Hindawi
International Journal of Intelligent Systems
Volume 2023, Article ID 8835111, 17 pages
https://doi.org/10.1155/2023/8835111

Research Article

Workload-Aware Performance Tuning for Multimodel Databases Based on Deep Reinforcement Learning

Jun Sun,[1] Feng Ye,[1] Nadia Nedjah,[2] Ming Zhang,[3] and Dong Xu[4]

[1]*School of Computer and Information, Hohai University, Nanjing 211100, China*
[2]*State University of Rio de Janeiro, Rio de Janeiro, Brazil*
[3]*Water Resources Department of Jiangsu Province, Nanjing 210025, China*
[4]*College of Water Conservancy and Hydropower Engineering, Hohai University, Nanjing 210098, China*

Correspondence should be addressed to Feng Ye; yefeng1022@hhu.edu.cn

Received 15 December 2022; Revised 18 April 2023; Accepted 21 August 2023; Published 5 September 2023

Academic Editor: Yu-an Tan

Currently, multimodel databases are widely used in modern applications, but the default configuration often fails to achieve the best performance. How to efficiently manage and tune the performance of multimodel databases is still a problem. Therefore, in this study, we present a configuration parameter tuning tool MMDTune+ for ArangoDB. First, the selection of configuration parameters is based on the random forest algorithm for feature selection. Second, a workload-aware mechanism is based on k-means++ and the Pearson correlation coefficient to detect workload changes and match the empirical knowledge of historically similar workloads. Finally, the ArangoDB configuration parameters are optimized based on the improved TD3 algorithm. The experimental results show that MMDTune+ can recommend higher-quality configuration parameters for ArangoDB compared to OtterTune and CDBTune in different scenarios.

1. Introduction

With the rapid development of the Internet of Things technology and network applications, the scale of data is growing explosively, and the types of data are becoming richer [1, 2]. For example, applications such as social commerce and smart water conservancy usually include structured relational data and unstructured graph data. Traditional relational databases have difficulty meeting the needs of storage and querying diverse data structures. The emergence of multimodel database [3] (MMDB) provides a new solution to effectively address the shortcomings of traditional database. As a new trend in the field of database management systems [4–6], multimodel database can store data of various structural forms in a single engine, without the need to deploy different databases for data of various structures. Multimodel database is also considered to be the next generation of data management system combining flexibility, scalability, and consistency [7, 8].

Currently, multimodel databases are widely used in modern applications [9, 10], but their default configuration often cannot achieve the best performance. Multimodel databases have the problem of configuration parameter optimization, which generally needs to be adjusted according to the actual workload and application configuration [11]. Configuration parameter tuning is always a key challenge and is the subject of significant research in the database field. Tuning is usually performed by a database administrator (DBA) with extensive tuning experience. However, the method of tuning by DBAs has some limitations. First, tuning configuration parameters is an NP-hard problem [7]. There are hundreds of parameters in a database system, and there are connections between parameters, which makes it difficult for DBAs to complete the tuning of these configuration parameters. Second, the scheme for configuring parameters cannot be reused for database systems deployed in different environments (such as local hosts, clouds, or memory) [12–14]. It is difficult for DBAs to achieve high database performance by efficient parameter

tuning under changing scenarios. Finally, DBAs are usually good at tuning the systems they are familiar with, but it is difficult to tune unfamiliar systems. The research on the configuration parameter tuning of multimodel database, as a new kind of database, is less than that of traditional databases, and the tuning experience applied to traditional databases cannot be fully applied to multimodel database.

In our previous work [15], we proposed MMDTune, a parameter-tuning method for multimodel databases based on deep reinforcement learning. MMDTune has the problems of long training time and low computational efficiency, which cannot be dynamically adjusted in the course of practical application.

To solve the above problems, we improved MMDTune by introducing a workload-aware mechanism. A configuration parameter tuning tool MMDTune+ (multimodel database tune plus) for ArangoDB [16] is proposed, which includes three modules: a configuration parameter selection module, a workload-aware mechanism, and a tuning algorithm. Among them, the configuration parameter selection module uses the random forest algorithm for feature selection, and the configuration parameters with high correlation to the current tuning indicators are extracted. Random forest algorithm can effectively handle high-dimensional, noisy, and correlated data while avoiding overfitting and providing a measure of variable importance. The workload-aware mechanism is based on k-means++ and Pearson correlation coefficients to detect changes in workload and match empirical knowledge of historically similar workloads. The advantages of using k-means++ and Pearson correlation coefficients in matching similar workloads lie in their ability to effectively cluster and measure the similarity of workload characteristics, resulting in improved accuracy and efficiency in workload matching [17]. Finally, ArangoDB configuration parameter tuning is implemented based on the improved TD3 algorithm [18]. TD3 can handle high-dimensional parameter spaces, optimize performance over long training times, and achieve great performance on a variety of benchmark tasks [18]. At the same time, we optimize TD3 to make it converge faster. Considering that a benchmarking tool is needed in the tuning experiment to generate indicators that can simulate the workload in the real environment and measure the performance of the reaction system [19, 20], we design and implement a benchmarking tool MMDBench (multimodel database benchmarking), which includes a workload generator and a metric collector.

The contributions of our work are summarized as follows: (1) we propose an automatic configuration parameter tuning tool MMDTune+ based on a workload-aware mechanism and deep reinforcement learning for the multimodel database ArangoDB. (2) We develop a database benchmarking tool to provide comprehensive performance testing of our proposed architecture. (3) Based on the above performance benchmark tool, we verify our proposed method under different scenarios, loads, query, and insert modes. Compared with the existing automatic database tuning methods OtterTune [21] and CDBTune [22], our proposed method has the tuning effect and provides more advantages in resource consumption.

The subsequent sections are organized as follows: In Section 2, we introduce some related works about database tuning. In Section 3, we present the proposed MMDTune+ and MMDBench. In Section 4, we mainly describe our experimental setup and experimental results. In Section 5, we draw some conclusions and point out some future directions for future work.

2. Related Work

2.1. Workload-Aware Mechanism. The workload is an important criterion for database performance tuning [23], which requires that automatic tuning is able to identify workload changes. Therefore, the first step to realizing automatic tuning of configuration parameters is to accurately classify workload. Workload is varied, and different times and scenarios may cause changes in workload. At present, the related methods for workload-aware mechanisms mainly include two methods: supervised learning-based methods and unsupervised learning-based methods.

Supervised learning-based workload-aware methods have a limitation in that a large amount of labeled training data is required. Zewdu et al. [24] chose two algorithms to realize the awareness of database workload, including hierarchical clustering and classification regression trees. Their experiments on TPC benchmark query and transaction type workload verified that this method could effectively predict the category of workload. Elmaffar and Martin [25] proposed the prediction framework of psychic skeptic prediction (PSP) to realize the self-optimization of DBMS, in which workload identification is performed by using a decision tree classification algorithm in two different workload sets, TPC-H + TPC-C and TPC-W. The workload is divided into three types: OLTP, DSS, and hybrid, and the advantages of PSP classification are verified.

Labeled workload datasets are rare in practice, so most database workload-aware methods use unsupervised learning. Literature [26] only uses SQL query structures to carry out similar matching; that is, the author carries out vectored mapping for different SQL query structures and then uses BetaCV, DunnIndex, and other methods to standardize these structural data pairs, which improves the accuracy of Aligon, Aouiche, and other algorithms based on clustering. The effectiveness of the standardized method is verified. Takahashi [27] proposed a perception method based on the k-means clustering algorithm and clustering effectiveness index, which took into account various data specifications such as heavy load and light load, and divided all sensor data into multiple clusters. In research studies [21, 28], the factor analysis (FA) method and k-means method are used to reduce the dimension of the internal state features of the database to improve the execution efficiency, and then, the Euclidean distance is calculated according to these features to match similar workloads.

Generally, in system tuning tasks, it is necessary to rely on tuning the rich experience and professional knowledge of expert databases to realize the optimal performance of the

system. A database self-tuning system can also use this form. According to the historical tuning experience data in the search for a similar workload of empirical knowledge, the tuning system can draw lessons from the experience knowledge, which can improve the efficiency of optimization. Therefore, methods for sensing workload changes have been studied and used, so that the tuning system can learn from historical experience knowledge to tune after the perceived workload changes, to efficiently recommend parameters for the system.

2.2. Configuration Parameter Tuning. Most of the previous work on automatic database tuning has focused on optimizing the physical design of the database [29], such as selecting indicators [30, 31], partitioning schemes [32], or materialized views [33]. At present, the methods of database configuration parameter tuning can be divided into two representative studies according to whether the learning-based form is used: configuration parameter tuning based on rules and learning methods [34, 35].

The rule-based method is the selection of database management system knobs according to a predefined set of rules or heuristics, and the knobs are the configuration parameters of the database system. In most cases, this approach is designed for a specific database and only for a specific set of configuration parameters. IBM DB2 [36] releases a self-tuning memory manager to provide adaptive tuning of the database memory heap and cumulative database memory allocation. This technology combines control theory, runtime simulation modeling, cost-benefit analysis, and operating system resource analysis to provide memory-tuning technology in the form of heuristics. The authors in [7] propose a recursive tuning method BestConfig, which divides the high-dimensional parameter space into subspaces and uses the restricted derivation principle to search for the optimal configuration from the given configuration resources.

