

Article

Not peer-reviewed version

Towards an Optimized Distributed Message Queue System for the AIoT Edge Computing: A Reinforcement Learning Approach

[Zaipeng Xie](#)^{*}, [Cheng Ji](#), Lifeng Xu, Mingyao Xia, Hongli Cao

Posted Date: 12 May 2023

doi: 10.20944/preprints202305.0944.v1

Keywords: Artificial intelligence of thing; Message queue system; Reinforcement learning approach; System throughput




Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Article

Towards an Optimized Distributed Message Queue System for the AIoT Edge Computing: A Reinforcement Learning Approach

Zaipeng Xie ^{1,2,*} , Cheng Ji ², Lifeng Xu ², Mingyao Xia ² and Hongli Cao ³

¹ Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, Nanjing 211100, China

² Department of Computer Science and Technology, Hohai University, Nanjing 211100, China;

³ Key Laboratory of Underwater Acoustic Signal Processing of Ministry of Education, Southeast University, Nanjing 210096, China;

* Correspondence: zaipengxie@hhu.edu.cn

Abstract: The convergence of artificial intelligence and the Internet of Things (IoT) has made remarkable strides in the realm of industry. The IoT devices in this computing model gather data from various sources and send it to edge servers for real-time processing, which presents a challenge to the existing message queue systems in Artificial Intelligence of Things (AIoT) Edge computing. These systems lack the adaptability to respond to the current state of the system, such as changes in the number of devices, message size, and frequency, and optimize the message transmission mechanism accordingly. Hence, it is critical to devise an approach that can effectively decouple message processing and mitigate workload fluctuations in the AIoT computing environment. To this end, this study introduces a distributed message queue system that is specifically tailored to the AIoT edge environment, utilizing a reinforcement learning approach to optimize message queue performance. Empirical findings reveal that this pioneering method significantly improves system throughput while handling varying message scenarios.

Keywords: artificial intelligence of thing; message queue system; reinforcement learning approach; system throughput

1. Introduction

The advancements in artificial intelligence (AI) have led to the widespread deployment of AI applications, ranging from Industry 4.0 [1] to smart cities [2]. With the growing adoption of the Internet of Things (IoT) and edge computing, the IoT devices used in these applications are generating vast amounts of data that require real-time processing by servers. The combination of AI and IoT has given rise to Artificial Intelligence of Things (AIoT) systems [3], which have the potential to enhance IoT operations, improve human-machine interaction [4], and optimize data management [5]. However, traditional message queues for IoT systems are designed to provide a lightweight publish/subscribe messaging transport, which is insufficient for the massive message transmission required in AIoT systems. AIoT systems may also face message floods from time to time due to network jitter or failures, making it essential for an ideal message queue to smoothly deliver entire message events under uncontrollable environmental factors. Various message queue systems have been developed for real-time data processing frameworks, including RabbitMQ [6], RocketMQ [7], ActiveMQ [8], and Kafka [9], which are widely used distributed message queues. These traditional message queues are primarily designed for message communication in the cloud environment, and they tend to be challenging to tune in the AIoT computing environment. In the AIoT computing environment, the network status, including the number of AIoT devices, message size, and the message sending frequency, may vary unpredictably. The performance optimization of such message queues can be constrained due to limited hardware resources [10], making it imperative to devise methods that can enhance the performance of distributed message queues for AIoT systems.

Distributed message queues are essential to different data processing application scenarios, providing users with configurable parameters to customize their performance. For instance, Kafka and RocketMQ offer more than two hundred adjustable parameters [11], which usually require manual adaptation [12] to different message transmission scenarios. However, the sheer number of parameter combinations makes it practically impossible to search for an optimized solution within a limited time for the system's varying scenarios. As a result, users often resort to applying the default configuration or seeking assistance from experienced experts for distributed message queue parameter configuration [13]. Therefore, an algorithm is needed to optimize the performance of distributed message queues by adaptively selecting and tuning critical parameters that significantly impact the throughput in the AIoT Edge computing scenarios.

This paper introduces a novel optimization technique for distributed message queue systems in AIoT Edge Computing: DMSCO (DDPG-based Distributed Message System Configuration Optimization algorithm). The DMSCO algorithm includes four essential stages. Firstly, the optimization process acquires a parameter list to create an initial training sample dataset. Secondly, the parameter list is dimensionally reduced by the Principal Component Analysis (PCA)[14] method. Thirdly, the pre-processed parameter list will be screened by Lasso regression [15] to identify the parameter variables that have a particularly significant impact on the performance of the distributed messaging system in order to build an optimization model. Finally, the DMSCO method establishes a performance prediction model and optimizes the parameters utilizing a deep deterministic policy gradient-based algorithm to enhance the performance of the distributed message queue. To evaluate the effectiveness of the proposed optimization method, experiments were conducted by integrating it into an existing distributed message queue. In summary, our contributions are as follows:

- We investigate the potential factors that affect system performance in a high-demand, large-scale messaging scenario using an existing distributed message queue system. To address this issue, we propose a partition selection algorithm for distributed message queues that enhances system throughput and improves message delivery success rates with appropriate configurations. We conduct evaluations to demonstrate the necessity of optimizing configurations for improved transmission performance in highly concurrent messaging scenarios at a large scale.
- We propose a DDPG-based Distributed Message Queue Systems Configuration Optimization algorithm (DMSCO), which leverages a pre-processed parameter list as an action space to train a decision model. By constructing rewards based on the distributed message queue system's throughput and message transmission success rate, the DMSCO algorithm can effectively optimize messaging performance in various AIoT scenarios by adapting the optimal parameter configurations.
- We evaluated the proposed DMSCO algorithm under varying message sizes and transmission frequency cases to validate its performance efficacy for the distributed message queue system in different AIoT Edge computing scenarios. Our comparative analysis against methods utilizing genetic algorithms and random searching revealed that the proposed DMSCO algorithm offers an efficient solution to address the unique demands of larger-scale, high-concurrency AIoT Edge computing applications.

The remainder of this paper is structured as follows. Section 2 provides an overview of related works on distributed message queues and their optimization. Section 3 details the proposed Partition Selection Algorithm for the Distributed Message Queue. In Section 4, we describe the reinforcement learning algorithm that optimizes the performance of the message queue. Section 5 discusses the experiments and analyzes the proposed algorithms with regard to message transmission success rate and system throughput. Finally, Section 6 offers conclusions.

