Reinforcement Learning Algorithms for Adaptive Load Balancing in Publish/Subscribe Systems: PPO, UCB, and Epsilon-Greedy Approaches

Rana Zuhair Al Shaikh*, Muna M. Jawad Al-Nayar, Ahmed M. Hasan
University of Technology, Iraq
E-mail: cse.22.11@grad.uotechnology.edu.iq , Muna.m.jawad@uotechnology.edu.iq , 60163@uotechnology.edu.iq
*Corresponding author

Keywords: load balancing, reinforcement learning, pub/sup system

Received: August 12, 2024

This research addresses load balancing challenges in publish/subscribe (Pub/Sub) systems by comprehensively exploring reinforcement learning (RL) techniques. Algorithms such as epsilon-greedy, Upper Confidence Bound (UCB), round-robin, and least connections were evaluated to establish baseline performance metrics. Building on this foundation, we develop enhanced versions of epsilon-greedy and UCB algorithms tailored to the Pub/Sub context. Additionally, we introduce a custom approach utilizing Proximal Policy Optimization (PPO) to learn adaptive load-balancing policies. Our work provides a thorough comparative analysis of diverse RL methods, offering insights into their strengths and weaknesses in optimizing Pub/Sub system performance. Experimental results demonstrate the potential of RL, particularly our developed algorithms, to improve performance significantly. These enhanced algorithms showed marked improvements in makespan, achieving completion times up to 30% faster than traditional methods. Moreover, they exhibited notable gains in throughput, particularly in the Burst Load scenario, where EUCB and PPO showcased a 10-15% increase in throughput compared to baseline algorithms. This research also highlighted the enhanced algorithms' superior ability to maintain high message success rates, exceeding 90% in most scenarios, and their contribution to more stable and predictable latency, leading to improved QoS. Notably, the PPO-based approach exhibits superior performance during burst traffic and failure scenarios, highlighting its resilience and adaptability in dynamic environments.

Povzetek: Razvit je nov pristop na osnovi globokega učenja za dinamično uravnavanje obremenitve v sistemih Pub/Sub. Modeli PPO, UCB in epsilon-greedy izboljšujejo učinkovitost, skrajšujejo čas obdelave in povečujejo prepustnost sporočil v dinamičnih okoljih.

1 Introduction

Load balancing enhances application performance and resource utilization by distributing incoming traffic across multiple servers [1] [2]. This is especially crucial in high-traffic environments since it ensures system responsiveness and reduces overload. [3]. In publish/subscribe (Pub/Sub) systems, efficient load balancing significantly affects messaging service reliability [4].

Load balancing strategies can be classified into static and dynamic [5] [6]. Static techniques employ established rules, frequently based on hashing algorithms [7], while dynamic approaches adjust to system variables such as server response delays in real time. Recent developments in dynamic algorithms provide greater efficiency and determinism for balanced load distribution [8].

Even though static algorithms are wildly deployed and robust for their swift response to direct the incoming flow to its distention, they still have some harmful features. The Round-Robin (RR) and Least Connection (LC), as examples used in this work as static algorithms for compression, are blind algorithms. In the case of RR, cyclic distribution can be problematic when subscriber loads or capabilities change dynamically; RR will continue to send it an equal share of messages, potentially leading to delays and performance degradation [9]. Moreover, RR doesn't consider each subscriber's capacity or processing power. This can result in some subscribers being underutilized while others are overwhelmed, leading to inefficient resource allocation. While LC's focus on the number of active connections can make it sensitive to short-term fluctuations in traffic[10]. A subscriber might temporarily have fewer connections due to a recent burst of messages, but it might still be heavily loaded regarding CPU or memory usage. LC might mistakenly send more messages to this subscriber, exacerbating its load imbalance. Also, LC assumes all subscribers have equal capabilities. In reality, subscribers might have different hardware configurations, software versions, or network connectivity, leading to varying processing speeds. LC's simplistic approach can lead to

suboptimal performance in such heterogeneous environments.

The shortcomings of static load balancing methods, particularly in scenarios with fluctuating traffic patterns and heterogeneous subscriber capabilities, motivate the investigation of Reinforcement Learning (RL) as a potential solution to overcome these limitations. The RL has recently emerged as a practical approach for dynamic load balancing across many domains. Its adaptability with fluctuating network conditions makes it well-suited for optimizing communication load balancing [11]. RL approaches are actively investigated in distributed systems to enhance parallel particle tracing [12] and enhance cloud resource allocation, hence improving system performance [13] [14].

While load balancing has been extensively studied in various computing domains, its application within pub/sub systems remains relatively unexplored, as exemplified by the limited research in this area. Existing works primarily focus on enhancing scalability and performance through techniques like vertical clustering [15] [16] [17], flooding-based message dissemination [18], and SDN-controlled multicast groups [19] [20]. These approaches, however, often lack the adaptability and efficiency required for large-scale, dynamic IoT environments. Additional research has investigated dynamic load-balancing techniques within networks, offering valuable insights into modern approaches in this field. Some studies, such as [21] [22], leverage reinforcement learning to improve load balancing and network performance. Others propose hybrid algorithms and optimization techniques, as exemplified by [23] [24]. Table 1 concisely summarizes these studies, highlighting their main contributions, the metrics employed, their key achievements, and any identified limitations.

Many researchers aim to enhance the performance of networks and task scheduling by mixing static and dynamic load balance, as in [25][26]. Others use optimization algorithms such as Particle Swarm Optimization or the Cuckoo algorithm in [27] [28] [29] [30].

While previous research has primarily focused on load balancing across distributed brokers in Pub/Sub systems, our work addresses the equally important challenge of optimizing load balancing within a single broker, particularly in scenarios where a distributed architecture might be impractical or introduce unnecessary complexity.

The present work aims to develop and evaluate an RL-based algorithm for load balancing in pub/sub systems. We seek to enhance existing algorithms like epsilon-greedy and UCB and introduce a custom PPO-based approach. Our objective is to demonstrate the effectiveness of these algorithms in improving system performance metrics such as makespan, throughput, Quality of service (QoS), performance, and system latency, particularly under dynamic conditions.

The remainder of this paper is organized as follows. Section 2 provides an overview of publish/subscribe systems, traditional load balancing methods, and the potential of RL in this domain. Section 3 details our methodology, including the system model, RL framework, algorithm descriptions, and experimental design. In Section 4, we present and analyze the results of our experiments, comparing the performance of different RL algorithms and highlighting the benefits of our PPO-based approach. Finally, Section 5 concludes the paper with a summary of our findings, their implications, and directions for future research.

