othman, M.; Umar, A.I.; Ahmad, J. Dynamic Energy Efficient Resource Allocation Strategy for Load Balancing in Fog Environment. *IEEE Access* **2020**, *8*, 199829–199839. [CrossRef]
14. Jena, U.K.; Das, P.K.; Kabat, M.R. Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 2332–2342. [CrossRef]
15. Ebadifard, F.; Babamir, S.M. Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment. *Clust. Comput.* **2021**, *24*, 1075–1101. [CrossRef]

16. Shafiq, D.A.; Jhanjhi, N.Z.; Abdullah, A.; Alzain, M.A. A load balancing algorithm for the data centres to optimize cloud computing applications. *IEEE Access* **2021**, *9*, 41731–41744. [CrossRef]
17. Yu, D.; Ma, Z.; Wang, R. Efficient smart grid load balancing via fog and cloud computing. *Math. Probl. Eng.* **2022**, *2022*, 3151249. [CrossRef]
18. Devaraj, A.F.S.; Elhoseny, M.; Dhanasekaran, S.; Lydia, E.L.; Shankar, K. Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments. *J. Parallel Distrib. Comput.* **2020**, *142*, 36–45. [CrossRef]
19. Latchoumi, T.P.; Parthiban, L. Quasi oppositional dragonfly algorithm for load balancing in cloud computing environment. *Wirel. Pers. Commun.* **2022**, *122*, 2639–2656. [CrossRef]
20. Negi, S.; Rauthan, M.M.S.; Vaisla, K.S.; Panwar, N. CMODLB: An efficient load balancing approach in cloud computing environment. *J. Supercomput.* **2021**, *77*, 8787–8839. [CrossRef]
21. Pradhan, A.; Bisoy, S.K. A novel load balancing technique for cloud computing platform based on PSO. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 3988–3995. [CrossRef]
22. Sefati, S.; Mousavinasab, M.; Zareh Farkhady, R. Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: Performance evaluation. *J. Supercomput.* **2022**, *78*, 18–42. [CrossRef]
23. Mapetu, J.P.B.; Kong, L.; Chen, Z. A dynamic VM consolidation approach based on load balancing using Pearson correlation in cloud computing. *J. Supercomput.* **2021**, *77*, 5840–5881. [CrossRef]
24. Kruekaew, B.; Kimpan, W. Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning. *IEEE Access* **2022**, *10*, 17803–17818. [CrossRef]
25. Zeng, F.; Zhang, K.; Wu, L.; Wu, J. Efficient Caching in Vehicular Edge Computing Based on Edge-Cloud Collaboration. *IEEE Trans. Veh. Technol.* **2023**, *72*, 2468–2481. [CrossRef]
26. Paikrao, P.; Routray, S.; Mukherjee, A.; Khan, A.R.; Vohnout, R. Consumer Personalized Gesture Recognition in UAV Based Industry 5.0 Applications. *IEEE Trans. Consum. Electron.* **2023**, *69*, 842–849. [CrossRef]
27. Khan, A.R. Using virtualized multimedia tools for video conferencing solution integrated in teaching and learning environment. *J. Discret. Math. Sci. Cryptogr.* **2022**, *25*, 801–815. [CrossRef]
28. Khan, A.R. Secure PAAS environment over hybrid cloud using load-balanced Docker containers. *Int. J. Adv. Appl. Sci.* **2022**, *9*, 133–141. [CrossRef]