2. Related Work

The field of distributed message queues has garnered significant attention from both industry and academia. Various distributed message queues have been developed for message transmission scenarios, with RabbitMQ[6], RocketMQ[7], ActiveMQ[8], and Kafka[9] being the most commonly used. ActiveMQ has a centralized architecture and limited single-machine throughput performance, making it less popular nowadays. RabbitMQ, on the other hand, is an enterprise-level message queue based on the AMQP protocol, designed as a high-concurrency system with good performance and latency as low as microseconds. RocketMQ is an open-source distributed message queue that deviates from the JMS specification, leading to limited usage in the industry due to difficulties in system migration. Apache Kafka is an open-source distributed message queue that boasts high throughput and scalability, using a publish/subscribe model suitable for real-time and batch processing. However, Kafka struggles to adapt to the changing message transmission scenarios in the AIoT computing environment. It also faces issues such as the inability to guarantee message ordering between partitions and repeated consumption problems. With Kafka's widespread use, researchers have begun paying attention to improving its cluster development, operation, and maintenance costs. Kafka relies on Zookeeper for leader election and other coordination management, and its various versions require tuning when the system updates.

Fu et al. [16] conducted a comparison of the performances of several different message brokers. They found that Kafka systems, compared to RabbitMQ, can process messages with low latency and high throughput with an increased number of partitions. Wu et al. [9] corroborated this finding, demonstrating that Kafka outperforms RabbitMQ when processing a large volume of messages. However, the authors also noted that Kafka delivers messages with less reliability compared to RabbitMQ. Despite its limitations, Kafka has shown a better ability to handle applications that need to process many messages in a content-based Apache Flink platform. Nonetheless, developing a practical Apache Kafka's message partition selection algorithm remains an open problem, and adaptive performance optimization is challenging.

Bao et al. [17] summarized several common parameter configuration optimization methods for message queues, including model-based method [18], measurement-based method [19,20], search-based methods [21,22], and learning-based method [23]. The authors noted that model-based configuration relies on abstracting models from the software architectures, which can be inaccurate and may not adapt well to varying conditions. Most measurement-based methods require collecting program configuration status to identify performance bottlenecks, which may not lead to overall performance improvement. Search-based methods can take a relatively long time when performing the optimization task and may not perform well in scenarios with a large state space. Learning-based methods usually require a sparse prediction model by sampling the high-dimensional configuration space for optimization, but it is unclear whether these methods are suitable for optimizing distributed message queues in the AIoT computing environment, where the message scale frequently changes. Existing distributed message queues lack an appropriate way [24] to respond to the current network status, such as changes in the number of devices, message size, and sending frequency. Research on an optimization method for the distributed message queue performance in AIoT application scenarios is desired.

3. Distributed Message Queue System for AIoT Edge Computing

The primary focus of performance optimization for distributed message queues lies in the transmission of messages from end devices to centralized servers in cloud computing settings[25]. However, in AIoT edge computing environments, there is a lack of adequate evaluation and optimization for message transmission between end devices. These environments can involve varying numbers of end devices and message scales, with sensors used for simple monitoring generating small data packages, and scenarios like robot collaboration, smart cities, drones, and autonomous driving requiring high message transmission volumes across numerous end devices. Moreover, federated

learning with AIoT poses a significant challenge due to its generation of a considerable volume of messages, including model updates and parameters, between end devices and the central server. Furthermore, the distributed message queues must handle varying computational capabilities and network connectivity among end devices, which may lead to delays or timeouts during message transmission. Thus, to operate efficiently, the distributed message queue system must be designed with high scalability, availability, and fault tolerance to manage these scenarios.

3.1. Distributed message system for Large-scale Message Transmission Scenarios

Traditional point-to-point message transmission often encounters challenges such as network congestion and a high rate of message losses in large-scale message transmission scenarios. To address these issues, a range of message queue solutions have been developed. Popular options include RabbitMQ, ActiveMQ, RocketMQ, and Kafka, each designed as distributed systems with architectures suitable for specific scenarios. For example, a typical distributed message queue system using Kafka comprises the following components [26]:

- Administrator module: this module assumes responsibility for the management and upkeep of all information associated with Topics and Partitions.
- API module: this module assumes the crucial role of encoding, decoding, assembling, and facilitating interactions with data.
- Client module: this module carries the responsibility of retrieving Broker metadata from Zookeeper and acquiring essential information regarding the mapping of Topics and Partitions.
- Cluster module: this module encompasses a collection of classes and their corresponding descriptions for key components such as Broker, cluster, partition, and replica.
- Control module: This module assumes a crucial role in overseeing various tasks, including leader election, replica allocation, partition expansion, and replica expansion.

In AIoT edge computing, ensuring message ordering is of utmost importance when transmitting messages between various AIoT edges and data centers. For this purpose, we utilize a distributed message queue system based on Kafka, where data reading and writing take place at the partition level. While adopting a single partition storage strategy, such as Kafka, can uphold message ordering, it does come at the expense of reduced throughput and a lack of load-balancing capabilities. Conversely, employing a multi-partition strategy holds the potential to enhance the message queue's throughput. However, it is important to note that when multiple partitions are utilized, Kafka only guarantees the ordered consumption of messages within each partition, leaving the ordering between partitions unassured. Hence, it is necessary to address the challenge of message disorder among partitions when employing a multi-partition strategy.

Our proposed distributed message queue incorporates a combination of two default partition strategies to optimize its functionality. The first strategy involves assigning each message a key and storing it in the corresponding partition. This ensures that messages are organized based on their keys. The second strategy utilizes a round-robin approach, sequentially allocating partitions and caching messages in a progressive manner, from the first partition to the last. However, the default partition strategy employed in Kafka presents a trade-off. While one strategy guarantees message order, it can potentially lead to congestion within specific partitions. On the other hand, the other strategy aims to balance the load across partitions but does not ensure strict message order. To address these challenges and strike a balance, we propose a customized partition selection approach. This approach involves checking if any partitions are associated with the received message's key. If a corresponding partition exists, the message is directed to that specific partition. If no such association is found and there are available partitions (where the number of available partitions is greater than zero, denoted as $m > 0$), the message is sent to a free partition determined using a hash function applied to the message key. The remainder of the result, obtained when dividing by $m + 1$, determines the target partition. In cases where m equals zero, the remainder is computed by dividing by $n + 1$, where n represents the total

number of partitions. By employing this customized partition selecting approach, we aim to overcome the limitations of the default strategies in terms of congestion and message order. This enables our distributed message queue to efficiently manage message distribution while ensuring load balance and, when possible, maintaining the desired message order.

Algorithm 1 outlines the proposed partitioning method to address the unique characteristics of messages in the AIoT edge computing environment. By implementing Kafka’s Partitioner interface and utilizing the terminal ID as the key, our method introduces an improved partition selection approach that prioritizes partitioning by the keys. Its primary objective is to prevent message disorder caused by polling strategies.