Table 1: The summary table of related work

No.	Main Contribute	Metrics used	Main achievements	Limitation
[15]	Proposed a distributed MQTT broker system for large-scale location-based IoT applications. The system introduces a topic structure suitable for handling location-dependent data and distributed brokers and gateways to reduce broker load and support heterogeneous brokers.	Number of requests and received messages by brokers.	The proposed method outperforms existing broker implementations, especially when the number of subscribers is significant or frequent subscription changes occur.	The system's reliance on pre-configured gateway knowledge about brokers and their assigned areas might limit its adaptability to changes in the network topology or broker availability.
[16]	Investigated the sub-linear scalability of MQTT clusters, where adding more brokers doesn't lead to a linear increase in performance. Proposed a multi-session best-matching strategy to reduce inter-broker traffic and improve scalability.	Routing and forwarding overheads, latency, CPU usage, memory consumption, and network traffic.	Reduced scaling penalty from 40% to 10%, demonstrating improved performance and resource utilization in MQTT clusters.	the increased complexity on both the client and broker side, along with the potential overhead of multiple TCP/IP connections
[17]	Proposes a fog-based pub/sub system using dynamic broadcast groups to manage	Latency, Redundant Messages	Achieves low latency comparable to global flooding while	The group formation process may not lead to a globally optimal

	the trade-off between latency and excess data dissemination.		significantly reducing excess data compared to flooding, approaching the efficiency of cloud relay	topology. Additionally, the choice of flooding for intra-group communication might lead to excess data if broadcast groups become too large.
[20]	Presents a complete solution for creating a flexible and efficient distributed system of MQTT brokers, focusing on broker discovery, overlay creation, message routing, and topic-based optimization	Average end-to-end delay, Traffic overhead, CPU and RAM usage, Convergence and repair time	Achieves lower end-to-end delay compared to flooding, especially with topic-based overlay. Reduces traffic overhead significantly in most scenarios. Maintains similar resource usage to the benchmark under normal conditions, with increased CPU usage under stress.	The topic-based overlay approach can lead to increased signaling overhead due to the maintenance of multiple overlay networks. The system's performance under extreme stress conditions requires further optimization to manage CPU usage effectively.
[23]	Introduced SWARM, an adaptive load-balancing protocol for distributed streaming systems processing big spatial data. SWARM continuously monitors workloads, reacting to changes in data and query distributions.	Throughput, response time, resource utilization	Achieved higher throughput and lower response time than static partitioning, improving cluster utilization and handling larger volumes of spatial data and queries.	Tuple-at-a-time systems may not be suitable for micro-batched systems with higher latency requirements.
[21]	Explored deep reinforcement learning for intelligent load balancing, focusing on QoS parameters correlated with QoE. Proposed centralized and distributed solutions using actor-critic and multi-agent architectures.	QoS parameters (flow delivery delay, packet dropping, throughput) and QoE (for video).	Showed improved QoE compared to traditional methods (ECMP) and matched performance of QoE-based reward methods using only network-level QoS metrics.	Primarily based on simulations, the algorithm's complexity could pose challenges for real-world deployment.
[18]	Proposed a multi-objective optimization scheduling model using the Artificial Bee Colony algorithm (ABC) with a Q-learning algorithm called the MOABCQ method.	Makespan, cost, and resource utilization.	It outperforms other algorithms in terms of reducing makepan, cost, and degree of imbalance and increasing throughput, average, and resource utilization.	The algorithm's performance can vary depending on the dataset, and its complexity might be a limitation for large-scale environments.
[19]	Proposed an automatic load-balancing architecture based on reinforcement learning (ALBRL) in SDN. It adapts the improved Deep Deterministic Policy Gradient (DDPG) algorithm to find a near-optimal path between network hosts and generates an inter-link-weight matrix.	Network throughput, link-load-balancing factor, link utilization.	Faster training speed than existing reinforcement-learning algorithms and significantly improves network throughput.	Therefore, its reliance on a single network topology for evaluation indicates a poor ability to handle dynamic networks.
[24]	Proposed a Differential Grey Wolf (DGW) load balancing with stochastic Bellman deep reinforced resource optimization (DGW-SBDR) in fog environments. It uses a DGW Optimization algorithm	Load balancing efficiency, makespan, latency, and energy consumption.	Compared to benchmark methods, improved load balancing efficiency, makespan, latency, and energy consumption in fog environments.	Primarily based on simulations, the algorithm's complexity could pose challenges for real-world deployment.

	for optimal resource management and an SBDR Learning-based Resource Allocation Model for optimal resource allocation.			
[22]	Proposed an RL-based load balancer (MERL-LB) for financial cloud services to reduce idle servers without disconnecting users. It uses a scalable neural network policy and evolutionary multi-objective training to balance load imbalance and server idle time.	Load imbalance and server idle time.	Reduced idleness by over 130% compared to traditional methods while slightly improving load balancing. Offers diverse Pareto-optimal policies for user flexibility.	It can be computationally expensive. The algorithm's complexity, primarily based on simulations, could pose challenges for real-world deployment.

2 Background

The publish/subscribe system is a messaging pattern system that facilitates flexible and scalable communication in distributed applications [24]. Unlike traditional request-response models, pub/sub introduces a layer of indirection, where publishers send messages to designated topics or channels without knowing who might receive them. Subscribers express interest in specific topics and receive only the messages relevant to those topics. Decoupling publishers and subscribers enable flexibility and scalability, while topic-based filtering only ensures efficient communication [24].

However, pub/sub systems present challenges in designing and implementing effective load-balancing mechanisms. While simple, traditional methods such as round-robin (RR) and least connections (LC) have limitations. RR distributes messages cyclically without considering subscriber capacities or loads, potentially leading to suboptimal resource utilization. LC directs messages to the subscriber with the fewest active connections but may not be ideal when subscriber capabilities or message processing times vary significantly [25]. These methods generally lack adaptability to changing traffic patterns and system conditions, may not efficiently utilize resources, and may not consider the varying nature of message content.[10]

These limitations underscore the need for more intelligent and adaptive load-balancing techniques. With its ability to learn from interactions and dynamically adjust decisions based on real-time feedback, RL offers a promising avenue for overcoming these challenges [10]. In the following section, we discuss how RL can be leveraged to improve load balancing in pub/sub systems.

The Publisher is responsible for generating and sending messages (requests) and categorizing them into relevant topics. Subscribers express interest in specific topics and receive messages only for those topics. The Broker (or Messaging Server) acts as an intermediary, receiving messages from publishers, filtering them according to their topic, and delivering them to the appropriate subscribers [24].

