Journal of Intelligent Information Systems (2023) 61:167–190
https://doi.org/10.1007/s10844-022-00762-0

Parameters tuning of multi-model database based on deep reinforcement learning

Feng Ye[1] · Yang Li[1] · Xiwen Wang[1] · Nadia Nedjah[2] · Peng Zhang[3] · Hong Shi[4]

Received: 30 December 2021 / Revised: 29 October 2022 / Accepted: 31 October 2022 /
Published online: 30 November 2022

Abstract
As we all know, the performance of database management system is directly linked to a vast array of knobs, which control various aspects of system operation, ranging from memory and thread counts settings to I/O optimization. Improper settings of configuration parameters are shown to have detrimental effects on performance, reliability and availability of the overall database management system. This is also true for multi-model databases, which use a single platform to support multiple data models. Existing approaches for automatic DBMS knobs tuning are not directly applicable to multi-model databases due to the diversity of multi-model database instances and workloads. Firstly, in cloud environment, they have difficulty adapting to changing environments and diverse workloads. Secondly, they rely on large-scale high-quality training samples that are difficult to obtain. Finally, they focus primarily on throughput metrics, ignoring tuning requirements for resource utilization. Therefore, in this paper, we propose a multi-model database configuration parameters tuning solution named MMDTune. It selects influential parameters, recommends the optimal configurations in a high-dimensional continuous space. For different workloads, the TD3 algorithm is improved to generate reasonable parameter adjustment plans according to the internal state of the multi-model databases. We conduct extensive experiments under 5 different workloads on real cloud databases to evaluate MMDTune. Experimental results show that MMDTune adapts well to a new hardware environment or workloads, and significantly outperforms the representative tuning tools, such as OtterTune, CDBTune.

Keywords Multi-model database · Tuning · Deep reinforcement learning · TD3

✉ Feng Ye
yefeng1022@hhu.edu.cn

[1] School of Computer and Information, Hohai University, Nanjing, 211100, China

[2] State University of Rio de Janeiro, Rio de Janeiro, Brazil

[3] Jiangsu Provincial Department of Water Resources, Nanjing, 210029, China

[4] Nanjing GuangSha Software LTD, Nanjing, 210000, China

1 Introduction

In practice, as the world becomes more interconnected, we are witnessing a torrent of digital data with different structures produced by various hardware or software. How to store and manage data of multiple models (Sawadogo & Darmont, 2021) and how to facilitate data interoperation (Braun et al., 2022) become key issues. Multi-model databases (Płuciennik & Zgorzałek, 2017) become a feasible and burgeoning solution, which can be understood as a database that can store data in different formats (relational, document, graph, object, etc) under one management system.

The performance of a multi-model database depends mainly on hundreds of tunable knobs that control many aspects, such as memory allocation, I/O optimization, query planning overhead, and other behaviors (Gordon-Ross & Vahid, 2007). Due to the diversity of workloads and the flexibility of the environment, multi-model databases are often not in the best state, and even deteriorate. So it is not possible to rely on a few experienced database administrators (DBAs) to set appropriate knob configurations. Most existing database automatic tuning studies rely on search-based algorithms and learning-based algorithms to recommend knobs. However, they are less able to adapt to the changing environment and more diverse workloads in the cloud or rely on large-scale high-quality training samples that are difficult to obtain. Moreover, as multi-model database is capable of managing multiple data models at the same time, it can implement CRUD operations on various data models and complex cross-model transactions. However, existing benchmarking platforms focus on relational databases and single data schema NoSQL stores (Davoudian et al., 2018; Huang et al., 2017), which make tuning significantly limited. This is also a challenge, with two main aspects: the complexity of the parameters and the heterogeneity of multi-model database workloads. For example, workloads of multi-model database are more diversified than traditional databases, and workloads may contain retrieval of documents, as well as operations on graph data. Meanwhile, some of the existing database benchmarks do not support long-term stress testing, and they only consider using execution time to evaluate performance and lack other metrics, such as latency.