Algorithm 1: Partition selection algorithm

Input: Number of partitions n , number of available free partitions m , and matrix data package key x

Output: The partition which should be sent currently

- 1 Get matrix data packet;
- 2 Search if there is a partition tag currently containing x ;
- 3 **if** (No tag contains x) && ($m > 0$) **then**
- 4 Send the matrix data packet to the partition value obtained by taking the hash value of the key modulo m and adding 1;
- 5 Add the x tag to the partition;
- 6 $m -= 1$;
- 7 **end**
- 8 **else if** The Partition tag currently contains x **then**
- 9 Send the matrix data packet to the partition;
- 10 **end**
- 11 **else if** (No partition tag contains x) && ($m = 0$) **then**
- 12 Send the matrix data packet to the Partition obtained by taking the hash value of the key and computing the remainder when divided by n , then adding 1;
- 13 **end**
- 14 **foreach** partition in partitions **do**
- 15 **if** Current partition is empty **then**
- 16 Clear all key value tags;
- 17 $m += 1$;
- 18 **end**
- 19 **end**

By utilizing this partition selection algorithm, a large number of messages sent to different terminals within the same time frame can be reasonably distributed among the available partitions, effectively achieving load balancing. Simultaneously, it ensures the order of messages destined for the same AIoT destination, mitigating the detrimental effects of message disorder on the distributed message system.

3.2. Performance modeling in Large-Scale Scenarios

We evaluate the performance of distributed message systems in large-scale scenarios by simulating an environment with a handful of identical edge devices. Each device comprises four processes responsible for maintaining a queue and a hash table. Process 1 generates matrices of varying sizes, following a normal distribution, as long as the local queue is not full, thus emulating real-time message generation. Process 2 consumes messages from other devices and generates verification data packets. Process 3 assesses the successful delivery of validation messages by verifying the hash table and the received verification data packet. Lastly, process 4 parses network packets to determine whether they correspond to matrices or verification data packets. To evaluate the performance of the distributed message systems, we conduct performance tests and comparisons under different scenarios involving

varying numbers of devices and running times. These tests provide insights into our system's efficiency, reliability, and scalability in accommodating large-scale message transmission environments.

The experiment involves the classification of groups based on the number of devices, each of which transmits messages at varying rates. For the group transmitting ten messages per second, the sizes of messages conform to a normal distribution with a mean of $\mu = 128\text{KB}$ and a standard deviation of $\sigma = 10$. In contrast, the group transmitting a single message per second exhibits a normal distribution with a mean of $\mu = 1024\text{KB}$ or 2048KB and a standard deviation of $\sigma = 200$. The efficacy of message transmission is measured in terms of success rate and throughput for a duration of 60 minutes, and the results are presented in Figure 1.

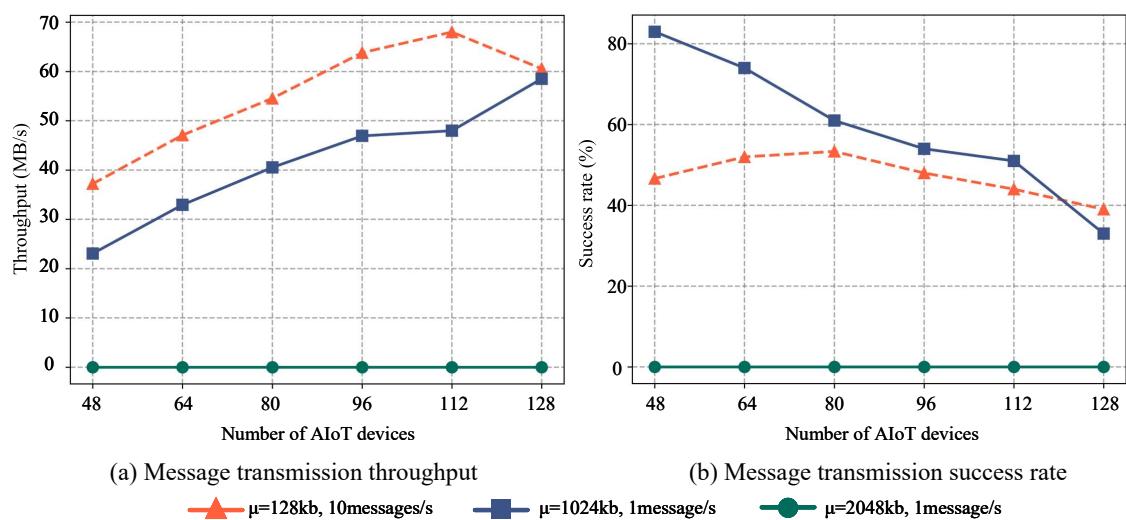


Figure 1. The performance of message transmission with different quantities and characteristics varies as the number of AIoT edge devices increases. (a) the throughput variation, and (b) the success rate variation.

Experimental results indicate that the success rate of the distributed message system declines significantly with an increase in the number of devices when small messages are transmitted frequently. The primary cause of this is the inadequate sending and receiving buffers, which result in considerable losses during message transmission to and from the message queue to the consumer. However, the system's overall throughput increases with an increase in the number of devices, peaking at 68.5MB/s. In the case of low-frequency one-time messages with a mean size of 1024KB, the average success rate of message transmission is around 50%, despite some devices showing message errors. This phenomenon occurs because the default maximum value for single messages is 976KB. Given the normal distribution of message sizes, where the mean is 1024KB, and the standard deviation is 200, roughly half of the messages exceed the default maximum value. Moreover, when the mean size of single messages is 2048KB, the system's success rate and throughput plummet to zero, causing all devices to display message send error prompts. This outcome arises because the single mean size of 2048KB and standard deviation of 200 completely exceed the default maximum value for single messages.

The results suggest that while the distributed message queue proposed in this paper is well-suited for large-scale message transmission scenarios among devices in edge environments, its performance can be suboptimal under default configurations, particularly when handling high-concurrency message transmission scenarios in AIoT edge computing environments. As a result, traditional distributed message queues may still lack the capacity to adjust and optimize message transmission mechanisms based on the current state, including the number of devices, message size, and frequency. To address

this issue, we propose a solution named DMSCO that employs reinforcement learning to optimize the parameter configuration for distributed message queues.

4. Reinforcement Learning-based Method for Optimized AIoT Message Queue System

The AIoT edge environment requires a data processing capability with real-time latency. However, utilizing the traditional point-to-point message transmission mode in this environment can result in message loss, network congestion, and inadequate real-time processing capability. While existing distributed message systems rely on message queues for message delivery, they lack the adaptability to optimize message delivery mechanisms in response to changes in device numbers, message sizes, and frequencies [11].