Deep learning has been successfully applied to solve computationally intensive learning tasks in many fields [37–41]. Zheng et al. [42] propose a self-tuning method using deep neural networks, and the authors suggest using statistical methods to identify key system parameters and neural networks to match the configuration of specific workloads. Xiong et al. [43] propose a multiobjective tuning framework MQTuner, which not only considered the performance indicators (throughput) concerned in most studies but also extended the latency indicators. The overall structure uses an artificial neural network (ANN) to learn the mapping relationship between configuration parameters and performance, input the predicted performance and configuration into the genetic algorithm at the same time, and then use the genetic algorithm to search for the globally optimal solution. IBTune [44] is proposed for a large-scale cloud database cache tuning framework. The author designs a two-layer deep neural network, according to the characteristics of the measured instance to predict the upper bound of the request and response time. The size of the target buffer pool can be adjusted only when the predicted response time is within the safety limit. However, there are still some limitations to the above methods. It is difficult for them to achieve optimal performance under limited training samples, and they are prone to overfitting [45–47].

Reinforcement learning, as a research hotspot of machine learning methods, has been used in some studies [22, 28, 48] to optimize the configuration parameters of database systems [49]. Zhang et al. [22] designed an end-to-end cloud database automatic tuning system (CDBTune) based on the deep reinforcement learning algorithm DDPG (deep deterministic policy gradient). The system uses DDPG to learn the mapping between database state features and high-dimensional configuration features. A two-state algorithm double-state deep deterministic policy gradient (DS-DDPG) built by QTune [48] combines neural networks and deep reinforcement learning methods. It utilizes the structure and internal characteristics of the database query and DS-DDPG input to perform database tuning simultaneously, taking into account the rich features of SQL query, and the model can use the query feature to predict changes in the value system's internal states. However, the system needs to collect a large number of real environments generated to train the neural network, which is a time-consuming process. The authors in [28] extend GPR, deep neural networks (DNN), DDPG, and improved DDPG+ for the configuration tuning of a real database production and application scenario of an international bank. Due to the overestimation problem of the DDPG algorithm, the errors in the training process are constantly accumulated, which may have a negative impact on the results.

In summary, previous studies have been able to optimize the performance of databases to some extent. However, they have some limitations. First, tuning with traditional machine learning methods and deep learning methods relies on a large number of high-quality training samples, which are often the experience data accumulated by DBAs. However, for multimodel databases, DBAs are relatively short of experience in this aspect. Second, there are often hundreds of parameters in the database, which are interrelated with each other. The simple regression method is far from sufficient for achieving the optimal goal. Finally, as a new trend in the database field, the research on configuration parameter tuning of multimodel database is less than that of traditional databases, and the previous tuning experience cannot be directly transferred to multimodel database. The configuration parameter tuning method based on depth DDPG can solve the first two problems. In the absence of high-quality empirical data, it can reduce the difficulty of data acquisition through empirical trial-and-error methods and achieve better performance in high-dimensional space configurations, but the estimation problem still exists.

3. Framework of MMDTune+

To realize automatic tuning of multimodel database, a tuning tool MMDTune+ for multimodel databases is proposed. Figure 1 shows the overall process of parameter tuning, which consists of four parts: the configuration parameter selection module, the workload-aware mechanism, the

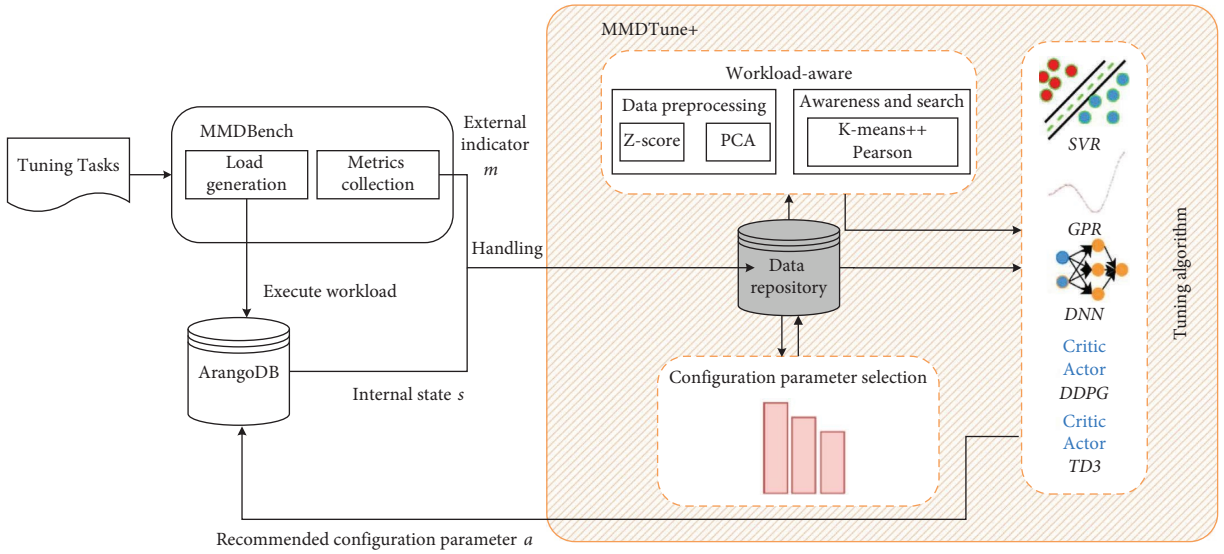

Figure 1: The overall process of parameter tuning.

tuning algorithm, and the benchmarking tool. Among them, a feature analysis model is proposed based on the random forest algorithm for database configuration parameter selection. The core of the workload-aware mechanism is to detect workload changes based on k-means++ and the Pearson correlation coefficient, calculate the similarity between the current workload and the historical workload database, match the historical workload, and then provide the tuning experience of similar workloads to the tuning algorithm. The proposed algorithm is mainly based on the improved TD3 algorithm to provide automatic tuning of database.

3.1. Configuration Parameter Selection Module. A multimodel database management system provides hundreds of configurable parameters to meet various requirements, which greatly increases the difficulty of tuning database configuration parameters. However, only a few configuration parameters provided by the database may have a significant impact on the performance. Tuning unnecessary configuration parameters is a waste of resources and inefficient behavior. Therefore, we propose a feature analysis model based on the RF algorithm. Random forest algorithm can be used for database tuning feature selection, which involves selecting the most important features in a database to improve its performance. Random forest can handle large datasets with a large number of features, which is common in databases. Meanwhile, random forest can handle noisy data, which can be common in databases with incomplete or inconsistent data. Also, it can identify complex relationships between features, which are important for identifying key features in databases that may be related to performance issues. Moreover, it can provide a ranking of feature importance, which can help identify the most important features that need to be tuned to improve database performance. Overall, random forest is a powerful and flexible algorithm for database tuning feature selection that can provide valuable insights into the most important features that need to be tuned to improve database performance. So, random forest algorithm is utilized to select the tuning parameters of the multimodel database. With configuration parameters and tuning performance indicators as inputs, a large number of configuration parameters are screened at the initial stage, and the parameters are sorted according to the weight of characteristic variables. Finally, the sorted configuration parameter labels are output. Compared with other feature selection methods, the RF algorithm has the advantages of fast training speed, good robustness, and high accuracy.

The main applications of the RF algorithm are classification and regression. It is composed of multiple decision trees, and each node in the tree is a condition about a certain input feature. In terms of regression application, the final result is the mean value of each decision tree. The commonly used objective functions for fitting RF regression tasks include the mean square error (MSE) and mean absolute error (MAE). MSE is used as the objective function in this paper. The specific process is shown in Algorithm 1.

After the model is trained, a feature correlation parameter *feature_importance* is generated. The module can sort configuration parameters according to *feature_importance*. The impact of configuration parameters on performance is positively correlated with this value. The sorted configuration parameters are saved in the database. One can set the number of configuration parameters to select the configuration parameters that are highly relevant to the tuning indicator.

3.2. Workload-Aware Mechanism Algorithm. Workloads in a database environment are constantly changing over time, and high-quality configuration parameters for a previous workload applied to the current target workload can degrade system performance. Therefore, the workload changes of the system should be monitored before configuration parameter

(1) *Input*: configuration parameter sets $X = \{a_1, a_2, \ldots, a_n\}$
(2) performance indicator $Y = \{m\}$; number of trees n; feature labels labels
(3) *Output*: sorted configuration parameter labels $F = \{a_1', a_2', \ldots, a_n'\}$
(4) RFModel.construct()
(5) RFModel.train()
(6) let coefs = RFModel.feature_importance
(7) let agg_feature_coefs = zip(coefs, feature_labels)
(8) sorted(agg_feature_coefs)

Algorithm 1: Configuration parameter selection module.

tuning so that the configuration parameter tuning system can sense the workload changes and recommend high-quality configurations for different workloads. At the same time, there are certain similarities in the configuration parameter adjustment strategies between similar workloads. If the tuning algorithm can be fine-tuned based on the experience of similar workloads, the tuning efficiency can be improved to a certain extent.

The workload data of this research are not labeled data, so the algorithm based on supervised learning cannot be selected. To solve the above problems, we propose a workload-aware mechanism solution for ArangoDB. The k-means ++ clustering algorithm is used to divide the historical workload into various categories. This algorithm is a popular clustering algorithm utilized in machine learning for data clustering. In the domain of multimodel database workload classification, the use of k-means++ offers several advantages. First, it is highly scalable and efficient, enabling its application to large volumes of data. Second, it is less sensitive to initialization conditions, thereby enhancing its stability and performance. Finally, it can lead to more accurate clustering, especially when the data are distributed in a nonuniform manner. These advantages position k-means++ as a viable option for multimodel database load classification tasks. If the new workload is different from the previous workload after classification, then the current workload has changed. The system can automatically use Pearson correlation coefficients to match the most similar workloads from the history library belonging to the new type and then tune for the new workload.