To solve the above problems, enrich the research on knob tuning of multi-model database, and explore the feasibility and effectiveness of tuning methods on multi-model database, we propose a performance tuning solution of multi-model database based on deep reinforcement learning called MMDTune. It consists of three parts, which are the multi-round sensitivity analysis method (Borgonovo & Plischke, 2016; Sobol, 2001; Zadeh et al., 2017), the knob tuning algorithm based on improved TD3 (Fujimoto et al., 2018; Dankwa & Zheng, 2019) and the benchmarking platform oriented to the multi-model databases. Among them, multi-round sensitivity method is used to select the configuration parameters that have a significant impact on the metrics. The tuning algorithm based on improved TD3 is used to recommend knob parameters for multi-model databases. It interacts with the real environment of the multi-model database so that it can be tuned without prior preparation of training samples. Moreover, using the trial-and-error strategy can make the interaction samples more diverse and increase the possibility to find the optimal configuration. The benchmarking platform includes various workloads and can collect performance metrics. In addition, the benchmarking platform introduces Prometheus (Prometheus Team, 2022) to collect the performance metrics of the multi-model database, so that it not only accurately evaluates the throughput of the multi-model database, but also monitors the resource utilization rate of the system in real time.

The contributions of our work are summarized as below: (1) Extended Sobol method is used to carry out multi-round sensitivity analysis on the tunable knobs to extract the key parameters, so as to reduce the size of the network search space and effectively avoid over fitting. (2) An improved TD3 algorithm is proposed, and the effectiveness of the algorithm is verified on the benchmarking platform proposed for different tuning tasks, operating environments and tuning objects. The experimental results show that MMDTune can recommend optimal configuration schemes under different scenarios. At the same time, compared with the existing database tuning methods OtterTune (Van Aken et al., 2017) and CDBTune (Zhang et al., 2019), MMDTune is able to further improve performance, namely gives a higher throughput and lower resource utilization.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 introduces the overall framework and components of MMDTune. In Section 4, the details and results of the experiments are described. At last, the conclusion is made and the prospect is put forward.

2 Related work

There are two classes of representative studies in DBMS configuration tuning. As shown in Table 1, they are the search-based approaches and the learning-based approaches.

2.1 Search-based approaches

IBM DB2 has released a self-tuning memory manager (Storm et al., 2006; Tian et al., 2003) that combines runtime simulation modeling with cost-benefit analysis to efficiently allocate memory to the DBMS's internal components using a heuristic approach. BestConfig (Zhu et al., 2017) divides the high-dimensional knob space into several subspaces, and uses heuristic methods to search for the optimal configuration from the history records, so as to realize recommending optimal configuration under the condition of limited resources. Tran et al. (2008) uses linear and quadratic regression models to conduct buffer adjustment, which can optimize buffer partitions, ensure fair buffer recovery, and dynamically adjust allocation when workloads change. Wei et al., (2014) proposes a performance tuning framework that can generate rules and use those rules for tuning. Similarly, D-Tunes (PN et al., 2013) provides a tuning solution for distributed database storage that uses an analysis model to capture the relationship between workloads and database performance and introduces self-tuning algorithms to accommodate workload changes over a short time horizon.

Table 1 Representative research on database parameter tuning technology

Type	Literature	Methodology
Search-based approaches	13, 14, 15, 16, 17, 18	Using rules or heuristics to search for the best database parameter configuration.
Learning-based approaches	11, 12, 19, 20, 21, 31, 32	Learning the mappings between parameter combinations and target data to recommend the configuration for optimal database performance.

The rule-based approach above has some limitations: in the case of many parameters and large state space, the tuning process needs to test a very large number of samples, which is very inefficient, and the end result may fall into a local optimal situation.