We first explore the connection between distributed message queues and their key parameter configurations in the AIoT edge computing environment. Lasso regression[27] is utilized to select key parameters and their weights, resulting in a performance prediction model. Additionally, we construct a parameter optimization model based on reinforcement learning and employ the Deep Deterministic Policy Gradient (DDPG)[28] method to optimize the parameters. This enables the current parameter configuration to maximize the system throughput under the current message scale, achieving adaptive optimization of system performance. The specific optimization process is depicted in Figure 2.

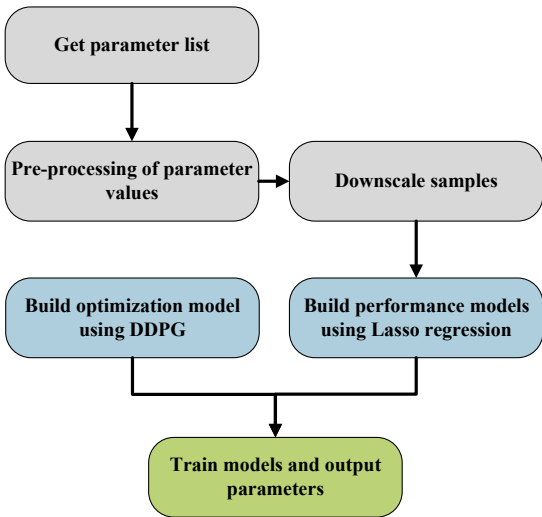


Figure 2. Performance optimization process diagram

4.1. Parameter Screening

In the AIoT Edge Computing environment, the distributed message system provides asynchronous, peak clipping, and decoupling capabilities, ensures sequential consumption under certain conditions, and offers redundant backup functions. The parameter configuration of the distributed message queue in this study can be modified through the use of the configuration file. The parameters that may impact the throughput performance of the distributed message queue are included in the Broker and Producer configurations. After analyzing the parameter description and technical arguments, twenty-two parameters are selected that are likely to have a significant impact on the throughput of the distributed message queue among the hundreds of parameters available.

To generate samples for the selected parameters that may affect the performance of distributed message queues, the types of values for the twenty-two parameters are classified as discrete or continuous parameters, among which discrete parameters include categorical and discrete numerical parameters. For example, "cType" is a categorical variable with values such as "uncompressed", "producer", and "gzip". In contrast, "bThreads" and "rTMs" are numerical variables, where "bThreads"

is a discrete numerical variable and "rTMs" is a continuous numerical variable. Given the environment of the distributed message system, three values are selected for each parameter: K0 (below default), K1 (default), and K2 (above default). The selection of these values aims to reflect the characteristics of the parameter and ensure a comprehensive evaluation of the parameter's performance under different settings.

Upon selecting the parameter values, the original data undergoes preprocessing. The parameter types of the distributed message queue are classified into numerical and categorical variables. Consequently, it is crucial to preprocess categorical variables by converting them into numerical variables. In this study, one-hot encoding [29] is utilized to represent categorical variables, resulting in the conversion of both discrete and continuous parameter types into numerical types. One-hot encoding employs an N -bit state register to encode N categorical values, with each value having only one corresponding register bit. For discrete or categorical parameters with only two possible values, the default value can be used. The 22 parameter values K0, K1, K2 are then combined to generate a training sample dataset St , which contains 322 samples. As a result, it is necessary to reduce the initial training sample dataset. After the preprocessing stage, a representative final training sample set Ft is selected for further analysis.

There are two primary approaches to reducing data dimensionality, including projection and manifold learning methodologies. Among these techniques, Principal Component Analysis (PCA) [14], t-distributed stochastic neighbor embedding (t-SNE), and multidimensional scaling (MDS) are among the most prominent. In this study, PCA is selected as the preferred method for dimensionality reduction due to its projection-based nature, suitability for reducing the dimensionality of the training sample dataset, and faster computational performance compared to alternative techniques. Therefore, PCA is employed to reduce the dimensionality of the initial training sample set St , with the detailed procedure outlined in Algorithm 2. The resulting sample dataset after dimensionality reduction is denoted as Y , which contains a total of 100 final training samples set Ft .

Algorithm 2: Dimensionality reduction method based on PCA for the initial training sample set

Input: Original samples $X = \{X_1, X_2, X_3, \dots, X_{22}\}$, where each row represents values of each parameter in the training samples, and each column represents the data of the i -th sample

Output: The final sample data set Y

- 1 Get the training sample data set matrix X ;
 - 2 Decenter X so that each parameter value is subtracted from the average of the three sample values of that parameter;
 - 3 Calculate the covariance: $XX^T / 21$;
 - 4 Solving the eigenvectors and eigenvalues of covariance $XX^T / 21$ by eigenvalue decomposition;
 - 5 Select the 100 eigenvalues with the largest eigenvalues, and form the eigenvector matrix Q with the 100 eigenvectors corresponding to them as column vectors;
 - 6 Calculate $Y = XQ$, then the reduced-dimensional sample data set is Y , which contains a total of 100 final training sample sets;
 - 7 Return Y ;
-

4.2. Lasso regression-based performance modeling

The current set of parameters for the distributed message system was based on the documentation and expert advice from other distributed message queues. However, given the particularity of the AIoT edge computing environment in this study, some parameters may have a negligible impact on the performance of the distributed message queue compared to others. Therefore, a regression algorithm can be employed to further screen the parameters based on the final training sample set Ft that has been selected.

Regression algorithms are supervised methods that utilize labeled samples to create a mathematical model. To develop a final performance prediction model relevant to the distributed message system and throughput, regression algorithms can be used to calculate parameter weights and conduct final screening. Our study elects a comprehensive set of 100 training samples to provide a large dataset and minimize noise interference. To avoid overfitting, regularization is also implemented. Our method utilizes Lasso regression [15] to eliminate parameters with minimal impact on performance during parameter screening, as illustrated in Algorithm 3.

Algorithm 3: Key parameter screening method based on Lasso regression

Input: Pre-processed samples $Y = \{Y_1, Y_2, Y_3, \dots, Y_{22}\}$

Output: Key parameters and their weightings

- 1 Get the pre-processed sample data set Y ;
 - 2 Configuring and running 100 sets of final parameter samples in a distributed message system;
 - 3 Test and obtain the throughput and status. Remove sample data that runs abnormally;
 - 4 Construction of a set of training data pairs consisting of parameter configurations and throughputs;
 - 5 Normalize the parameters to a normal distribution of $N(0, 1)$;
 - 6 Set the performance model $f_a(x) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_{22} x_{22}$ and the loss function $J(\alpha) = \frac{1}{200} \sum_{i=1}^{100} (f_a(x^{(i)}) - y^{(i)})^2 + \lambda \cdot \sum_{j=1}^{22} |\alpha_j|$;
 - 7 Update $[\alpha_1, \alpha_2, \dots, \alpha_{22}]$ using gradient descent;
 - 8 Remove the parameters with small or zero absolute weights;
 - 9 Return parameters and their weights;
-

The significant configuration parameters in large-scale distributed message systems highlighted in this section are determined through the aforementioned steps, along with their corresponding weight magnitudes indicating their impact on performance, as shown in Table 1.