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns optimal decision-making by interacting with an environment and receiving feedback as rewards or penalties [26]. Unlike supervised or unsupervised learning, RL does not rely on pre-labeled data or inherent data patterns. In load balancing in a pub/sub system, the RL agent would act as the message broker, the environment would be the dynamic state of the pub/sub system, and actions would involve selecting the appropriate subscriber for each incoming message. Rewards for the agent could be based on minimizing message response time and ensuring balanced utilization across subscribers.

The present work focuses on three RL algorithms for load balancing:

- Epsilon-greedy: This algorithm balances exploration (trying random actions to gather information) and exploitation (choosing the action with the highest estimated reward). A parameter ε controls the balance between these two modes [27].
- Upper Confidence Bound (UCB) favors actions that have been less explored or have previously shown potential for high rewards. It uses an optimistic estimate of the potential reward for each action to guide its decision [11].
- Proximal Policy Optimization (PPO): this policy gradient method aims to improve the current policy by taking small steps in the direction that maximizes the expected reward. It is known for its stability and sample efficiency [28].

By leveraging the learning capabilities of these RL algorithms, the broker can dynamically adapt its load-balancing decisions based on the current system state and the feedback received, potentially leading to more efficient and resilient load balancing than the traditional methods.

Having established the foundation of the Pub/Sub systems, traditional load balancing limitations, and the principles of RL, we now delve into the specific methodology employed in this research. The following section describes our system model, RL framework, algorithm enhancements, evaluation metrics, and experimental design.

3 The methodology

The Pub/Sub system is deployed on the Amazon Web Services (AWS) cloud, leveraging Elastic Compute Cloud (EC2) instances for its core components. A single t3. medium EC2 instance (2 vCPUs, 4GB RAM) hosts the custom Python message broker, which incorporates a load balancer capable of implementing the tested algorithms and manages internal queues for message distribution. Three EC2 instances are dedicated to the subscribers, each paired with a Nginx web server to handle incoming requests. The publisher component, however, runs locally on a separate machine, generating diverse traffic patterns analogous to using the JMeter tool to thoroughly evaluate the load balancer's performance.

Figure 1 illustrates the architecture of the Pub/Sub system, designed to emulate a real-world web application. Publishers send messages to the custom Python message broker, which employs a load balancer to intelligently route each message to the most suitable queue based on the chosen load-balancing algorithm. Each subscriber is hosted on a dedicated EC2 instance. It utilizes the Pika library and threading to efficiently consume messages from its assigned queue, similar to how Nginx workers handle concurrent requests. Upon receiving a message, the subscriber unpacks it into a web request format and forwards it to the co-located Nginx web server, which simulates the application logic and generates the appropriate response.

3.1 Reinforcement learning framework

Our load-balancing strategy is based on a reinforcement learning framework, where the broker acts as the agent, learning to make optimal load-balancing decisions. The following components characterize the RL framework:

- **States:** A vector representing the current state of each subscriber. The current load on each subscriber (e.g., active connections, CPU utilization, response time), queue length, and system-wide averages.
- **Actions:** The broker's decisions, such as which subscriber to assign a new message to.
- **Rewards:** Feedback signals that indicate the effectiveness of the broker's actions. The reward function is designed to incentivize both fast response times, queue length, and efficient resource utilization, calculated as shown in Equation (1):

Reward *= 1.0 + (1.0 / response_time) - (cpu_percent * 0.02)- (queue_length * 0.01)* ……Eq. (1)

This ensures the algorithm prioritizes both fast responses and efficient resource utilization. It starts with a base value of 1.0. This provides a positive reward even for somewhat slower response times. The term (1.0 / response_time) is added. It means shorter response times result in a more significant fraction, increasing the overall reward, and the longer response times lead to a smaller fraction, decreasing the reward. The (cpu_percent * 0.02) portion is subtracted from the reward, which means at 100% CPU usage, the reward is reduced by 2. Finally,

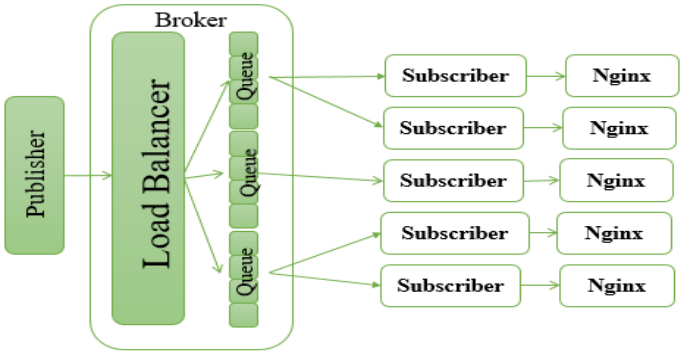

Figure 1: Proposed system design

(*Queue_length* * 0.01) penalizes the agent for selecting subscribers with longer queues, encouraging it to distribute the load more evenly.

Additionally, the adaptive exploration strategy is introduced. The adjusting of the exploration rate (ε) through the implementing Equation (2):

exploration rate *= 0.01+ sqrt (2 *log(total_counts) / (n_a + 1))* ------- *Eq. (2)*

The initial value of the exploration rate is 0.01 to prevent it to be zero at the start of the system. Where (total_counts) represents the total number of iterations, and (*n_a*) represents the number of times the specific action (*a*) has been taken. The natural logarithm aligns perfectly with the desired behavior of the exploration bonus. It emphasizes exploration early in learning and gradually reduces exploration over time.

Policy: The broker's strategy is to select actions based on the current state. RL aims to learn an optimal policy that maximizes the cumulative reward over time. Reinforcement Learning Algorithms

Three RL algorithms were evaluated in the present work for load balancing in Pub/Sub systems. These algorithms are as follows:

- **Epsilon-Greedy (EG)**

  Our enhanced epsilon-greedy algorithm Incorporates several modifications to improve its adaptability to dynamic Pub/Sub environments. The reward function dynamically considers both server

responsiveness and CPU load, as shown in Eq. (1). This ensures the algorithm prioritizes fast responses and efficient resource utilization. Additionally, we introduce an adaptive exploration strategy, adjusting the exploration rate (ε) using Equation (2).

- **Upper Confidence Bound (UCB) Algorithm**

The UCB algorithm shares a similar structure with epsilon-greedy but differs in its action selection strategy. Instead of relying on random exploration, UCB selects actions based on an upper confidence bound that considers the estimated Q-value and the uncertainty associated with that estimate [26]. This approach encourages the exploration of less-visited actions while exploiting actions with high expected rewards, leading to more informed and potentially superior decision-making. Figure 2 presents the general learning process for the epsilon-greedy and UCB algorithms within the proposed load-balancing framework. Both algorithms initialize parameters, compute the system state, and determine whether to explore or exploit. However, they differ in their action selection strategies. Epsilon-greedy explores with probability ε and exploits with probability 1-ε, while UCB uses a confidence interval-based formula to balance exploration and exploitation. Regardless of the algorithm, the chosen action is executed, a reward signal is received, and the Q-table is updated, enabling the agent to learn over time.