2.2 Learning-based approaches

Duan et al. (2009) introduces iTuned, the first tool for database knobs tuning using predefined experiments. iTuned uses statistical methods to find the most influential knobs, and establishes the Gaussian process response surface model for automatic configuration adjustment. Researchers from Carnegie Mellon University develop an automatic parameter adjustment tool OtterTune (Van Aken et al., 2017). It builds a ML (Machine Learning) model by maintaining a knowledge base accumulated in a previous tuning process, and by capturing the response of the database system to different parameter settings, it recommends the setting of the knobs. Basu et al. (2016) proposes a learning method to adjust database performance. It learns the cost model through reinforcement learning, and models the execution of query and updates as a Markov decision process. Its state is database configuration, action is configuration change, and return is a function of configuration change cost and query and updates evaluation. However, this method is only proved to be feasible in index tuning, and whether it is suitable for other aspects of database configuration needs further study. In Van Aken et al. (2017), researchers design an end-to-end cloud database automatic adjustment system CDBTune using deep reinforcement learning. It uses DDPG (Wu et al., 2018; Fekry et al., 2020) algorithm to find the optimal configuration for cloud database in high-dimensional continuous space. Then, Li et al. (2019) proposes a database tuning system QTune in query dimension. Similarly, the system combines reinforcement learning with neural network, and adds a predictor on the basis of DDPG to predict the changes of external metrics before and after query processing, which finally proves the effectiveness of the model. In Zhang et al. (2021), an improved version of CDBTune + has been released. Compared with the original one, a big improvement in this paper is the use of Prioritized Experience replay (Schaul et al., 2015) in the tuning process, which speeds up the convergence of model training and greatly improves the efficiency of tuning. However, DDPG algorithm has the problem of over estimation in the process of training. Van Aken et al. (2021) conducts a comprehensive evaluation of ML-based DBMS knob tuning methods in an enterprise database application, and it is verified that GPR, DNN and DDPG can be effectively applied to the knob tuning work and their differences in this scenario. The validity of the learning-based algorithms for the database are tested. However, the applicability and effectiveness of the above methods in multi-model databases have not been verified.

The automatic tuning of database has achieved some research results, and the quality of tuning is gradually improving. However, due to the complexity and diversity of multi-model databases, the related research is not abundant. In addition, the methods based on ML do not perform well in high dimension continuous space. Although the methods based on deep learning has certain ability to understand and recommend configurations, it is one-sided and inefficient to some extent.

3 Framework of MMDTune

To realize the automatic operation after the tuning target is determined, we propose a knob tuning solution named MMDTune for multi-model databases. Figure 1 shows the overall framework, which consists of three parts: the multi-round sensitivity analysis method, the

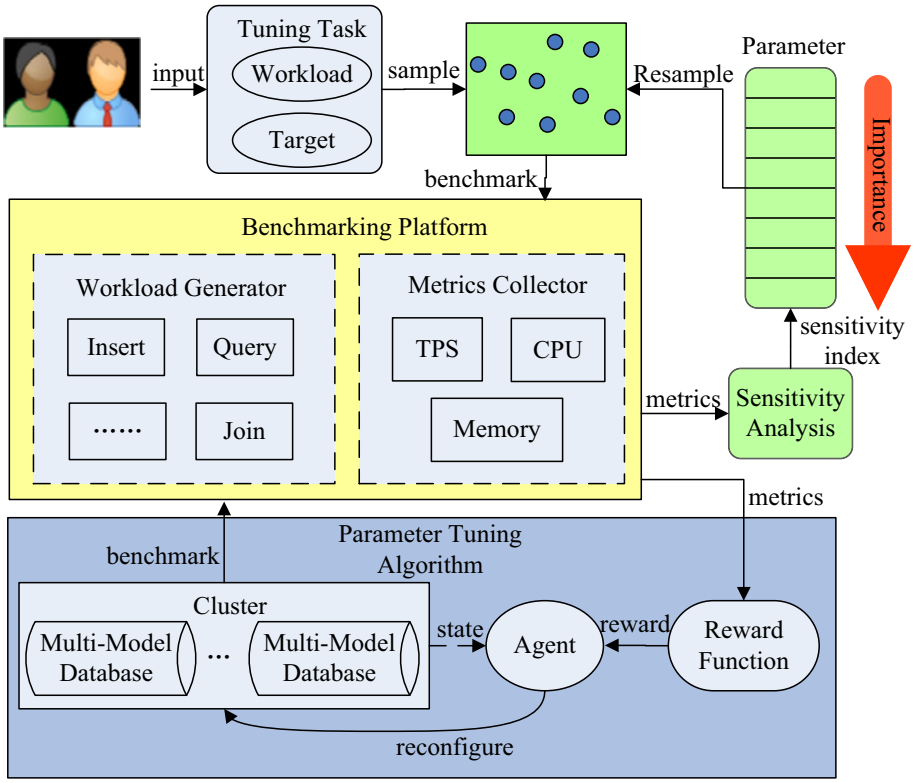

Fig. 1 The framework of MMDTune

parameter tuning algorithm and the benchmarking platform. Among them, the multi-round sensitivity analysis method is to explore important parameters related to the tuning task through the iterative Sobol method. The tuning algorithm based on improved TD3 is used to recommend optimal configuration. The benchmarking platform is used to generate workloads and collect performance metrics from the multi-model database system. During the whole tuning steps, the benchmarking platform provides metrics data for the multi-round sensitivity analysis and knob tuning algorithm simultaneously.