Table 1. Key parameters and their weights obtained by Lasso regression screening.

Parameter name	weight
bThreads	15.03
cType	70.35
nNThreads	23.74
nIThreads	25.16
mMBytes	60.35
qM-Requests	124.32
nRFetchers	-24.59
sRBBBytes	70.42
sSBBBytes	120.35
sRMBBytes	54.36
acks	43.58
bMemory	73.66
bSize	-170.95
lMs	34.32

To attain performance optimization of the distributed message system in the AIoT edge network, a preliminary model for performance optimization is desired, based on the final selection of 14 key parameters, as demonstrated in the equations below.

$$f(x) = \alpha_0 + 15.03x_1 + 70.35x_2 + 23.74x_3 + 25.16x_4 + 60.35x_5 + 124.32x_6 - 24.59x_7 + 70.42x_8 + 120.35x_9 + 54.36x_{10} + 43.58x_{11} + 73.66x_{12} - 170.95x_{13} + 34.32x_{14} \quad (1)$$

where $x_1 \sim x_{14}$ are the key parameters, and α_0 is a constant with different sizes in different distributed message system scenarios.

4.3. Parameter optimization method based on deep deterministic policy gradient algorithm

DDPG is a powerful deep reinforcement learning algorithm [28] that effectively handles continuous action spaces in high-dimensional environments by combining Q-learning and Actor-Critic approaches. The algorithm employs an Actor-network to generate actions and a Critic network to estimate the Q-value function, and updates the policy by computing the gradient of the Q-value function with respect to the policy parameters and using it to update the Actor-network. DDPG also utilizes experience replay and target networks to ensure stability during training and prevent overfitting. In contrast, DQN is a popular reinforcement learning method that uses deep learning to handle high-dimensional state spaces by approximating the Q-value function using a neural network and treating rewards as labels for training the network. However, DQN is designed for handling discrete action spaces and may not be suitable for continuous action spaces. In continuous action spaces, approximating the Q-value function directly using a neural network can be difficult, limiting the algorithm's ability to handle high-dimensional continuous actions. Thus, in this study, we use the DDPG algorithm illustrated in Figure 3 to effectively address the challenges posed by continuous action spaces in high-dimensional environments.

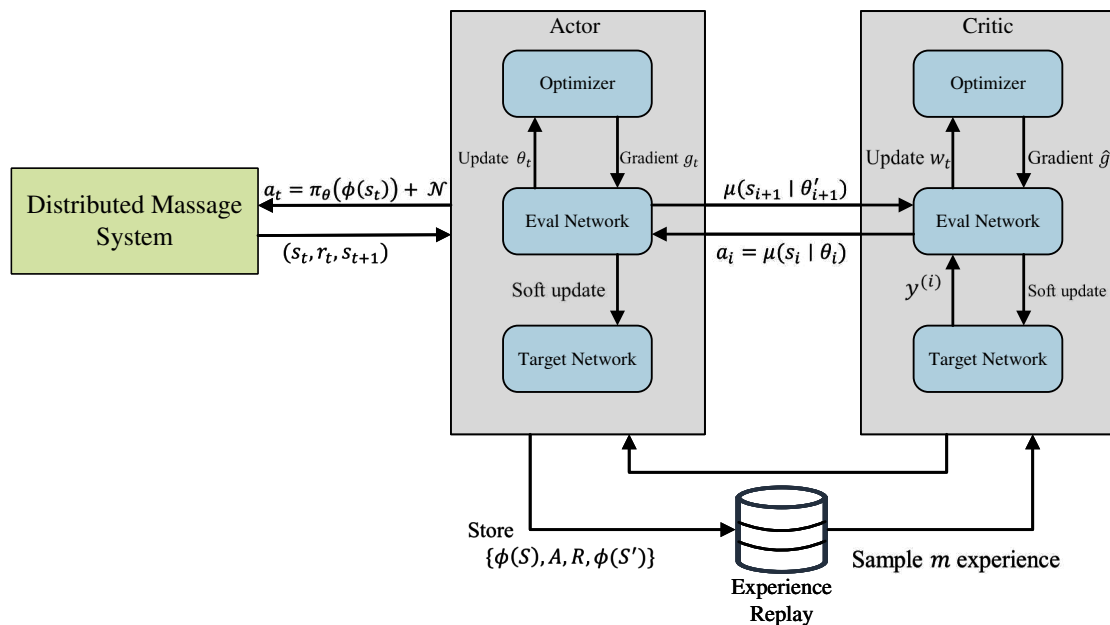


Figure 3. DDPG algorithm for distributed message system configuration optimization

The DDPG algorithm utilizes two neural networks [28], the Actor network and the Critic network, to effectively handle continuous action spaces. To prevent complications during the update process caused by constant changes to the target, the DDPG algorithm employs separate current and target networks. As a result, there are four neural networks: the current Critic network, the current Actor network, the Critic target network, and the Actor target network. The current Actor network takes the current state, reward, and next state as input and outputs the action to be executed in the next step. On the other hand, the current Critic network evaluates the current Q-value. It takes the action and state output by the Actor network as input and outputs the Q-value.

To update the two target networks based on the two current networks, the DDPG algorithm periodically updates the network parameters of the Actor target network and the Critic target network

in a soft manner based on the parameters of the current Actor network and current Critic network, respectively. The Actor target network selects the optimal next action A' based on the next state S' sampled from the experience replay pool. Meanwhile, the Critic target network calculates the target Q-value. The detailed algorithmic description is shown in Algorithm 4.