As a clustering algorithm [50], k-means++ can classify samples well without labeled training data, and it has the characteristics of fast computation speed and high robustness. Specifically, k-means++ is a variant of k-means. The k-means algorithm itself is a distance-based clustering algorithm. Clustering is based on the similarity between data; that is, similar things are divided into a class, but it uses random rules to determine the initial clustering center. A poor initial cluster center setting can have a very bad effect on the results. In view of this, k-means++, on the basis of k-means, improves the original k-means method of initializing cluster centers. The basic idea is that the initial cluster centers should be as far away from each other as possible.

First, the internal state data vector is converted to Z-score standardization, so that these variables are on the same order of magnitude, and then, the important information is screened. This is processed by the principal component analysis (PCA) method. Then, k-means++ is used to detect the change in workload, and the Pearson correlation coefficient is used to calculate the similarity between the new workload and the historical workload under this workload type. The Pearson correlation coefficient is a measure of the linear correlation between two variables X and Y. It is a widely used statistical measure to evaluate the strength and direction of the relationship between two continuous variables. The value of the Pearson correlation coefficient ranges from −1 to +1, where a coefficient of +1 indicates a perfect positive correlation, 0 indicates no correlation, and −1 indicates a perfect negative correlation. A positive correlation means that as one variable increases, the other variable also tends to increase, while a negative correlation implies that as one variable increases, the other variable tends to decrease. The workload-aware mechanism module selects the previous samples with high similarity as the initial model parameters of the TD3 algorithm, which are then fine-tuned to improve the tuning efficiency and performance improvement rate. The whole process does not require human intervention, and the tuning system will automatically complete the subsequent tuning work after detecting the workload change.

3.3. Tuning Method. MMDTune+ implements ArangoDB configuration parameter tuning based on the deep reinforcement learning algorithm TD3. To make the TD3 algorithm more suitable for database configuration parameter tuning, the network structure of TD3 is improved. In addition, tuning algorithms SVR, GPR, DNN, and DDPG are extended in MMDTune+ for tuning ArangoDB parameters and evaluating the effectiveness of the TD3 algorithm.

TD3 is a deep reinforcement learning algorithm that has shown promising results in multimodel database tuning. This algorithm is particularly useful in settings where the objective function is nondifferentiable and noisy. One of the key advantages of TD3 is its ability to handle continuous action spaces, which is particularly relevant in the context of database tuning, where tuning parameters typically vary continuously. TD3 uses a deterministic policy, which enables it to effectively optimize continuous action spaces. Additionally, TD3 uses a twin network structure, which consists of two separate critic networks, to reduce overestimation of the value function. This twin network structure

enables TD3 to learn more accurate value estimates and improve the stability of the learning process. Another advantage of TD3 is its ability to handle the exploration-exploitation tradeoff effectively. In multimodel database tuning, it is important to balance the exploration of new parameter configurations with the exploitation of already learned information. TD3 uses a replay buffer to store past experiences and a target policy network to estimate the value of future states. By using a combination of both these techniques, TD3 can effectively explore the search space while also exploiting previously learned information. Moreover, TD3 also has the advantage of being a model-free algorithm, meaning that it does not require a priori knowledge of the underlying system dynamics. This makes it particularly useful in the context of database tuning, where the relationship between tuning parameters and system performance can be complex and difficult to model accurately. By not requiring a model, TD3 can learn directly from the data and adapt to changes in the environment without needing to update a model. These features make TD3 a powerful tool for optimizing the performance of multimodel databases.

TD3 consists of two kinds of networks: an Actor and two Critics, each of which has a target network corresponding to it. The Actor network takes the internal state indicator s obtained after the environment executes the workload as the input and outputs the actions in the size range 0 to 1. Then, MMDTune+ obtains the configuration parameters according to the action mapping to overwrite the original configuration of the environment. As shown in Figure 2, the target network μ' is the same as the current network structure μ.

In the network, the activation functions LeakyReLU and Tanh are used to capture the nonlinear relationships between variables. The average function value after Tanh is close to zero, which is conducive to the learning of neurons in the next layer. LeakyReLU can alleviate the situation of gradient disappearance and gradient explosion and speed up the convergence of the model to a certain extent. Finally, a dropout layer is added to the network to prevent overfitting of the model and increase exploration of the configuration space.

TD3 has two Critic networks Q_1 and Q_2, and corresponding two target Critic networks Q_1' and Q_2'. Their network structure is the same, and they share the experience playback pool. The design of the network structure is similar to that of the Actor strategy network, except that the weight update frequency is different from that of the Actor. Critic networks are used to evaluate the value of state action and guide Actor behavior according to value feedback.

The Critic network structure is shown in Figure 3. Both Critic networks take the internal state s of ArangoDB and action a output of Actor network as input. After deeply connecting the network to learn the relationship between state and action, the final output is the evaluation Q value of state s and action a.

The reward function is crucial for reinforcement learning because it determines the feedback between the agent and the environment. The goal of the agent is to maximize the total revenue it receives, and the reward function must make the agent to achieve the goal while maximizing the revenue.

First, the performance time change rate with respect to the initial performance and the time is calculated respectively. The computations are performed according to the following equations:

$$\Delta_{t,0} = \begin{cases} \dfrac{y_{t,i} - y_{0,i}}{y_{0,i}}, & m_i \text{ is throughput,} \\ \dfrac{y_{0,i} - y_{t,i}}{y_{0,i}}, & m_i \text{ is resource utilization,} \end{cases} \tag{1}$$

$$\Delta_{t-1,t} = \begin{cases} \dfrac{y_{t,i} - y_{t-1,i}}{y_{t-1,i}}, & m_i \text{ is throughput,} \\ \dfrac{y_{t-1,i} - y_{t,i}}{y_{t-1,i}}, & m_i \text{ is resource utilization.} \end{cases} \tag{2}$$

According to equations (1) and (2), the reward function is shown in equation (3), where $\Delta_{t-1,t}$ indicates the difference between the current performance and the ArangoDB performance under the default configuration, and $\Delta_{t,0}$ indicates the difference between the current performance and the historical optimal performance of ArangoDB.

$$\text{reward}_{m_i} = \begin{cases} \left(\left(1 + \Delta_{t-1,t}\right)^2 - 1\right)\left(1 + \Delta_{t,0}\right), & \Delta_{t-1,t} \geq 0, \Delta_{t,0} \geq 0, \\ 0, & \Delta_{t-1,t} < 0, \Delta_{t,0} \geq 0, \\ \left(\left(1 - \Delta_{t-1,t}\right)^2 - 1\right)\left(\Delta_{t,0} - 1\right), & \Delta_{t,0} < 0. \end{cases} \tag{3}$$

According to equation (3), a nonnegative reward will be obtained only when the performance of the database is better than the initial state and the historical optimal performance at the same time. Considering that the ultimate goal of tuning is to achieve better performance than the initial setup, there is a need to reduce the impact of the intermediate process of tuning on the design of the reward function. Therefore, when the result of $\Delta_{t-1,t}$ is positive and $\Delta_{t,0}$ is negative, the *reward* must be set to 0.

Different tuning tasks may choose different tuning metrics. Therefore, a weight coefficient ω_i is assigned to the tuning indicator to indicate the tuning direction, so that the tuning system can simultaneously tune multiple indicators. Thus, the final reward function is as defined in equation (4),

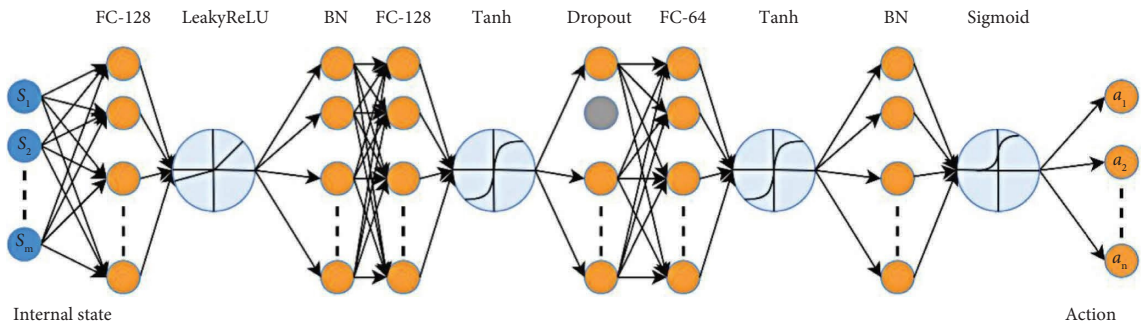

Figure 2: Network structure of Actor.