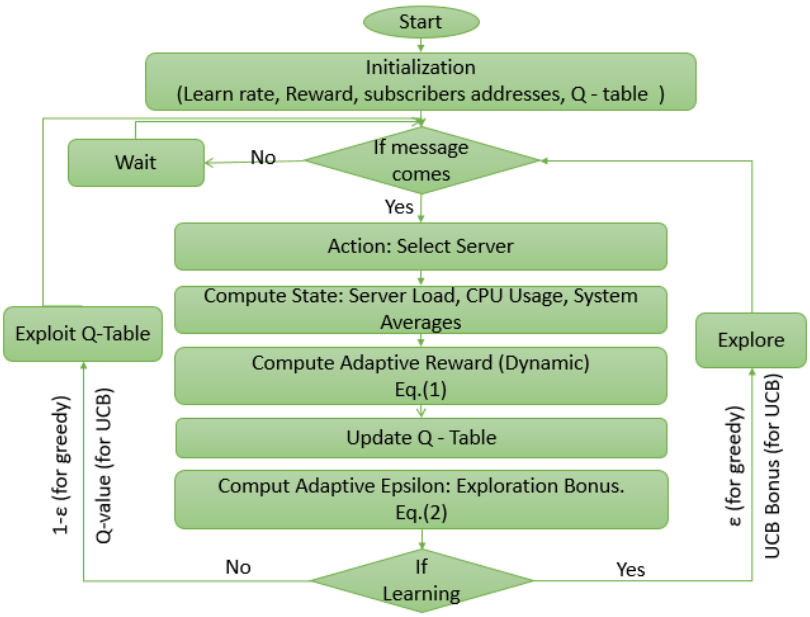

Figure 2: Flowchart of enhanced epsilon greedy and UCB algorithms for pub/sub system

- **Proximal policy optimization (PPO)**

Proximal Policy Optimization (PPO) is a policy gradient reinforcement learning algorithm renowned for its stability and efficiency. Unlike value-based methods like epsilon-greedy and UCB, PPO directly learns a policy that maps states to actions, connections, CPU utilization, response time, and queue length) for each subscriber and system-wide averages. To balance simplicity with adequate capacity for learning, the network utilizes two hidden layers with ReLU activation functions with 128 neurons each. Finally, an output layer produces a probability distribution over the available actions, effectively guiding the selection of a specific subscriber for each incoming message.

The reward function is crucial in guiding the PPO agent's learning process. The same dynamic reward function in Eq (1) is used.

Figure 3 illustrates the architecture of the Proximal Policy Optimization (PPO) agent employed for load balancing in the Pub/Sub system. The agent's neural network receives a state representation encompassing load metrics (active connections, CPU utilization, response time) for each subscriber, queue length, and system-wide averages as input. This state information is processed through two hidden layers with ReLU activation functions, culminating in an output layer that produces two key components: a policy, which is a probability distribution over possible actions (subscriber selections), and a value function, which estimates the expected cumulative reward from the current state.

The selected action, according to the policy's probabilities, is then executed within the Pub/Sub environment, specifically targeting one of the subscriber instances. The environment responds by

providing a reward signal, calculated using Equation (1), and a new state reflecting the updated system conditions. This reward and state information is fed back to the PPO agent. The clipped surrogate objective function utilizes them to update the policy and value function networks. This iterative process of interaction, reward collection, and network updates allows the PPO agent to learn and refine its

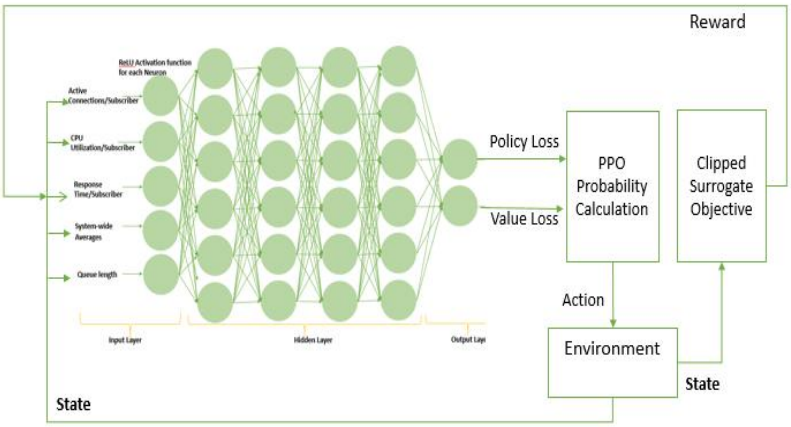

Figure 3: PPO Architecture for Load Balancing in Pub/Sub Systems

load-balancing strategy over time, adapting to the dynamic nature of the Pub/Sub system. The value function helps the agent estimate the long-term consequences of its actions, guiding it towards better decisions.

JMeter was employed to simulate different load scenarios and evaluate the system's performance under various conditions. The following four scenarios were designed:

- **Scenario 1: Normal load**. This scenario simulated a moderate traffic pattern using a Thread Group with 50 threads (representing 50 concurrent users) and a loop count of 20. A total of 1000 messages were sent to the message broker.
- **Scenario 2: Burst load**. This scenario aimed to stress the system with a sudden surge in traffic. It was implemented using a Thread Group with 100 threads and a loop count of 20, with target throughput 6000 message per minute.
- **Scenario 3: Server failure**. This scenario tested the system's resilience to subscriber failures. A normal load of 50 threads and 20 loops (1000 messages) was used, with one subscriber intentionally failing during the test.
- **Scenario 4: Heterogeneous instances**. This scenario evaluated the system's performance with subscribers having varying resource capabilities. A normal load was simulated, but one of the three subscriber EC2 instances was configured with reduced resources (1 vCPU and 2GB RAM). At the same time, the remaining two instances retained their original configuration (2 vCPU and 4GB RAM).

4 Result and discussion

This section presents the results of our experimental evaluation of various load balancing algorithms in a simulated Pub/Sub system deployed on AWS.

The performance of baseline algorithms (RR, LC, EG and UCB) are alkalized and compared with the enhanced algorithms (Enhanced Epsilon Greedy (EEG), Enhanced UCB (EUCB), and PPO) across four scenarios: normal load, burst load, server failure, and heterogeneous instances. Each experimental scenario was run 10 times, and the results are presented as mean values with 95% confidence intervals. The evaluation metrics include makespan, message throughput, latency, successful massage rate, Quality of Service, and efficiency.