Algorithm 4: DDPG based distributed message system configuration optimization (DMSCO)

Input: Init all networks for actor and critic, init experience buffer and stochastic noise \mathcal{N}
Output: Actor optimal network parameters θ , Critic optimal network parameters w

```

1 State = env.reset();
2 for t from 1 to T do
3   Get the current action  $a_t$  based on the current state  $s_t$  through the Actor network;
4   Execute  $a_t$  to get the  $s_{t+1}$  and reward  $r_t$ ;
5   Store experience  $\{s_t, a_t, r_t, s_{t+1}\}$  to buffer  $\mathcal{D}$ ;
6   for i from 1 to m do
7     Sample a experience  $\{s_i, a_i, r_i, s_{i+1}\}$ ;
8     Calculate the target Q value  $u_j = R_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1} | \theta') | w')$ ;
9     Minimize the MSE loss  $L(w) = \frac{1}{m} \sum_{j=1}^m (u_j - Q(s_i, a_j | w))^2$ ;
10    Calculate gradient for Actor:  $\nabla_{\theta} J = \frac{1}{m} \sum_{i=1}^m \nabla_a Q(s, a | \theta) \cdot \nabla_w \mu(s | w)$ ;
11    Update model with gradient ascent;
12    if Update cycle then
13       $w' \leftarrow \tau \cdot w + (1 - \tau)w'$ ;
14       $\theta' \leftarrow \tau \cdot \theta + (1 - \tau)\theta'$ ;
15 Return  $\theta, w$ ;
```

We commence by initializing four deep neural networks: the evaluation critic network, evaluation actor network, target critic network, and target actor network. These networks utilize learnable parameters w , θ , w' , and θ' to approximate the Q-value function and policy function. At each time step, the current state s_t and action a_t are input to the current critic and actor networks, yielding the Q-value and action. Subsequently, several hyperparameters are specified, including the soft update coefficient τ , discount factor γ , experience replay buffer \mathcal{D} , batch size m for batch gradient descent, target Q-network update frequency C , and maximum number of iterations T . Additionally, a random noise function \mathcal{N} is initialized to enhance learning coverage and introduce stochasticity during training. Finally, the first state in the state sequence s_0 is designated as the initial state, from which the learning algorithm proceeds. We can train the DDPG algorithm through these initialization steps to obtain an optimized policy for the given task.

We optimize the distributed message system by initially mapping the range of the final key parameter set Kp (which has been converted to a unified numerical key parameter) to the state space. The current state S is then defined as the throughput of the system under the current key parameter configuration. The action A is specified to increase, keep constant, or decrease each key parameter. The reward R is computed based on the ratio of the current state (i.e., the current throughput) to the previous state throughput after performing the action A . If the ratio is greater than 1, the reward is set to 1. If the ratio is less than 1, the reward is set to -1 . Otherwise, the reward is set to 0.

Moreover, the key parameters of the distributed message system are initially set to default values to form the initial state sequence S . The feature vector of this state sequence is then computed to represent the current state's parameter configuration. The current state S is subsequently fed into the actor network, which utilizes policy gradient calculations as described in Equation 2 to determine the appropriate action that the network outputs.

$$A = \pi_{\theta}(\phi(S)) + \mathcal{N} \quad (2)$$

To update the parameter values of the distributed message system, action A is executed, resulting in a new state. The reward R is then computed based on the new state's throughput compared to the previous state S . The new state is observed, and its feature vector is obtained. The quadruple $\{S, A, R, S'\}$ is then stored in the experience replay pool \mathcal{D} . If the reward R equals 1, the current state is updated to S' ; otherwise, the current state remains unchanged. Through this process, the distributed message system can optimize performance by learning from past experiences.

Our proposed DMSCO algorithm randomly samples 32 experience data from the buffer \mathcal{D} , denoted as $\{\phi(S_i), A_i, R_i, \phi(S'_i)\}, i \in [1 \sim 32]$, the target Q value can be calculated as in Eq. (3).

$$u_j = R_j = \gamma \cdot Q'(\phi(S'_i), \pi'_\theta(S'_i) | w') \quad (3)$$

Then, the algorithm minimizes the MSE loss in Eq. (4) to update the critic evaluation network.

$$L(w) = \frac{1}{32} \sum_{i=1}^3 2(u_j - Q'(\phi(S_i), A_i | w))^2 \quad (4)$$

Finally, DMSCO calculates the loss function in Eq. (5). And update the actor network with gradient ascent in Eq. (6)

$$J(\theta) = -\frac{1}{32} \sum_{i=1}^{32} Q(\phi(S_i), A_i | \theta) \quad (5)$$

$$\nabla_\theta J = \frac{1}{32} \nabla_A Q(S, A | \theta)|_{S=Q(\phi(S_i), A=\mu(Q(\phi(S_i))))} \nabla_w \mu(S | w)|_{S=Q(\phi(S_i))} \quad (6)$$

The iteration continues until the maximum number of iterations $T = 1000$ is reached, at which point the training process terminates. The final output is the configuration of parameters in the optimal action A , which represents the optimal configuration of the distributed message system for the present message transmission scenario.

5. Experiments

To assess the efficacy of the proposed reinforcement learning-based configuration optimization method for distributed message queues, we establish a simulated edge AIoT environment comprising multiple edge devices.

5.1. Comparison set up

The Evaluation session consists of three parts. Firstly, we verify the actual performance of the proposed method through multiple iterations of training optimization under different high concurrency scenarios. Secondly, we compare the performance of our proposed DDPG method to both genetic algorithm and random searching in large-scale AIoT scenarios featuring high frequency and small messages. Lastly, we verify the optimization performance of the genetic algorithm and random searching in practical use for low-frequency and large messages.

5.2. Analysis on performance and results

To assess the efficacy of the reinforcement learning-based performance optimization method for large-scale message systems, we conducted experiments using 128 devices, with the following specifications: (1) Each device transmitted 10 messages per second, with the size of each message following a normal distribution with a mean of $\mu=128\text{KB}$ and a standard deviation of $\sigma=10$. (2) Each device transmitted 1 message per second, with the size of each message following a normal distribution with a mean of $\mu=1280\text{KB}$ and a standard deviation of $\sigma=100$.

As illustrated in Figure 4, the optimized distributed message system demonstrates a consistent increase in message transmission success rate across various high-concurrency scenarios with each iteration. In the initial setup, each AIoT terminal sends 10 messages per second, with the size of