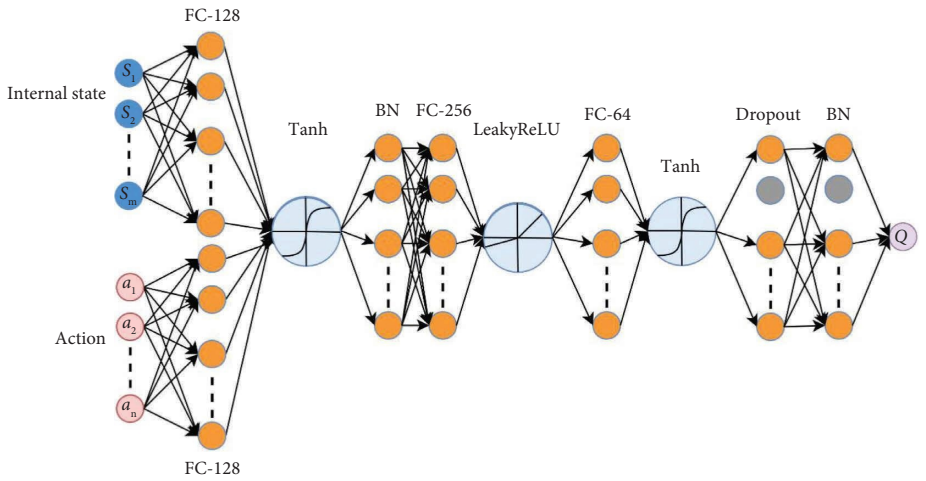

Figure 3: Network structure of Critic.

where the sum of all ω is 1. In this study, the ratio of throughput and latency is the same, both 0.5.

$$r = \sum_i \text{reward}_{m_i} * \omega_i. \tag{4}$$

Algorithm 2 describes the proposed tuning algorithm. First, the Actor networks, the two Critic networks, and their corresponding parameters of the target network are initialized according to the hyperparameters, and the structure and parameters of the target network are the same as those of its corresponding current network. Then, before tuning, initialize the ArangoDB runtime environment, benchmark with ADBench in the initial configuration, and obtain the initial state of the database and external metrics. Then, in the iterative training stage, the Actor network generates future actions according to the current moment and noise. This strategy gives the Actor network a stronger exploration ability of tuning. This ensures the diversity of the generated samples and reduces the possibility of local optimization of the algorithm. Finally, when the number of samples in the experience pool is greater than R, samples will be taken to train the network, and then, the parameters of the Critic network will be updated separately according to the gradient. The corresponding parameters will be updated using the optimizer.

3.4. Benchmarking Platform. The benchmark measurement tool plays an important role in the configuration parameter tuning. The status data and performance indicator data during the tuning process are obtained during the benchmarking of ArangoDB [51]. Database benchmarking tools focus on simulating the most realistic production environment workload and how to accurately measure the indicators reflecting database performance. As a result, excellent database evaluation tools such as TPC-C [52], TPC-DI [53], YCSB [54], Sysbench [55], and UniBench [56] have been created, but they have some limitations.

Among them, YCSB is a relatively perfect benchmark, but it cannot monitor the system resource utilization indicator, and the workload type is relatively simple. UniBench is designed for multimodel database, but it does not support long-running and multithreading, and only one indicator can be monitored. To solve the above problems, we implement a benchmark measurement tool MMDBench for ArangoDB to perform stress tests and measure performance indicators. It consists of two main parts: the load generator and the metric collector. The load generator generates the appropriate workload for the tuning task and launches the execution workload to the ArangoDB, which includes simple read and write operations, complex graph operations, and aggregation operations. The metric collector collects

(1) Initialize replay buffer R
(2) **if** isExist(model) **then**
(3) model.load()
(4) **else**
(5) Initialize actor network μ and critic network Q with weights θ^{μ} and θ^{Q}
(6) $\theta^{\mu'} \longleftarrow \theta^{\mu}$, $\theta^{Q_1'} \longleftarrow \theta^{Q}$, $\theta^{Q_2'} \longleftarrow \theta^{Q}$
(7) Initialize $s_0 \leftarrow \text{cost}(C_d, q)$
(8) **for** epoch = $1, 2, \ldots, M$ **do**
(9) **for** $t = 1, 2, \ldots, T$ **do**
(10) $a_t \longleftarrow \mu(s_t)$
(11) $C_t \longleftarrow$ create_knobs(a_t)
(12) Configure multimodel database with C_t
(13) Perform workload q and observe new state $s_{t+1} \longleftarrow \text{cost}(C_t, q)$ and $r_t \longleftarrow \text{reward}(s_{t+1})$
(14) Push (s_t, s_{t+1}, a_t, r_t) into R
(15) Sample a random mini-batch (s_i, s_{i+1}, a_i, r_i) from R
(16) target $\longleftarrow r_i + \gamma \min(Q_1'(s_{i+1}, \mu'(s_{i+1}) + \epsilon), Q_2'(s_{i+1}, \mu'(s_{i+1}) + \epsilon)$
(17) Update Critics
(18) $\theta^{Q_{m=1,2}} \longleftarrow \arg\min_{\theta^{Q_m}} 1/N \sum_i (\text{target} - Q_m(s_i, a_i))^2$
(19) **if** t mod d **then**
(20) Update μ by $1/N \sum_i \nabla Q_1(s_i, \mu(s_i)) \nabla \mu(s_i)$
(21) $\theta^{\mu'} \longleftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$,
(22) $\theta^{Q_m'} \longleftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q_m'}$
(23) **end if**
(24) $s_t \longleftarrow s_{t+1}$
(25) **end for**
(26) $P \longleftarrow a_T$
(27) **end for**
(28) **end if**

Algorithm 2: Parameter tuning algorithm based on TD3.

statistics that reflect database performance. This part integrates Prometheus to monitor cloud server resource utilization in real time.

4. Experiments

In Section 4.1, the relevant experimental environment and dataset are introduced. Then, the various components of MMDTune+ are tested and analyzed in Section 4.2. In Section 4.3, we compare MMDTune+ with other existing works under different execution workloads.

4.1. Experimental Environment and Dataset. We utilize four Aliyun Cloud servers to build an experimental environment, three of which act as the server to build a three-node cluster, and the remaining one serves as the client to access the database cluster. Table 1 shows the cloud server configuration.

In this study, we design related multimodel database operations on two datasets, including one social commercial network dataset and one hydrologic-related dataset. The data structure and size of each dataset are shown in Table 2.

The databases built by these two datasets both contain multiple data models. The social business network includes the collection of customer, order, stamp post, product, invoice, and evaluation feedback. It also contains the graph formed by the existing relationships between these collections. Hydrologic datasets mainly contain geographic location information and sensor data, including provinces, cities, monitoring stations, and sensor data.

Table 1: The experimental environment.

Attribute	Information
CPU	Intel Xeon platinum 8269@2.6 GHz, 4 core
Memory	16 GB
OS	CentOS 7.6
ArangoDB version	3.7.6
PyTorch version	1.7.1
Prometheus version	2.21.0

Finally, the multimodel database operations designed by MMDBench based on the above two datasets are shown in Table 3. Due to space limitations, only the multimodel database operations required in the experiment are listed.

4.2. Tuning Experiment and Result Analysis. In this section, the effect of MMDTune+ tuning on ArangoDB is evaluated, and relevant experiments are carried out on the three modules included in it to verify its effectiveness and necessity.

4.2.1. Parameter Settings. To verify the generalization ability of the TD3 algorithm in MMDTune+, we select different workloads to execute. The workloads involved in the following experiments, and their corresponding execution

Table 2: Datasets.

Dataset	The number of collection	Data structure	The number of records
Social network	12	Relation document key-value graph	5.77 million+
Water	10	Relation document key-value graph	14 million+

Table 3: Multimodel database operations.

Operation	Data structure	Technical dimensions
Q1	Document; graph	(1) Document query (2) Graph and document join query
Q2	Document; key-value; graph	(1) Document join query (2) Document and key-value join query (3) Graph and document join query (4) Document aggregation
Q3	Document; key-value; graph	(1) Document join query (2) Document and key-value join query (3) Graph and document join query (4) Graph query
Q4	Document; graph	(1) Document join query (2) Graph and document join query
Q5	Document; graph	(1) Document query (2) Graph DFS query
Q6	Document; graph	(1) Graph shortest path query (2) Graph and document join query
Q7	Document; graph	(1) Document query (2) Graph and document join query (3) Document aggregation
I1	Document	Document insert
I2	Document; graph	Document and graph insert
U1	Document	Document update
D1	Document	Document delete
T1	Document; graph	(1) Document insert and update (2) Graph update
T2	Document	Document insert, update, and delete

parameters are shown in Table 4. Considering the limitation of the number of CPU cores in the system, it is not the case that the larger the number of threads, the higher the throughput. Therefore, a unified thread number of 10 is selected, which can be adapted according to the CPU version of the machine. The parameters of the data request mode are selected according to different scenarios.

The execution parameters of the tuning algorithm TD3 are set as follows: the maximum step length of offline training optimization is 1000, the size of the experience playback pool is 10000, and the number of samples is 16. We chose Adam as the optimizer for model training, with a learning rate of 0.0005 for the Actor network and 0.0001 for the Critic network. The discount factor is 0.99, and the exploration strategy parameter is set to 0.2. The update frequency of the target network is 2, and the soft update coefficient is 0.01.

4.2.2. Configuration Parameter Selection Experiment. To verify the effectiveness of the feature sorting method based on the RF algorithm for tuning in MMDTune+, we conduct a set of experiments in workload W1 and tune ArangoDB by increasing the number of configurations sorted by the RF algorithm.

As shown in Figure 4, with the increase in the number of configuration parameters, the throughput improvement rate increases, while the latency decreases continuously. Compared with the default configuration, the performance is significantly improved. Additionally, in the case of the same number of configurations, the performance improvement of the configuration selected based on the RF algorithm is generally higher than that of the randomly selected configuration. Depending on the important feature parameters of the selected configuration, the RF algorithm can select a configuration with a high correlation of performance impact, making it easier for ArangoDB to achieve optimal performance.

Figure 5 shows the convergence rate of the TD3 network model under different configurations selected by the RF algorithm in the previous experiment. It can be seen that with the increase in the number of configurations, the number of iterations of network training also continues to increase. The reason for this situation is that the greater the

Table 4: Workload execution parameters.