The first step of program execution is to receive the specified tuning tasks, including the workload and the performance metrics to be optimized. Next, a corresponding file repository is generated to store the metrics collected by the benchmarking platform and the network parameters in the tuning algorithm. In the initial stage of automatic tuning, the multi-round sensitivity analysis will calculate the sensitivity of each parameter for a specific tuning task to reduce the size of the search space. Once these parameters are identified, the tuning algorithm begins to build a strategy and value network based on deep reinforcement learning to explore the optimal configuration. Benchmarking again plays an important role at this stage. It will serve as the middleware, connecting to the running environment at one end. The other end is connected to the agent in the algorithm, which is used to provide external state data for the agent. For different multi-model databases, we can extend the interfaces in the benchmark to guide experiments and recommend the best parameters.

3.1 Identifying important parameters

Parameters have a significant impact on overall performance (Lu & Holubová, 2019). Trying to adjust parameters that have no effect is not only a waste of resources, but can lead to over-fitting results. Therefore, in the initial stage of tuning, parameters that are positively or negatively correlated with performance should firstly be found, and these parameters should be tuned and trained to achieve better learning effects and more efficiency. MMDTune combines sensitivity analysis method to investigate how the variation in execution cost (for example, execution time) of a multi-model database is attributable to different configuration parameters. The relationship between them is not simply linear. There are dependencies between some of the multi-model database's knobs, so changing one parameter may affect the other. For example, in OrientDB (Lu et al., 2018), the maximum heap space and disk cache are tunable parameters. In theory, increasing the heap cache and disk cache will improve the performance of running the multi-model database, but if their sum is too high, it will cause a huge slowdown. Based on the above two points, MMDTune uses a global sensitivity analysis method to select important parameters.

Traditional sensitivity analysis method typically involves running intensive off-line benchmarks with many different configuration values and constructing a set of influencing parameters by analyzing the performance differences caused by each configuration parameter. Not only is it expensive to apply this approach directly, but it also takes hundreds of executions, which can be quite time consuming. Therefore, MMDTune uses an iterative Sobol method to find key parameters that have a significant impact on performance metrics. In each iteration, the approach at first uses Monte Carlo method to sample in the parameter space of the multi-model database. It then combines the large amount of sampled data and configures them separately into a multi-model database. The benchmarking platform is then used to execute specific workloads and the resulting measurements are used to calculate the corresponding sensitivity metrics. The principle of the Sobol method is to assume that the variance of the model output is the sum of the variances of a single parameter and the combination of each parameter. Therefore, for a configuration parameter p_i in multi-model database, its first-order sensitivity is expressed as the ratio of the variance of the feature to the total variance,the calculation is shown in (1).

$$S_i = \frac{var_{p_i}(Y)}{var(Y)} \tag{1}$$

In order to obtain the relationship between the database parameters, it can also be obtained by calculating the higher-order sensitivity. The calculation method is as (2). Among them, $S_{p_1,p_2,...,p_k}$ is the k-order sensitivity. In multi-model database, in addition to a single configuration parameter that affects performance, the relationship between other parameters is mostly expressed as two parameters working together to bring changes to database performance.

$$S_{p_1,p_2,...,p_k} = \frac{var_{p_1,p_2,...,p_k}(Y)}{var(Y)} \tag{2}$$

Therefore, MMDTune focuses on the first-order and second-order sensitivity indicators of each parameter. In each round, the first-order sensitivity corresponding to each parameter is sorted from high to low. When the variance from top-k to top-$(k+1)$ decreases significantly (i.e., $S_{top-k} - S_{top-(k+1)} > S_{top-(k+1)}$), we choose the parameter with high

sensitivity as the key parameter. In addition, when the second-order sensitivity between the two parameters is greater than 0.5, they are also included in the selection range.

3.2 Multi-model database parameter tuning algorithm based on deep reinforcement learning

To simulate the try-and-error method that the DBAs adopt and overcome the shortcoming caused by regression, we introduce reinforcement learning which originates from the method of try-and-error in animal learning psychology and is a key technology to solve NP-hard problems of database tuning in continuous space (Zhang et al., 2019). Therefore, it is a reasonable choice to combine deep reinforcement learning to find the reasonable knobs for multi-model database.