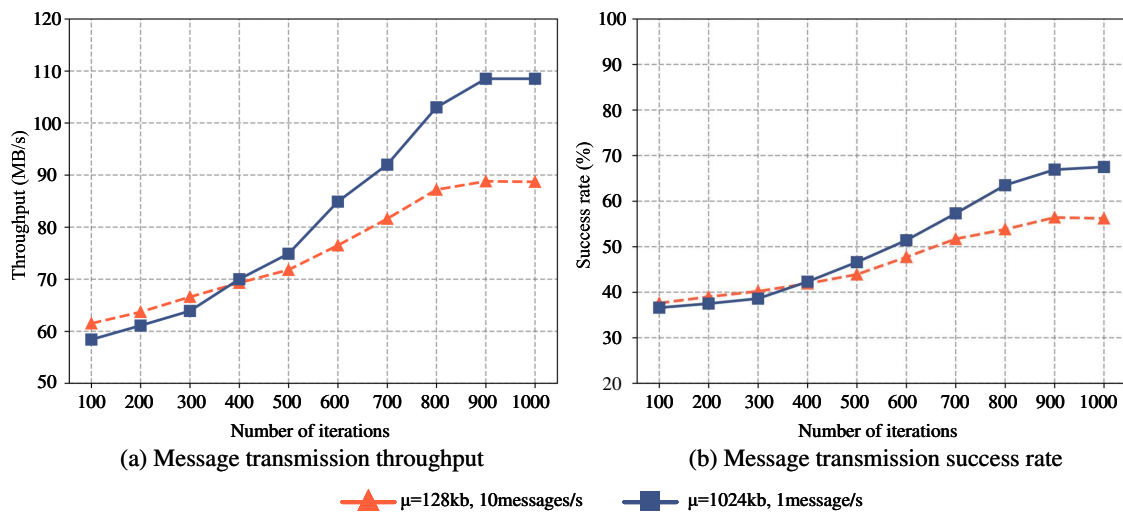


Figure 4. The performance of message transmission with different quantities and characteristics after optimized with DDPG. (a) illustrates the throughput variation, and (b) illustrates the success rate variation.

a single message following a normal distribution with a mean of $\mu=128\text{KB}$ in the high-frequency small message scenario. The message transmission success rate is marginally higher than that of the low-frequency large message scenario, where each terminal sent 1message per second, with the size of a single message following a normal distribution with a mean of $\mu=1280\text{KB}$. The two success rates intersect when the number of iterations reached approximately 400, indicating that they are nearly equal. However, when the number of iterations exceeded 400, a substantial improvement in the message transmission success rate of the low-frequency large message scenario can be observed compared to that of the high-frequency small message scenario.

To validate the efficacy of the proposed optimization method, we perform comparative experiments with traditional genetic algorithms and random search-based methods. For the traditional genetic algorithm, we set the initial population to 100, the crossover probability to 0.5, and the mutation probability to 0.01. In the random search-based method, we randomly search for values for each dimension of the key parameters.

Figure 5 (a) presents a comparison of the message transmission throughput among edge terminals using a message system based on a distributed message queue, where each terminal sends 10 messages per second. The size of each individual message follows a normal distribution with a mean of $\mu=128\text{KB}$ and a standard deviation of $\sigma=10$. We compare the optimization methods based on reinforcement learning, traditional genetic algorithms, and random searching at different iteration counts.

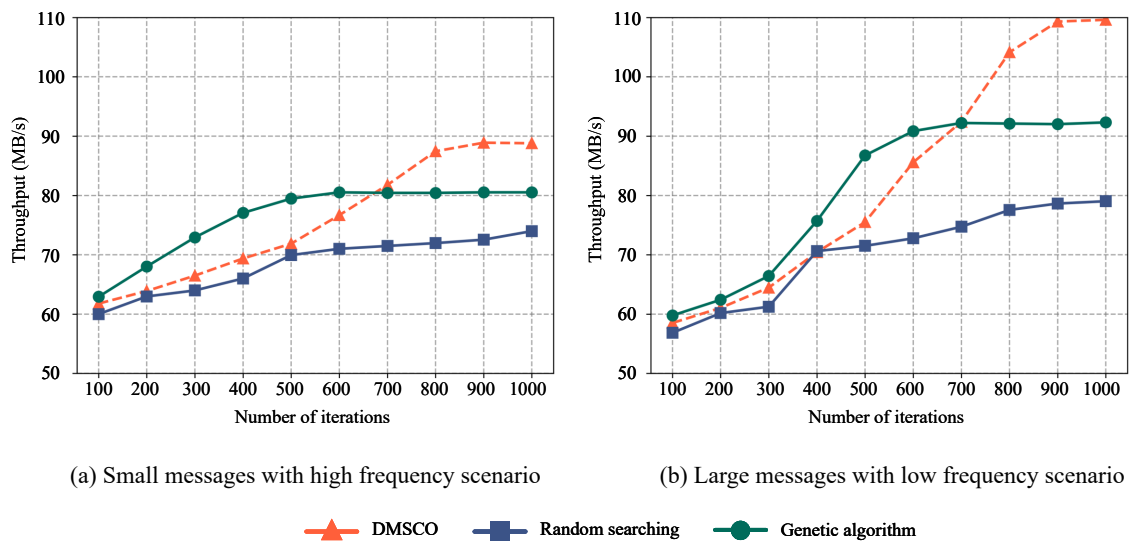


Figure 5. Optimization effects comparison of the proposed RL-based method, genetic algorithms and random searching on distributed message systems in different concurrent scenarios. (a) Small message with high frequency (b) Large message with low frequency

Figure 5 (b) depicts the throughput variation of message transmission in a distributed message system, where communication is based on the Kafka message queue. Each terminal sends one message per second, with the size of each message following a normal distribution with a mean of $\mu=1280\text{KB}$ and a standard deviation of $\sigma=100\text{KB}$. The results indicate that all three optimization methods positively impact the throughput of the distributed messaging queue in the scenario with infrequent large messages. Over the course of 1000 iterations, both the optimization method based on reinforcement learning and the one based on genetic algorithm show a higher overall improvement in throughput than the method based on a random search. Moreover, after 900 iterations, the proposed optimization method surpasses the genetic algorithm optimization method and achieves the best result of 108.5MB/s.

Experimental data indicate that the proposed DMSCO method performs exceptionally well in different message scenarios. In high-frequency, small-message scenarios, our method can enhance the message transmission success rate and performance by approximately 54% compared to the default configuration after 1000 iterations. In low-frequency, large-message scenarios, this method outperforms itself, with the message transmission success rate and performance improving by approximately 85% compared to the default configuration after 1000 iterations. Additionally, a comparative analysis of the experiments revealed that both the reinforcement learning-based and genetic algorithm-based optimization methods exhibited better convergence and performance optimization effects compared to the random search-based optimization method. Moreover, the optimization method based on reinforcement learning demonstrated better performance in improving throughput. This is because in the genetic algorithm optimization process, premature convergence can occur, where a super-individual appears in the population, leading to the optimization result converging too early to a local optimal solution.

6. Conclusions

This study aims to tackle the challenge of performance and adaptability in distributed message queue systems in AIoT Edge computing environments. Our studies have revealed that the system's throughput and success rate can decline significantly at high concurrency levels, highlighting the need for an optimization method. To address this issue, we propose a distributed message queue system based on Kafka and a reinforcement learning-based optimization method for large-scale distributed

message queue systems. Our experimental results demonstrate that the proposed method outperforms both genetic algorithm-based and random searching methods in different messaging scenarios. In summary, this paper provides an effective solution for optimizing the message transmission mechanism, which can significantly enhance the performance and adaptability of distributed message queue systems in Edge computing and AIoT environments.