Workload	Operation	Request	Thread	Execution time (s)
W1	Q1	SkewedLatest	10	90
W2	Q3	Uniform	10	90
W3	Q4	Zipfian	10	90
W4	Q2	Uniform	10	90
W5	I1	Uniform	10	120
W6	I2	Uniform	10	120
W7	T1	Uniform	10	120
W8	T2	Uniform	10	120
W9	30% I1 and 70% Q1	SkewedLatest	10	120
W10	50% I1 and 50% Q7	Normal	10	120
W11	30% U1 and 70% Q7	Uniform	10	120
W12	60% I1 and 40% Q1	Zipfian	10	120

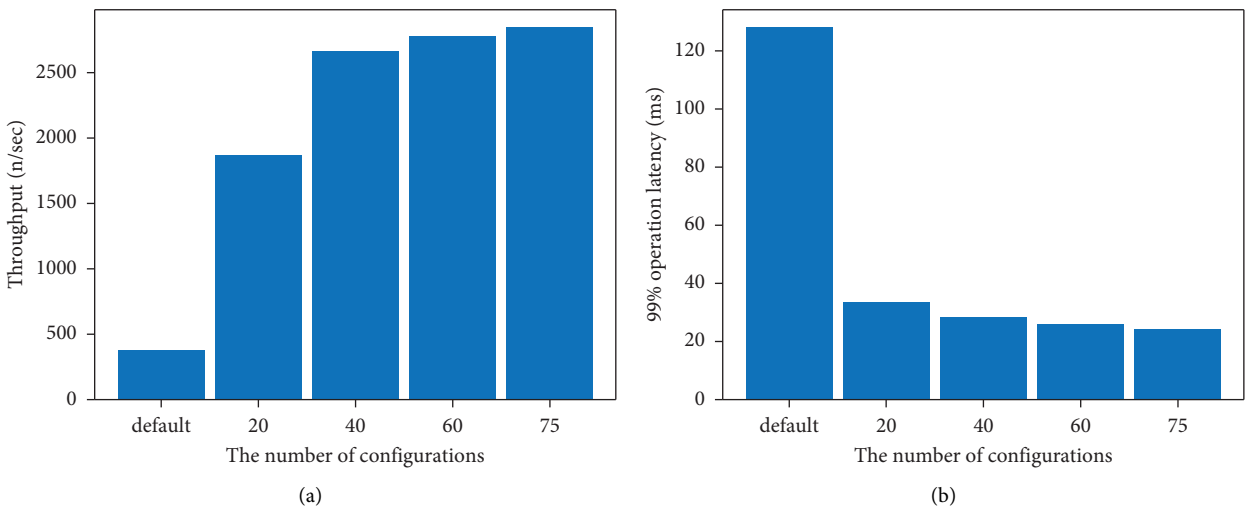

Figure 4: The effect of the number of configurations on the tuning result.

number of configurations, the more parameters in the model, the more complex the structure, and the longer it takes the model to reach consumption. When the number of configuration parameters ranges from 60 to 75, the increase in the number of configuration parameters does not significantly improve the tuning effect. This is because the increase in the number of configuration parameters does not significantly affect the performance. Therefore, to improve the efficiency of tuning, one can select an appropriate number of configurations for tuning when the tuning effect is met. In the subsequent throughput and delay tuning experiments, the configuration parameters recommended by the tuning algorithm were all 75 configurations selected by the RF algorithm.

4.2.3. TD3 Offline Training Tuning Experiment. In this part, the TD3 algorithm in MMDTune+ will be used to tune the 75 configuration parameters selected above in the offline phase, and the effectiveness of the TD3 algorithm for tuning ArangoDB configuration parameters will be further verified. At the same time, the tuning data will be collected for use in the online tuning phase. First, we select workloads W2, W6, W7, and W9 as target workloads for offline training tuning.

Table 5 shows the performance changes of the above four workloads after tuning. Under these four workloads, the corresponding throughput and 99% operation latency have been optimized to different degrees. Therefore, it can be concluded that the TD3 algorithm can be applied to ArangoDB configuration parameter tuning, and the performance improvement after tuning is also considerable. This is because the TD3 algorithm can adapt well to the high-dimensional configuration space and recommend high-quality continuous configuration parameters for the system. Meanwhile, the trial-and-error strategy adopted is similar to the DBA tuning strategy, which can continuously explore the configuration space and reduce the possibility of falling into local optima.

4.2.4. Workload-Aware Mechanism Tuning Experiments. Without using a workload-aware mechanism, the training of the model or tuning process has difficulty using historical experience for learning and requires the user to specify the

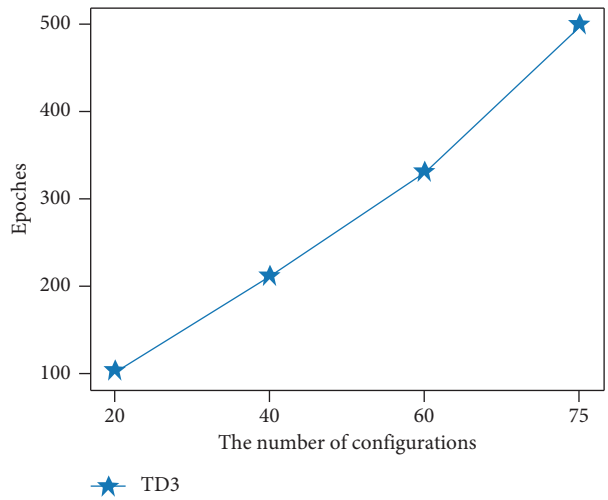

Figure 5: The relationship between the convergence rate of the TD3 algorithm and the number of configurations.

Table 5: Configuration parameter tuning performance changes.

Workload	Indicator	Before tuning	After tuning	Performance improvement (%)
W2	Throughput (n/sec)	36.06	127.67	254.04
	99% operation latency (ms)	488.61	235.09	51.88
W6	Throughput (n/sec)	7939.57	10533.61	32.67
	99% operation latency (ms)	1.25	0.94	24.80
W7	Throughput (n/sec)	944.28	1186.21	25.62
	99% operation latency (ms)	54.20	39.21	27.66
W9	Throughput (n/sec)	1492.43	2214.06	48.35
	99% operation latency (ms)	71.63	64.14	10.46

workload to learn. However, the artificial selection of a similar workload for multimodel database applications is more complex.

Therefore, MMDTune+ uses the k-means++ algorithm to cluster workloads to detect changes in workloads. The final clustering result is shown in Figure 6. Historical workloads can be roughly divided into seven types. k-means++ can classify similar workloads into the same class by using distance.

The specific operation of detecting workload changes is that when MMDTune+ finds that the current workload type is different from the newly arrived workload type, it can start to use the similarity calculation method to match and fine-tune the parameters using the pretraining model of the previous similar workload, and finally recommend high-quality configuration parameters. Next, a new type of workload, W3, will be used to validate the subsequent actions that detected the change using a workload-aware mechanism.

Figure 7 shows the order of correlation coefficients between the new workload W3 and the historical workload. The most similar workload to the target workload is Q25, which consists of multiple queries. Next, the pretraining model of Q25 will be migrated to the current workload W3 to tune its parameters to verify whether the tuning process can take advantage of historical learning to improve tuning efficiency.

In the offline phase under the condition of not using the training model, the TD3 algorithm requires over 500 inclines to achieve convergence. For workload W3, in the use of load sensing history after the training model for fine-tuning results as shown in Figure 8, one can see a new workload tuning, and within 5 steps, one can achieve a good performance. Compared with the previous results, the overall tuning efficiency is greatly improved, and the tuning effect also increases with the increase in the number of tuning steps.

The following conclusions can be drawn from the above experiments. First, MMDTune + based on the deep reinforcement learning algorithm TD3 can learn and gain decision-making experience in complex environments. Second, the tuning algorithm can learn from past experience to improve the tuning efficiency, and the workload-aware mechanism module of MMDTune+ is able to accurately match the empirical knowledge of the historical workload.

During the configuration tuning process, the results of each workload execution will be stored in the data warehouse by MMDTune+. As the configuration tuning system continues to execute, the historical data will increase, which will make the online tuning effect even better.

4.3. Comparison Experiment with Existing Tuning Algorithms. In this section, we fully evaluate the effects of workload-aware mechanism tuning on all modules of MMDTune+ on multiple different workloads and compare the TD3

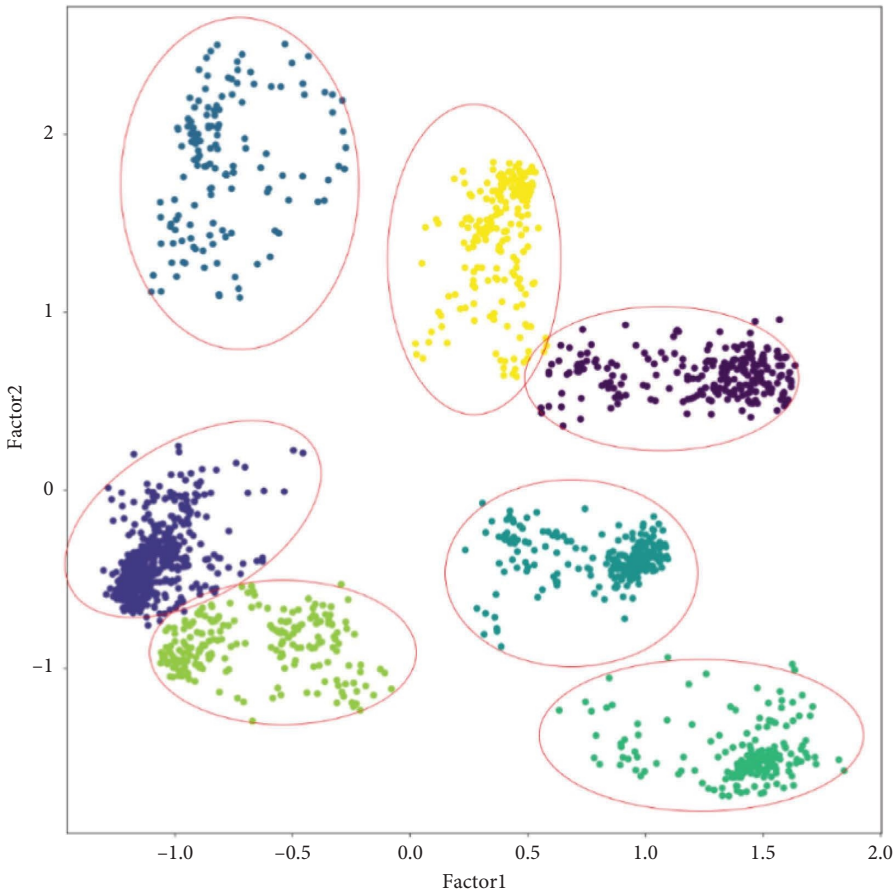

Figure 6: The clustering results of k-means++.