4.1 Makespan comparison

The makespan, representing the total time to process all messages, is a critical indicator of system behavior. As depicted in Figure 4, the enhanced algorithms consistently outperform the baseline algorithms across all scenarios. This advantage stems from their ability to dynamically adapt to changing conditions. Unlike RR and LC, which rely on fixed rules, the enhanced algorithms leverage real-time feedback to make informed decisions. For instance, in the 'Failed Scenario,' the improved algorithms, especially PPO, demonstrate significantly lower makespans, indicating their superior ability to adapt and recover from disruptions.

4.2 Message throughput

Figure 5 presents a comparative analysis of throughput achieved by various load-balancing algorithms across the four distinct scenarios. Across all scenarios, the enhanced algorithms (EUCB and PPO) consistently outperform the other algorithms specially (RR, LC, and EG), demonstrating their superior ability to maintain high throughput even under challenging conditions. This is particularly evident in the Burst Load scenario, where EUCB and PPO exhibit significantly higher throughput than the other algorithms. The results highlight the effectiveness of the enhanced algorithms in maximizing

the system's capacity to process messages and maintain a high rate of successful message delivery.

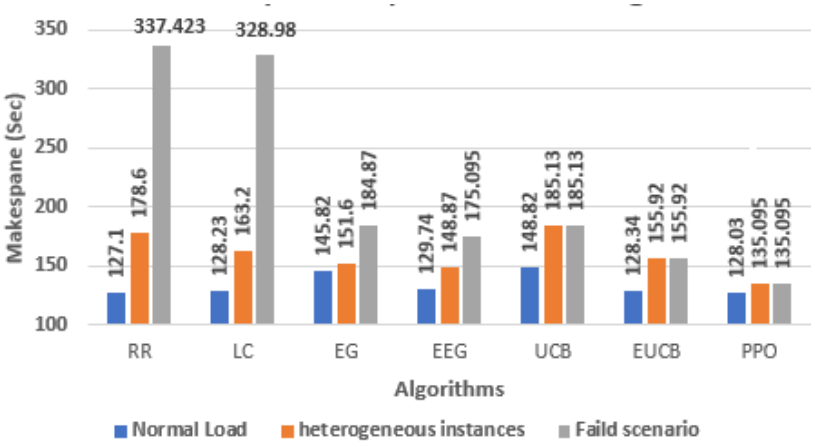

Figure 4: Makespan comparison across all algorithms

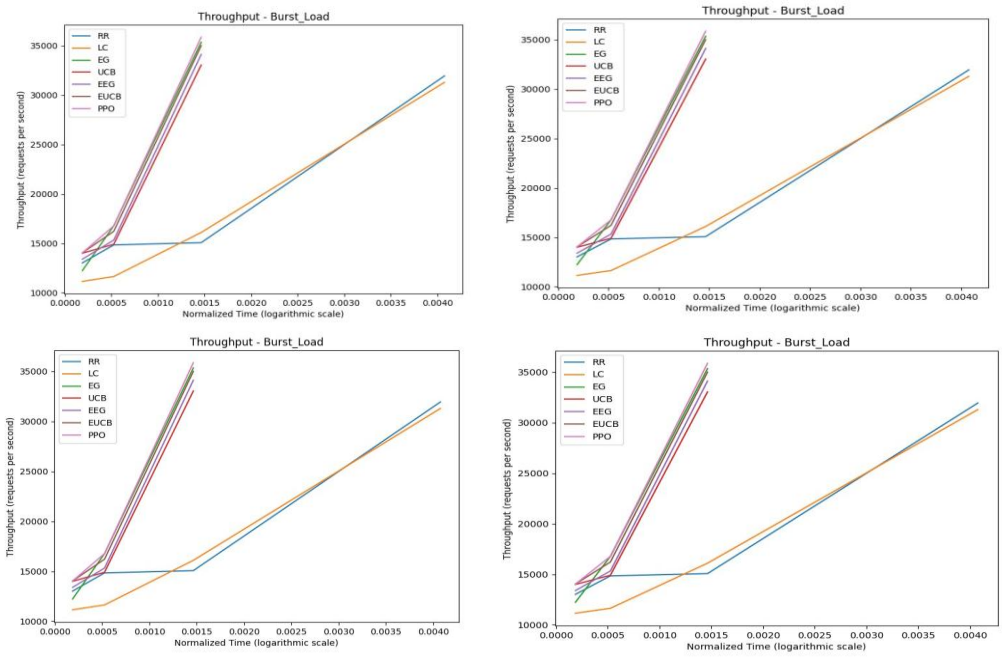

Figure 5: System message throughput across all algorithms

4.3 The latency

Figure 6 illustrates a comparative analysis of latency across the four scenarios. A key observation is the variability in latency for the baseline algorithms (RR, LC, EG, and UCB), particularly in the Normal Load and Burst Load scenarios. This indicates an inconsistency in response times and potential delays for specific messages. In contrast, the enhanced algorithms (EUCB and PPO) exhibit more stable and predictable latency across all scenarios, with tighter interquartile ranges and fewer outliers. This suggests these algorithms can deliver more consistent and reliable performance, even under challenging conditions. However, it's important to note that the enhanced algorithms generally show slightly higher median latency than the baseline algorithms, indicating a potential trade-off between performance consistency and raw speed.

4.4 Successful message rate

Table 2 clearly shows the percentage of successful messages achieved by each load-balancing algorithm across different scenarios. Notably, the enhanced UCB and PPO algorithms consistently demonstrate higher success rates than the other algorithms in all scenarios. This highlights their effectiveness in ensuring reliable message delivery and minimizing request failures, especially in challenging conditions like burst loads or

server failures. The PPO algorithm, in particular, exhibits the highest success rates across most scenarios, further emphasizing its potential for robust load balancing in Pub/Sub systems.

4.5 Quality of service (QoS):

Figure 7 illustrates the relationship between latency and throughput, two critical QoS metrics, for various load-balancing algorithms under different scenarios. The baseline algorithms exhibit more significant variability in balancing latency and throughput. In contrast, the enhanced algorithms, especially PPO, consistently achieve higher throughput with more stable latency. Distinct clusters for each scenario underscore the impact of varying load conditions on algorithm behavior. For instance, the Burst Load scenario generally results in higher latency and throughput than the Normal Load scenario, reflecting the increased system stress.