Author Contributions: Conceptualization, Z.X.; methodology, Z.X. and M.X.; software, M.X. and L.X.; validation, C.J., L.X. and M.X.; formal analysis, M.X.; investigation, C.J., L.X. and M.X.; resources, H.C.; data curation, C.J., L.X. and M.X.; writing—original draft preparation, Z.X. and M.X.; writing—review and editing, Z.X., C.J., and M.X.; visualization, C.J.; supervision, Z.X.; project administration, Z.X.; funding acquisition, Z.X. and H.C.. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by The Belt and Road Special Foundation of the State Key Laboratory of Hydrology-Water Resources and Hydraulic Engineering under Grant number 2021490811, and the National Key R&D Program of China Grant number 2016YFC0402710.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Peres, R.S.; Jia, X.; Lee, J.; Sun, K.; Colombo, A.W.; Barata, J. Industrial artificial intelligence in industry 4.0-systematic review, challenges and outlook. *IEEE Access* **2020**, *8*, 220121–220139.
- Ullah, Z.; Al-Turjman, F.; Mostarda, L.; Gagliardi, R. Applications of artificial intelligence and machine learning in smart cities. *Computer Communications* **2020**, *154*, 313–323.
- Li, H.; Ota, K.; Dong, M. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Network* **2018**, *32*, 96–101.
- de Freitas, M.P.; Piai, V.A.; Farias, R.H.; Fernandes, A.M.R.; de Moraes Rossetto, A.G.; Leithardt, V.R.Q. Artificial Intelligence of Things Applied to Assistive Technology: A Systematic Literature Review. *Sensors* **2022**, *22*. doi:10.3390/s22218531.
- Bourechak, A.; Zedadra, O.; Kouahla, M.N.; Guerrieri, A.; Seridi, H.; Fortino, G. At the Confluence of Artificial Intelligence and Edge Computing in IoT-Based Applications: A Review and New Perspectives. *Sensors* **2023**, *23*. doi:10.3390/s23031639.
- Ionescu, V.M. The analysis of the performance of RabbitMQ and ActiveMQ. 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER). IEEE, 2015, pp. 132–137.
- Yue, M.; Ruiyang, Y.; Jianwei, S.; Kaifeng, Y. A MQTT protocol message push server based on RocketMQ. 2017 10th International Conference on Intelligent Computation Technology and Automation (ICICTA). IEEE, 2017, pp. 295–298.
- Snyder, B.; Bosanac, D.; Davies, R. Introduction to apache activemq. *Active MQ in action* **2017**, pp. 6–16.
- Wu, H.; Shang, Z.; Wolter, K. Performance Prediction for the Apache Kafka Messaging System. 21st IEEE International Conference on High Performance Computing and Communications, 2019, pp. 154–161.
- Jo, H.C.; Jin, H.W.; Kim, J. Self-adaptive end-to-end resource management for real-time monitoring in cyber-physical systems. *Computer Networks* **2023**, *225*, 109669. doi:https://doi.org/10.1016/j.comnet.2023.109669.
- Stewart, J.C.; Davis, G.A.; Igoche, D.A. AI, IoT, and AIoT: Definitions and impacts on the artificial intelligence curriculum. *Issues in Information Systems* **2020**, *21*.
- Le Noac'H, P.; Costan, A.; Bougé, L. A performance evaluation of Apache Kafka in support of big data streaming applications. 2017 IEEE International Conference on Big Data (Big Data). IEEE, 2017, pp. 4803–4806.
- Nguyen, C.N.; Hwang, S.; Kim, J.S. Making a case for the on-demand multiple distributed message queue system in a Hadoop cluster. *Cluster Computing* **2017**, *20*, 2095–2106.
- Dong, W.; Woźniak, M.; Wu, J.; Li, W.; Bai, Z. Denoising aggregation of graph neural networks by using principal component analysis. *IEEE Transactions on Industrial Informatics* **2022**, *19*, 2385–2394.

15. Sarker, I.H. Machine learning: Algorithms, real-world applications and research directions. *SN computer science* **2021**, *2*, 160.
16. Fu, G.; Zhang, Y.; Yu, G. A fair comparison of message queuing systems. *IEEE Access* **2020**, *9*, 421–432.
17. Bao, L.; Liu, X.; Xu, Z.; Fang, B. Autoconfig: Automatic configuration tuning for distributed message systems. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 29–40.
18. Shakya, S.; others. IoT based F-RAN architecture using cloud and edge detection system. *Journal of ISMAC* **2021**, *3*, 31–39.
19. Brünink, M.; Rosenblum, D.S. Mining performance specifications. *ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE*. ACM, 2016, pp. 39–49.
20. Grochla, K.; Nowak, M.; Pecka, P.; others. Influence of Message-Oriented Middleware on Performance of Network Management System: A Modelling Study. *Multimedia and Network Information Systems - Proceedings of the 10th International Conference MISSI*, 2016, Vol. 506, *Advances in Intelligent Systems and Computing*, pp. 379–393.
21. Henard, C.; Papadakis, M.; Harman, M.; Le Traon, Y. Combining multi-objective search and constraint solving for configuring large software product lines. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 2015, Vol. 1, pp. 517–528.
22. He, H.; Jia, Z.; Li, S.; Yu, Y.; Zhou, C.; Liao, Q.; Wang, J.; Liao, X. Multi-Intention-Aware Configuration Selection for Performance Tuning. *44th International Conference on Software Engineering; Association for Computing Machinery: New York, NY, USA, 2022; ICSE '22*, p. 1431–1442. doi:10.1145/3510003.3510094.
23. Sarkar, A.; Guo, J.; Siegmund, N.; Apel, S.; Czarnecki, K. Cost-efficient sampling for performance prediction of configurable systems (t). *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 342–352.
24. Tao, Z.; Xia, Q.; Hao, Z.; Li, C.; Ma, L.; Yi, S.; Li, Q. A survey of virtual machine management in edge computing. *Proceedings of the IEEE* **2019**, *107*, 1482–1499.
25. Gou, F.; Wu, J. Message transmission strategy based on recurrent neural network and attention mechanism in IoT system. *Journal of Circuits, Systems and Computers* **2022**, *31*, 2250126.
26. Fu, G.; Zhang, Y.; Yu, G. A Fair Comparison of Message Queuing Systems. *IEEE Access* **2021**, *9*, 421–432.
27. Wang, Y.; Xiang, Z.J.; Ramadge, P.J. Lasso screening with a small regularization parameter. *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*. IEEE, 2013, pp. 3342–3346.
28. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; others. Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR*, 2016.
29. Johnson, J.M.; Khoshgoftaar, T.M. Encoding high-dimensional procedure codes for healthcare fraud detection. *SN Computer Science* **2022**, *3*, 362.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.