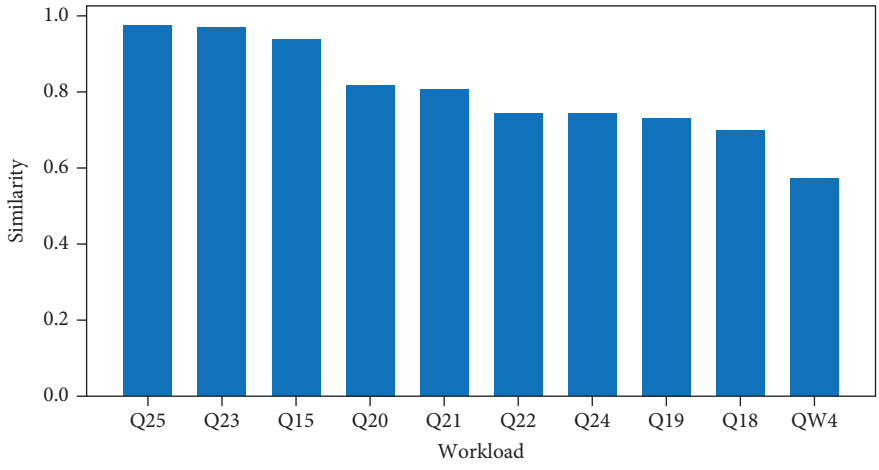

Figure 7: The similarity of workload W3 to the historical workload.

algorithm used by MMDTune+ with other tuning algorithms. First, the time consumption of five algorithms based on SVR, GPR, DNN, DDPG, and TD3 in the whole tuning process was evaluated. To better understand the distribution of execution time in the tuning steps, the time consumption of each module of MMDTune+ is compared. The tuning process mainly consists of five parts with time consumption: data preprocessing, workload-aware mechanism, configuration parameter generation, deployment and execution of benchmark evaluation, and model weight update. The experimental results are shown in Table 6. The consumption time of data preprocessing and the workload-aware mechanism of each algorithm are the same, so they will not be discussed.

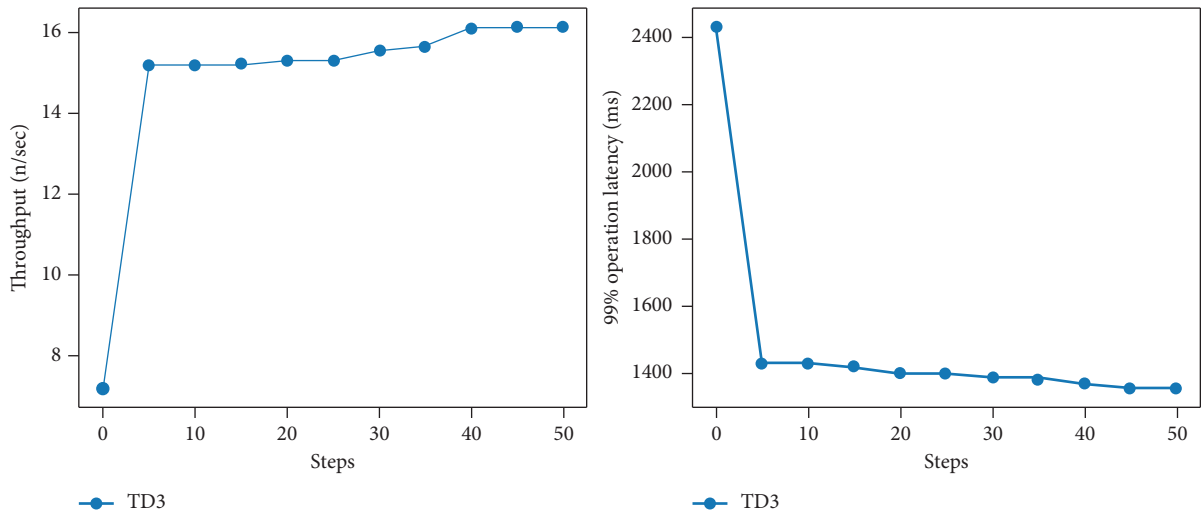

Figure 8: The relationship between the number of tuning steps and performance.

Table 6: Time consumption of different tuning algorithms.

Algorithms	Step	Configuration parameter generation (ms)	Model weight update (ms)
SVR	10	29.13	411.44
GPR	10	57.06	452.21
DNN	10	19.93	291100
DDPG	10	2.13	272
TD3	10	2.11	451

The time complexity of machine learning algorithms is typically expressed in terms of big O notation, which describes the upper bound on the growth rate of the computational resources required to solve a problem of a given size. SVR is a linear regression algorithm that finds the hyperplane that best separates the data into classes. The time complexity of SVR is typically O (n^3), where n is the number of training samples. This makes SVR relatively fast for small datasets, but it can become slow for larger datasets. GPR is a nonparametric regression algorithm that models the relationship between the inputs and outputs using a Gaussian process. The time complexity of GPR is typically O (n^3), where n is the number of training samples. This makes GPR relatively fast for small datasets, but it can become slow for larger datasets. DNNs are composed of multiple layers of artificial neurons and are used for a wide range of tasks, including image classification, speech recognition, and natural language processing. The time complexity of DNNs is typically O (mn^2), where m is the number of training samples and n is the number of neurons. This makes DNNs relatively slow for small datasets, but they can scale to large datasets. DDPG is a reinforcement learning algorithm that is used to train agents in control tasks. The time complexity of DDPG is highly dependent on the complexity of the environment, and the number of interactions required to train the agent. As a result, the time complexity of DDPG can vary widely and can be difficult to estimate. TD3 is a variant of DDPG that is used to train agents in control tasks. Similar to DDPG, the time complexity of TD3 is highly dependent on the complexity of the environment, and the number of interactions required to train the agent. As a result, the time complexity of TD3 can vary widely and can be difficult to estimate. The actual time complexity of these algorithms can vary widely depending on the specific implementation, the size and structure of the data, and the hardware used.

As seen in the millisecond level, the TD3 algorithm used by MMDTune+ is second only to DDPG in tuning efficiency, but the difference is not significant, and the efficiency is faster than other algorithms. SVR, GPR, and DNN are based on supervised learning methods. This type of algorithm requires each step to complete model training convergence and add Gaussian noise for exploration to meet the tuning requirements. Moreover, with the increase in training samples, its time consumption will gradually increase.

Meanwhile, we compare the tuning results of different algorithms with the proposed methods. First, experiments are carried out on workload W4 composed of multiple queries. The workload contains three data models and different data operations, which can better represent the characteristics of a multimodel database. As can be seen from Figure 9, multiple tuning algorithms are able to optimize performance well, including extended related algorithms of other studies. On workload W4, the throughput is improved by more than 132.54%, and the 99% operation latency is reduced by more than 7%. Among them, DDPG and TD3 are better than SVR, GPR, and DNN which use regression characteristics, and TD3 is the best.

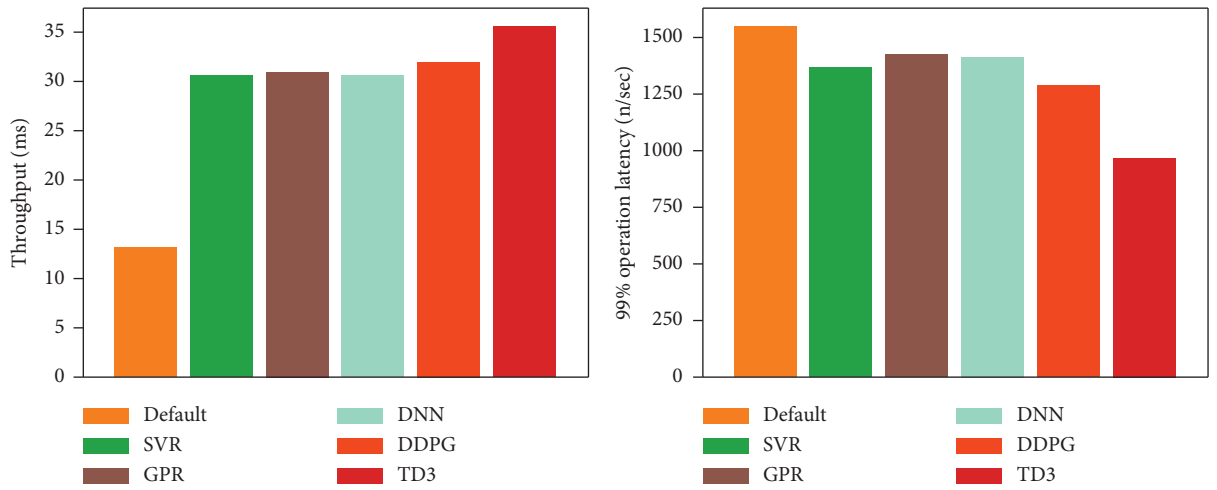

Figure 9: Workload W4 tuning performance comparison.