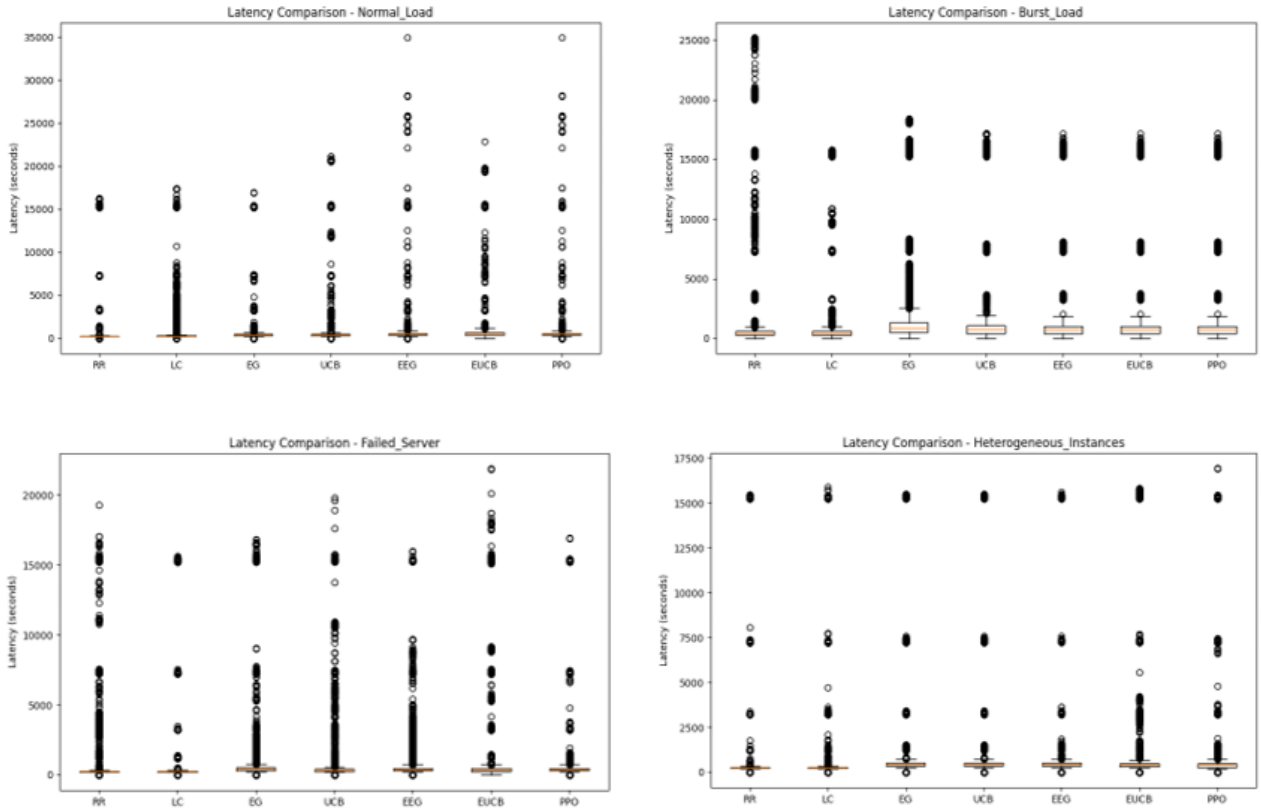

Figure 6: System Latency across all algorithms.

Table 2: The Percentage of successful requests across all scenarios.

ios / Algorithms (%)	*RR*	*LC*	*EG*	*UCB*	*EEG*	*EUCB*	*PPO*
Normal Load	89.5	92.0	91.1	92.2	91.6	91.9	97.1
Burst Load	86.9	86.2	93.0	87.6	90.3	92.5	94.7
Failed Server	84.4	88.5	92.1	91.3	83.9	91.8	93.6
Heterogeneous Instances	90.4	90.9	91.0	91.0	91.8	92.0	93.1

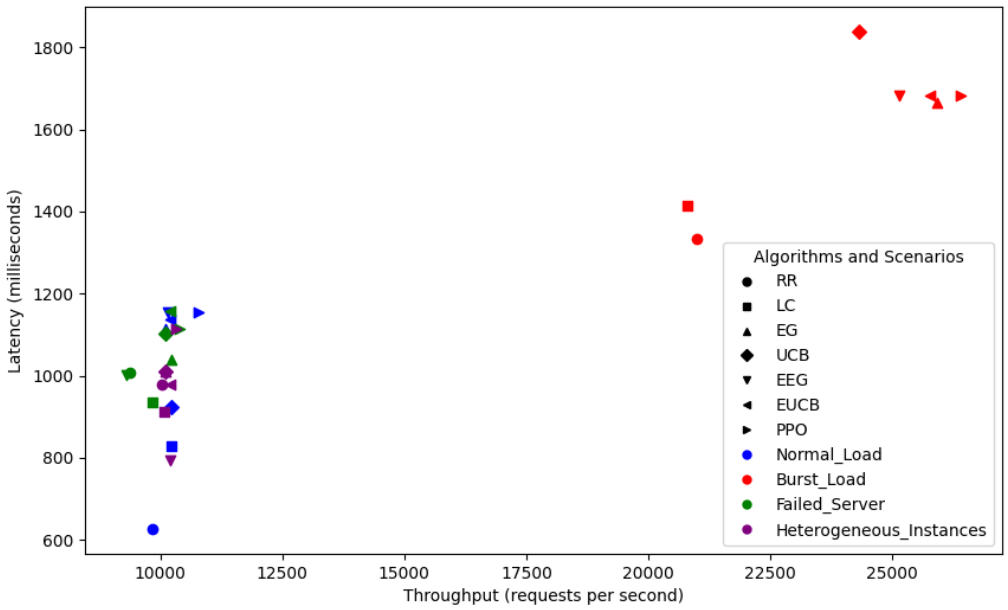

Figure 7: The Quality of Services across all scenarios.