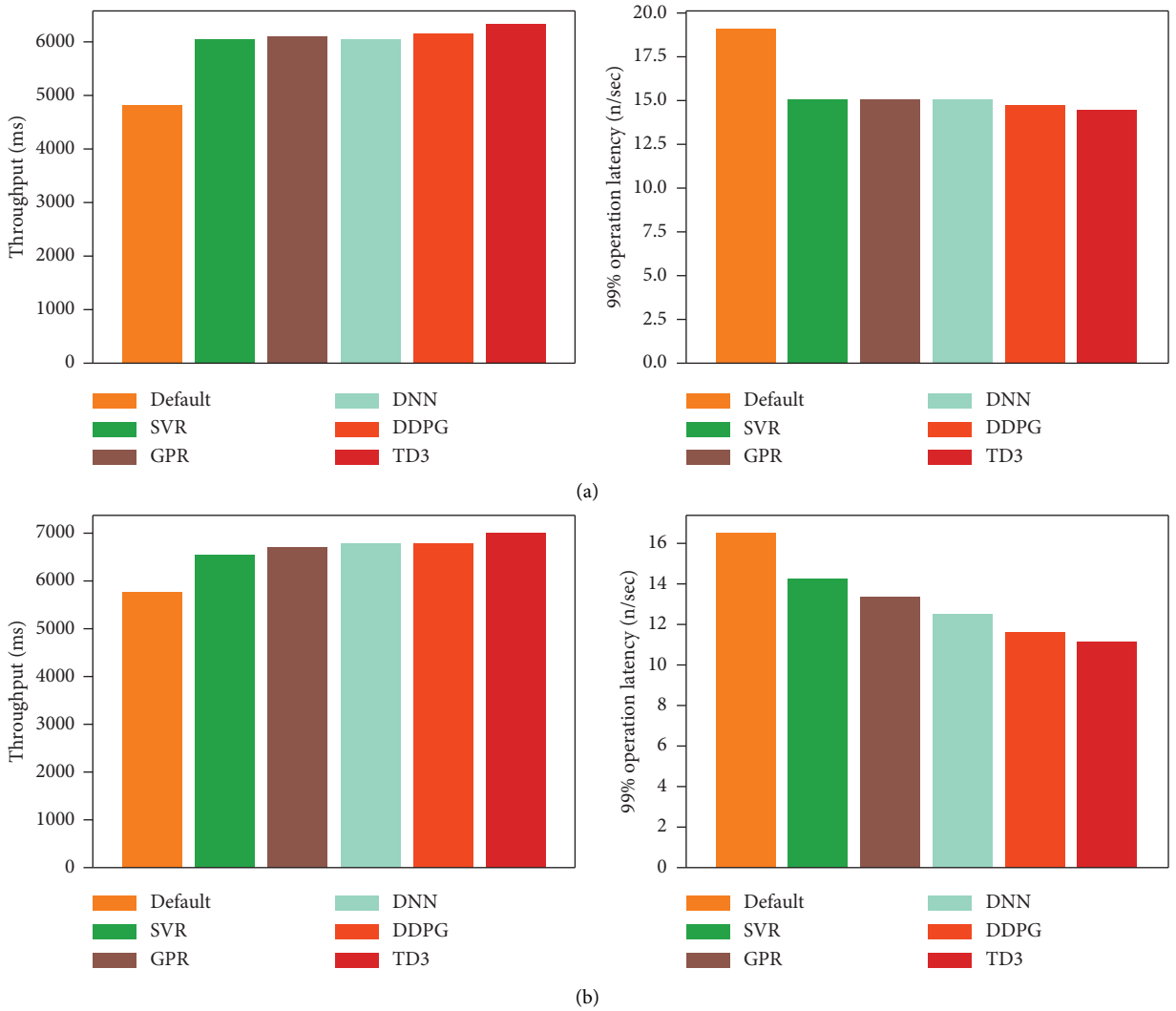

Figure 10: Workload W4 and W8 tuning performance comparison.

Related experiments were performed on write workload W5 and transaction workload W8. As shown in Figure 10, the throughput of workload W5 increased by 31.1% after TD3 tuning, and the 99% operation latency decreased by 25.55%. Other tuning algorithms also achieved good tuning effects, but they were slightly weaker than TD3, and the effect of W8 was similar to that of W5.

The tuning results based on the TD3 algorithm were the best on most workloads, and its speed was not far from that of the most efficient DDPG algorithm, which is more efficient than other supervised learning methods. This is because reinforcement learning uses well-known exploration and utilization strategies through the interaction of agents with the database environment, which not only gives full play to the capability of the model but also explores a better configuration that has never been tried before and reduces the possibility of falling into local optima. Although DDPG is also a deep reinforcement learning algorithm, its cumulative errors have a negative impact on the tuning effect, which makes the tuning effect of this method inferior to that of TD3. As supervised learning algorithms, SVR, GPR, and DNN use the characteristics of regression to predict performance by using configuration, which makes them rely on a large amount of high-quality training data.

4.4. Limitations. MMDTune+ realizes the performance optimization of ArangoDB, but there are still some problems that need to be further studied and solved. First, the MMDBench benchmarking tool requires improvements as it currently only conducts tests on ArangoDB, a multimodel database, and simulates a limited number of real-world scenarios. Currently, MMDTune+ has only been applied to one tuning object. As the number of data models and storage engines increases, the search space of tuning parameters can become increasingly large, making it more challenging to find an optimal solution. The performance of the MMDTune+ may depend on the specific characteristics of the database, such as the number and distribution of data types, the size of the database, and the complexity of queries. Different types of queries and data access patterns may require different tuning parameters, and it may be difficult to identify a single set of parameters that works well across all possible workload scenarios.

5. Conclusion

The emergence of multimodel databases provides a new solution to effectively address the deficiencies of traditional database. ArangoDB is a widely used multimodel database. However, ArangoDB has a difficult problem with configuration parameter optimization, which needs to be tuned for the actual workload or application. At the same time, the tuning is not only for a single workload but also for the dynamic change of the workload exploratory automatic tuning, and how to sense the changes of the workload and further use the historical experience to improve the tuning effect is particularly important. Under the above background, we study the configuration parameter tuning of the multimodel database ArangoDB.

Aiming to address the ArangoDB configuration parameter tuning problem, an ArangoDB configuration parameter tuning tool MMDTune+ combined with a workload-aware mechanism is proposed, which includes a configuration parameter selection module, workload-aware mechanism, and tuning algorithm. The configuration parameter selection module uses the random forest algorithm to select configuration parameters with high correlation to the tuning indicator, to meet the performance improvement rate, and to improve the tuning efficiency. Workload-aware mechanism is based on k-means ++ and Pearson correlation coefficients to detect workload changes and match historical empirical data or pretraining models of similar workloads and migrate them to the current task to improve performance gains and efficiency. Finally, based on our improved TD3 algorithm, we perform database tuning for ArangoDB.

The current limitations with our existing work include insufficient workload, a single tuning target, and a lack of targeted optimization for different tuning targets in the TD3 algorithm. In future work, we can apply the configuration parameter tuning algorithm to other multimodel databases, build more workloads with more diverse datasets, and simulate more realistic complex scenarios.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was partly supported by the National Key R&D Program of China (2019YFE0109900), Water Science and Technology Project of Jiangsu Province (2022003, 2022057), and Jiangsu Province Key Research and Development Program (Modern Agriculture) Project (BE2018301).

References

[1] R. Han, L. K. John, and J. Zhan, "Benchmarking big data systems: a review," *IEEE Transactions on Services Computing*, vol. 11, no. 3, pp. 580–597, 2018.

[2] P. Sawadogo and J. Darmont, "On data lake architectures and metadata management," *Journal of Intelligent Information Systems*, vol. 56, no. 1, pp. 97–120, 2021.

[3] J. Lu and I. Holubová, "Multi-model databases: a new journey to handle the variety of data," *ACM Computing Surveys*, vol. 52, no. 3, pp. 1–38, 2019.

[4] S. Bimonte, E. Gallinucci, P. Marcel, and S. Rizzi, "Logical design of multi-model data warehouses," *Knowledge and Information Systems*, vol. 65, no. 3, pp. 1067–1103, 2022.

[5] Z. H. Liu, J. Lu, D. Gawlick, H. Helskyaho, G. Pogossiants, and Z. Wu, "Multi-model database management systems-a look forward," in *Heterogeneous Data Management, Polystores,*

and Analytics for Healthcare, pp. 16–29, Springer, Berlin, Germany, 2019.

[6] N. B. Hunt, H. Gardarsdottir, M. T. Bazelier, O. H. Klungel, and R. Pajouheshnia, "A systematic review of how missing data are handled and reported in multi-database pharmacoepidemiologic studies," *Pharmacoepidemiology and Drug Safety*, vol. 30, no. 7, pp. 819–826, 2021.

[7] H. Herodotou, Y. Chen, and J. Lu, "A survey on automatic parameter tuning for big data processing systems," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–37, 2020.

[8] X. Wang, M. J. Carey, and V. J. Tsotras, "Subscribing to big data at scale," *Distributed and Parallel Databases*, vol. 40, no. 2-3, pp. 475–520, 2022.

[9] E. Guliyev, *Comparative Analysis of Multi-Model Databases*, Univerzita Karlova, Prague, Czech Republic, 2022.

[10] S. Bimonte, E. Gallinucci, P. Marcel, and S. Rizzi, "Data variety, come as you are in multi-model data warehouses," *Information Systems*, vol. 104, Article ID 101734, 2022.

[11] J. Zhang, K. Zhou, G. Li et al., "CDBTune+: an efficient deep reinforcement learning-based automatic cloud database tuning system," *The VLDB Journal*, vol. 30, no. 6, pp. 959–987, 2021.

[12] M. Seidemann, N. Glombiewski, M. Körber, and B. Seeger, "Chronicledb: a high-performance event store," *ACM Transactions on Database Systems*, vol. 44, no. 4, pp. 1–45, 2019.

[13] M. Nasar and M. A. Kausar, "Suitability of influxdb database for iot applications," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 10, pp. 1850–1857, 2019.

[14] B. Hentschel, P. J. Haas, and Y. Tian, "General temporally biased sampling schemes for online model management," *ACM Transactions on Database Systems*, vol. 44, no. 4, pp. 1–45, 2019.

[15] F. Ye, Y. Li, X. Wang, N. Nedjah, P. Zhang, and H. Shi, "Parameters tuning of multi-model database based on deep reinforcement learning," *Journal of Intelligent Information Systems*, pp. 1–24, 2022.

[16] N. Zaniewicz and A. Salamończyk, "Comparison of MongoDB, Neo4j and ArangoDB databases using the developed data generator for NoSQL databases," *Studia Informatica. System and information technology*, vol. 26, no. 1, pp. 61–72, 2022.

[17] S. Mydhili, S. Periyanayagi, S. Baskar, P. M. Shakeel, and P. Hariharan, "Retracted article: machine learning based multi scale parallel K-means++ clustering for cloud assisted internet of things," *Peer-to-Peer Networking and Applications*, vol. 13, no. 6, pp. 2023–2035, 2020.

[18] J. Zhou, S. Xue, Y. Xue, Y. Liao, J. Liu, and W. Zhao, "A novel energy management strategy of hybrid electric vehicle via an improved TD3 deep reinforcement learning," *Energy*, vol. 224, Article ID 120118, 2021.

[19] C. Zhang and J. Lu, "Holistic evaluation in multi-model databases benchmarking," *Distributed and Parallel Databases*, vol. 39, no. 1, pp. 1–33, 2021.

[20] R. Wu and E. Keogh, "Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, p. 1, 2021.

[21] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1009–1024, Chicago, IL, USA, May 2017.

[22] J. Zhang, Y. Liu, K. Zhou et al., "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in *Proceedings of the 2019 International Conference on Management of Data*, pp. 415–432, Amsterdam, Netherlands, June 2019.

[23] J. K. Ge, Y. F. Chai, and Y. P. Chai, "WATuning: a workload-aware tuning system with attention-based deep reinforcement learning," *Journal of Computer Science and Technology*, vol. 36, no. 4, pp. 741–761, 2021.

[24] Z. Zewdu, M. K. Denko, and M. Libsie, "Workload characterization of autonomic DBMSs using statistical and data mining techniques," in *Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops*, pp. 244–249, Bradford, UK, May 2009.

[25] S. Elnaffar and P. Martin, "The Psychic–Skeptic Prediction framework for effective monitoring of DBMS workloads," *Data and Knowledge Engineering*, vol. 68, no. 4, pp. 393–414, 2009.

[26] G. Kul, D. T. A. Luong, T. Xie, V. Chandola, O. Kennedy, and S. Upadhyaya, "Similarity metrics for SQL query clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 12, pp. 2408–2420, 2018.

[27] N. Takahashi, "Development of data extraction method based on clustering method and CVI," in *Proceedings of the CIRED 2020 Berlin Workshop (CIRED 2020)*, pp. 517–520, Berlin, Germany, September 2020.

[28] D. Van Aken, D. Yang, S. Brillard et al., "An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems," *Proceedings of the VLDB Endowment*, vol. 14, no. 7, pp. 1241–1253, 2021.

[29] S. Chaudhuri and V. Narasayya, "Self-tuning database systems: a decade of progress," in *Proceedings of the 33rd international conference on Very large data bases*, pp. 3–14, Vienna, Austria, September 2007.

[30] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 international conference on management of data*, pp. 489–504, Houston, TX, USA, May 2018.

[31] W. G. Pedrozo, J. C. Nievola, and D. C. Ribeiro, "An adaptive approach for index tuning with learning classifier systems on hybrid storage environments," in *Proceedings of the International conference on hybrid artificial intelligence systems*, pp. 716–729, Berlin, Germany, June 2018.

[32] A. Pavlo, C. Curino, and S. Zdonik, "Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 61–72, Scottsdale, AZ, USA, May 2012.

[33] X. Liang, A. J. Elmore, and S. Krishnan, "Opportunistic view materialization with deep reinforcement learning," 2019, https://arxiv.org/abs/1903.01363.

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[35] E. Zemmour, P. Kurtser, and Y. Edan, "Automatic parameter tuning for adaptive thresholding in fruit detection," *Sensors*, vol. 19, no. 9, p. 2130, 2019.

[36] A. J. Storm, C. Garcia-Arellano, S. S. Lightstone, Y. Diao, and M. Surendra, "Adaptive self-tuning memory in DB2," in *Proceedings of the 32nd international conference on Very large data bases*, pp. 1081–1092, Seoul, Korea, September 2006.

[37] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, 2020.

[38] A. Paszke, S. Gross, F. Massa et al., "Pytorch: an imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[39] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: a survey," 2019, https://arxiv.org/abs/1905.05055.

[40] S. Lu, Z. Zhu, J. M. Gorriz, S. H. Wang, and Y. D. Zhang, "NAGNN: classification of COVID-19 based on neighboring aware representation from deep graph neural network," *International Journal of Intelligent Systems*, vol. 37, no. 2, pp. 1572–1598, 2022.

[41] Y. Liu, D. Li, S. Wan et al., "A long short-term memory-based model for greenhouse climate prediction," *International Journal of Intelligent Systems*, vol. 37, no. 1, pp. 135–151, 2022.

[42] C. Zheng, Z. Ding, and J. Hu, "Self-tuning performance of database systems with neural network," in *Proceedings of the International Conference on Intelligent Computing*, pp. 1–12, Berlin, Germany, August 2014.

[43] W. Xiong, K. Yang, and H. Dai, "Improving nosql's performance metrics via machine learning," in *Proceedings of the 2019 Seventh International Conference on Advanced Cloud and Big Data (CBD)*, pp. 90–95, Suzhou, China, September 2019.

[44] J. Tan, T. Zhang, F. Li et al., "ibtune: individualized buffer tuning for large-scale cloud databases," *Proceedings of the VLDB Endowment*, vol. 12, no. 10, pp. 1221–1234, 2019.

[45] S. Kwon and S. Kwon, "Optimal feature selection based speech emotion recognition using two-stream deep convolutional neural network," *International Journal of Intelligent Systems*, vol. 36, no. 9, pp. 5116–5135, 2021.

[46] Y. Liu, L. Yao, B. Li, C. Sammut, and X. Chang, "Interpolation graph convolutional network for 3D point cloud analysis," *International Journal of Intelligent Systems*, vol. 37, no. 12, pp. 12283–12304, 2022.

[47] T. Subba Reddy, J. Harikiran, M. K. Enduri et al., "Hyperspectral image classification with optimized compressed synergic deep convolution neural network with aquila optimization," *Computational Intelligence and Neuroscience*, vol. 2022, Article ID 6781740, 14 pages, 2022.

[48] G. Li, X. Zhou, S. Li, and B. Gao, "Qtune: a query-aware database tuning system with deep reinforcement learning," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2118–2130, 2019.

[49] I. Trummer, J. Wang, Z. Wei et al., "Skinnerdb: regret-bounded query evaluation via reinforcement learning," *ACM Transactions on Database Systems*, vol. 46, no. 3, pp. 1–45, 2021.

[50] D. Ran, H. Jiaxin, and H. Yuzhe, "Application of a combined model based on K-means++ and XGBoost in traffic congestion prediction," in *Proceedings of the 2020 5th International Conference on Smart Grid and Electrical Automation (ICSGEA)*, pp. 413–418, Zhangjiajie, China, June 2020.

[51] J. Cowls, A. Tsamados, M. Taddeo, and L. Floridi, "A definition, benchmark and database of AI for social good initiatives," *Nature Machine Intelligence*, vol. 3, no. 2, pp. 111–115, 2021.

[52] A. Kamsky, "Adapting TPC-C benchmark to measure performance of multi-document transactions in MongoDB," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2254–2262, 2019.

[53] M. Poess, T. Rabl, H. A. Jacobsen, and B. Caufield, "TPC-DI: the first industry benchmark for data integration," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1367–1378, 2014.

[54] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 143–154, Indianapolis, IN, USA, June 2010.

[55] J. W. Krogh, "Benchmarking with Sysbench," in *MySQL 8 Query Performance Tuning*, pp. 19–53, Springer, Berlin, Germany, 2020.

[56] C. Zhang, J. Lu, P. Xu, and Y. Chen, "UniBench: a benchmark for multi-model database management systems," in *Proceedings of the Technology conference on performance evaluation and benchmarking*, pp. 7–23, Berlin, Germany, January 2018.