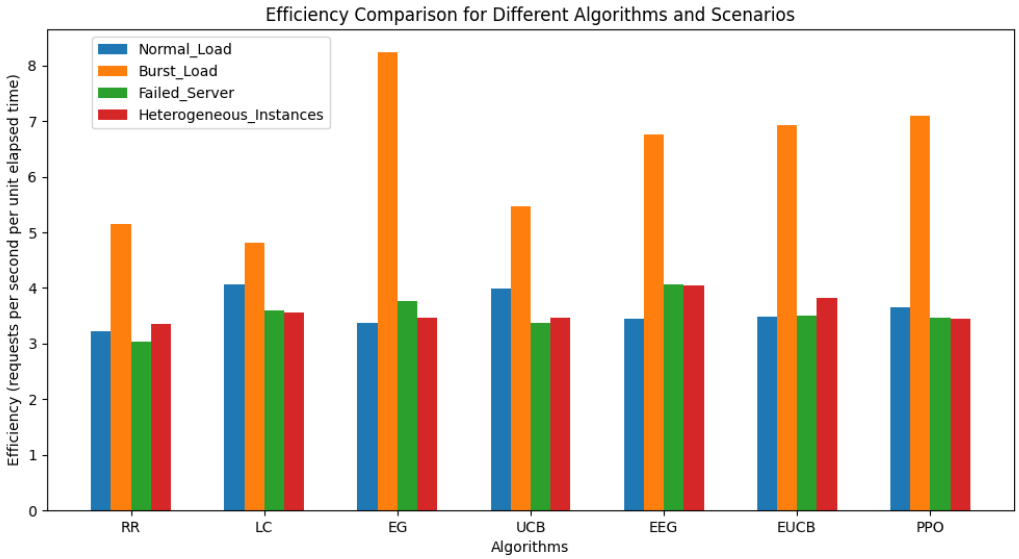

Figure 8: The Effeciency analysis of the proposed Pub/Sub System.

4.6 The efficiency of the Pub/Sub system

Figure 8 presents the efficiency of various load-balancing algorithms across four distinct scenarios. Enhanced algorithms (EUCB and PPO) consistently outperform the other algorithms, demonstrating their superior ability to maintain high throughput, especially during Burst Load. These results highlight the effectiveness of the enhanced algorithms in maximizing the system's capacity to process messages and ensure high delivery rates.

The enhanced algorithms, particularly PPO, utilize reinforcement learning to make dynamic load-balancing decisions based on real-time system feedback. This adaptability enables them to effectively handle fluctuations in traffic patterns and subscriber loads, outperforming static algorithms. Moreover, these algorithms actively route messages to available and healthy subscribers, resulting in lower failure rates and a more resilient system. Including CPU usage in the reward function of the enhanced algorithms further contributes to balanced resource allocation and improved system performance by detecting and mitigating potential subscriber bottlenecks.

While recent research is focused on distributed brokers, our single-broker optimization approach using RL demonstrates the potential for enhancing Pub/Sub system performance, complementing existing work, and opening avenues for broader system improvements.

5 Conclusions

This research explored the application of reinforcement learning (RL) for load balancing in publish/subscribe systems. We evaluated traditional algorithms (Round Robin, Least Connections) and developed enhanced versions of epsilon-greedy and UCB alongside a custom PPO-based approach. Our experimental results demonstrate that RL significantly outperforms traditional methods, particularly the PPO and EUCB algorithms. These enhanced algorithms showed marked improvements in makespan, achieving completion times up to 30% faster than conventional methods. Moreover, they exhibited notable gains in throughput, particularly in the Burst Load scenario, where EUCB and PPO showcased a 10-15% increase in throughput compared to baseline algorithms. This research also highlighted the enhanced algorithms' superior ability to maintain high message success rates, exceeding 90% in most scenarios, and their contribution to more stable and predictable latency, leading to improved QoS. Overall, this research underscores the potential of RL for adaptive and efficient load balancing in Pub/Sub systems, paving the way for more resilient and responsive distributed applications.

6 Future work

This research can be expanded upon in several promising ways. One area of focus is exploring alternative reward function formulations and systematically varying the weights assigned to different factors. Additionally, the investigation of more comprehensive state representations could enhance the algorithms' learning capabilities. Another potential direction is developing hybrid algorithms that combine the strengths of different RL methods or integrate RL with traditional load-balancing techniques. Furthermore, evaluating the proposed algorithms in more diverse and realistic scenarios would provide a more comprehensive understanding of their performance characteristics. Finally, deploying and evaluating the RL-based load-balancing system in a real-world production environment would provide valuable insights into its practical feasibility and effectiveness.

References

[1] A. Javahar, R. Ananth, K. K. Arun Ritthik, and R. Dharun, “Efficient load balancing for Micro Services based applications,” in *2023 International Conference on Computer Communication and Informatics (ICCCI)*, IEEE, Jan. 2023, pp. 1–5. https://doi.org/10.1109/iccci56745.2023.10128431

[2] G. Barlas, “Load balancing,” in *Multicore and GPU Programming*, Elsevier, 2023, pp. 887–941. doi: https://doi.org/10.1016/B978-0-12-814120-5.00022-6.

[3] D. I. Sukhoplyuev and A. N. Nazarov, “Analysis of Application-Level Load Balancing Algorithms,” in *2023 Systems of Signals Generating and Processing in the Field of on-Board Communications*, IEEE, Mar. 2023, pp. 1–4. doi: 10.1109/IEEECONF56737.2023.10092019.

[4] M. G. Spina, G. M. Marotta, S. Gualtieri, and F. De Rango, “Topic Load Balancing in a multi IoT Gateways Scenario under Publish/Subscribe Paradigm,” in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, Jan. 2022, pp. 521–522. doi: https://doi.org/10.1109/CCNC49033.2022.9700606.

[5] D. Man, W. Yang, and G. Tian, “Polymorphic Load Balancing Algorithm Based on Packet Classification,” in *Proceedings of the 2nd International Conference on Telecommunications and Communication Engineering*, New York, NY, USA: ACM, Nov. 2018, pp. 258–261. doi: https://doi.org/10.1145/3291842.3291911.

[6] S. Gilbert, U. Meir, A. Paz, and G. Schwartzman, “On the Complexity of Load Balancing in Dynamic Networks,” in *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, New York, NY, USA: ACM, Jul. 2021, pp. 254–264. doi: https://doi.org/10.1145/3409964.3461808.

[7] S. K. Mishra, B. Sahoo, and P. P. Parida, “Load balancing in cloud computing: A big picture,” Feb. 01, 2020, *King Saud bin Abdulaziz University*. doi: https://doi.org/10.1016/j.jksuci.2018.01.003.

[8] Panjwani, K., Pathan, S., Yadav, N., Lokhande, S. and Thakare, B., “Load Balancing, Optimal Routing and Scheduling in Hyper-Local.,” *International Journal of Computer Applications*, p. 975, Dec. 2015. https://doi.org/10.5120/ijca2015907393

[9] Erl, Ricardo Puttini Thomas, and Z. Mahmood, *Cloud Computing: Concepts, Technology & Architecture (The Pearson Service Technology Series from Thomas Erl) 1st Edition*, 1st ed. Pearson, 2013.

[10] A. Jyoti, M. Shrimali, S. Tiwari, and H. P. Singh, “Cloud computing using load balancing and service broker policy for IT service: a taxonomy and survey,” *J Ambient Intell Humaniz Comput*, vol. 11, no. 11, pp. 4785–4814, Nov. 2020, doi: https://doi.org/10.1007/s12652-020-01747-z.

[11] D. Wu, Li Jimmy, Ferini Amal, Xu Yi Tian, Jenkin Michael, Jang Seowoo, Liu Xue, and Dudek Gregory, “Reinforcement learning for communication load balancing: approaches and challenges,” *Front Comput Sci*, vol. 5, May 2023, doi: https://doi.org/10.3389/fcomp.2023.1156064.

[12] J. Xu, H. Guo, H.-W. Shen, M. Raj, S. W. Wurster, and T. Peterka, “Reinforcement Learning for Load-Balanced Parallel Particle Tracing,” *IEEE Trans Vis Comput Graph*, vol. 29, no. 6, pp. 3052–3066, Jun. 2023, doi: https://doi.org/10.1109/TVCG.2022.3148745.

[13] J. Wang, "A reinforcement learning-based network load balancing mechanism," in *Fifth International Conference on Computer Information Science and Artificial Intelligence (CISAI 2022)*, Y. Zhong, Ed., SPIE, Mar. 2023, p. 162. doi: https://doi.org/10.1117/12.2667915.

[14] M. Shahakar, S. Mahajan, and L. Patil, "Load Balancing in Distributed Cloud Computing: A Reinforcement Learning Algorithms in Heterogeneous Environment," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 11, no. 2, pp. 65–74, Mar. 2023, doi: https://doi.org/10.17762/ijritcc.v11i2.6130.

[15] R. Kawaguchi and M. Bandai, "A Distributed MQTT Broker System for Location-based IoT Applications," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, Jan. 2019, pp. 1–4. doi: https://doi.org/10.1109/ICCE.2019.8662069.

[16] A. Detti, L. Funari, and N. Blefari-Melazzi, "Sub-Linear Scalability of MQTT Clusters in Topic-Based Publish-Subscribe Applications," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1954–1968, Sep. 2020, doi: https://doi.org/10.1109/TNSM.2020.3003535.

[17] J. Hasenburg, F. Stanek, F. Tschorsch, and D. Bermbach, "Managing Latency and Excess Data Dissemination in Fog-Based Publish/Subscribe Systems," in *2020 IEEE International Conference on Fog Computing (ICFC)*, IEEE, Apr. 2020, pp. 9–16. doi: 10.1109/ICFC49376.2020.00010.

[18] B. Kruekaew and W. Kimpan, "Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm with Reinforcement Learning," *IEEE Access*, vol. 10, pp. 17803–17818, 2022, doi: https://doi.org/10.1109/ACCESS.2022.3149955.

[19] J. Chen, W. Yong, O. Jiangtao, F. Chengyuan, L. Xiaoye, L. Cenhuishan, and H. Xuefeng "ALBRL: Automatic Load-Balancing Architecture Based on Reinforcement Learning in Software-Defined Networking," *Wirel Commun Mob Comput*, vol. 2022, pp. 1–17, May 2022, doi: https://doi.org/10.1155/2022/3866143.

[20] E. Longo and A. E. C. Redondi, "Design and implementation of an advanced MQTT broker for distributed pub/sub scenarios," *Computer Networks*, vol. 224, p. 109601, Apr. 2023, doi: https://doi.org/10.1016/j.comnet.2023.109601.

[21] O. Houidi, Z. Djamal, P. Victor, A. Quang, P. Tran, H. Nicolas, L. Jeremie, M. Paolo, "Constrained Deep Reinforcement Learning for Smart Load Balancing," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, Jan. 2022, pp. 207–215. doi: https://doi.org/10.1109/CCNC49033.2022.9700657.

[22] P. Yang, L. Zhang, H. Liu, and G. Li, "Reducing idleness in financial cloud services via multi-objective evolutionary reinforcement learning based load balancer," *Science China Information Sciences*, vol. 67, no. 2, p. 120102, Feb. 2024, doi: https://doi.org/10.1007/s11432-023-3895-3.

[23] A. Daghistani, W. G. Aref, A. Ghafoor, and A. R. Mahmood, "SWARM: Adaptive Load Balancing in Distributed Streaming Systems for Big Spatial Data," *ACM Transactions on Spatial Algorithms and Systems*, vol. 7, no. 3, pp. 1–43, Sep. 2021, doi: https://doi.org/10.1145/3460013.

[24] S. V. Nethaji and M. Chidambaram, "Differential Grey Wolf Load-Balanced Stochastic Bellman Deep Reinforced Resource Allocation in Fog Environment," *Applied Computational Intelligence and Soft Computing*, vol. 2022, pp. 1–13, Aug. 2022, doi: https://doi.org/10.1155/2022/3183701.

[25] N. M. M. Muna Mohammed Jawad, "RHLB: Improved Routing Load Balancing Algorithm Based on Hybrid Policy," *Journal of University of Babylon for Engineering Sciences*, , vol. 27, no. 1, Feb. 2019. Doi: https://doi.org/10.29196/jubes.v27i1.2005

[26] H. A. J. Saja Dheyaa Khudhur, "DLSTM-MSF: Distributed LSTM Models for Multimedia Streaming Workload Forecasting Based on Kafka Environment," *Iraqi Journal of Computers, Communications, Control, and Systems Engineering*, vol. 24, no. 1, pp. 103–118, Mar. 2024. Doi: https://doi.org/10.33103/uot.ijccce.24.1.7

[27] E. K. H. Eman K Ibraheem, "Load Balancing Performance Optimization for LI-Fi/Wi-Fi HLR Access Points Using Particle Swarm Optimization and DL Algorithm," *International Journal of Intelligent Engineering & Systems*, vol. 15, no. 6, Nov. 2022. Doi: https://doi.org/10.22266/ijies2022.1231.34

[28] A. Mudheher, K. Ghalib, Safanah Mudheher, "Enhanced Performance of Consensus Wireless Sensor Controlled System via Particle Swarm Optimization Algorithm," *Journal of Engineering*, vol. 23, no. 9, Sep. 2017. Doi: https://doi.org/10.31026/j.eng.2017.09.05

[29] Yossra Ali, Nuha Ibrahim, sajjad jaber, "Task Scheduling in Cloud Computing Based on The Cuckoo Search Algorithm," *Iraqi Journal of Computer, Communication, Control and System Engineering*, vol. 22, no. 1, pp. 86–96, Mar. 2022, doihttps:/doi.org/10.33103/uot.ijccce.22.1.9.

[30] Hanan Al-asady. Ekhlas K. Hamza, "Indoor Localization System Using Wireless Sensor Network," *Iraqi Journal of Computers, Communications, Control, and Systems Engineering*, vol. 18, no. 1, 2018. Doi: https://doi.org/10.33103/uot.ijccce.18.1.3