After determining the parameters to be adjusted, MMDTune will adjust the value of each parameter based on the idea of TD3 algorithm. As shown in Fig. 2, the algorithm is mainly composed of environment and agent. The detailed Actor-Critic network and parameters of TD3 are shown in the Table 2.

Among them, the environment is a multi-model database cluster, which constantly interacts with the agent to provide quantifiable internal state and network training data. The agent is composed of Actor and Critic, which are two independent deep neural networks. The task of the Actor network is to map the observed internal state of the multi-model database to a set of parameters to maximize the cumulative reward, that is, it takes the internal state of the multi-model database as an input and can output a vector composed of parameter values. The Critic network takes the internal state and configuration parameters of the multi-model database as input and outputs a Q value that reflect whether or not the action output by Actor is valid. The agent in the initial stage is a model without knowledge, it learns through a series of fine-tuning actions. As it becomes more experienced in configuration parameters

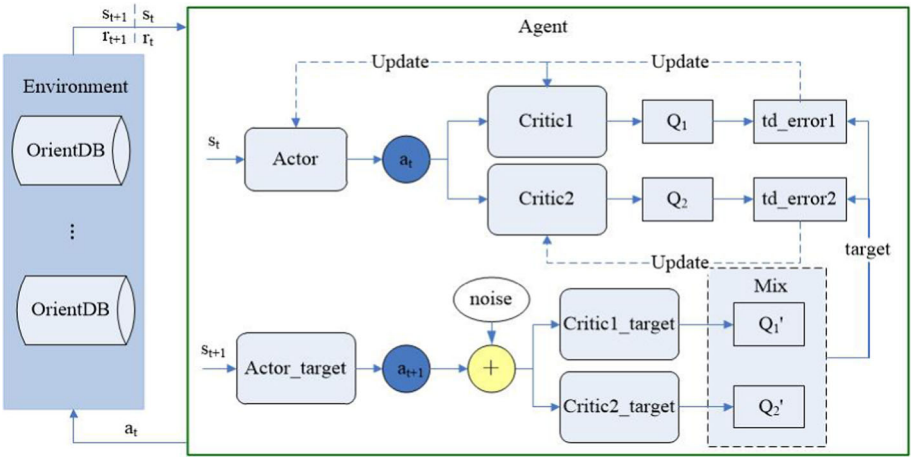

Fig. 2 The framework of the neural network

Table 2 Detailed Actor-Critic network and parameters of TD3

Step	Actor		Critic	
	Layer	Param	Layer	Param
1	Input	#States	Input	#Knobs + #States
2	Full Connection	128	Parallel Full Connection	128 + 128
3	LeakyReLU	0.2	Full Connection	256
4	BatchNorm	128	LeakyReLu	0.2
5	Full Connection	128	BatchNorm	256
6	Tanh	–	Full Connection	256
7	Dropout	0.3	Full Connection	64
8	Full Connection	128	Tanh	–
9	Tanh	–	Dropout	0.3
10	Full Connection	64	BatchNorm	64
11	Output	#Knobs	Output	1

and performance, it will recommend the optimal configuration parameters for a multi-model database. The detailed description of each part is as follows:

1) State space S: The internal state obtained when the multi-model database cluster finishes executing the workload is s, that is, counter information. For example, in OrientDB, counter information includes global information at the server level and count information at the session level.
2) Action space A: Suppose, after multi-round sensitivity analysis, the selected set of parameters is $P = \{p_1, p_2, \ldots, p_m\}$, where m is the number of key parameters. So, action a is a set of these parameter values, expressed as $\{c_1, c_2, \ldots, c_m\}$, where c_i is the value of parameter p_i.
3) Reward function R: For parameter tuning problems in a multi-model database, the reward is used to reflect the performance changes before and after the new knobs configuration recommended by MMDTune. So the reward needs to consider three aspects: a) it can provide valuable feedback on the performance of systems; b) it can provide accurate evaluation of knobs tuning with the maximum probability for the RL network; c) multiple metrics of the system performance can dynamically assign different weighting matrix to indicate different importance. Formally, the performance index of the database is expressed as $M = \{m_1, m_2, m_3\}$, where m_i corresponds to throughput, CPU and memory utilization respectively. It is assumed that the measured index values at time t are $Y_t = \{y_{t,1}, y_{t,2}, y_{t,3}\}$. Here, $y_{t,i}$ corresponds to the value of m_i. In particular, $y_{0,i}$ is the index value in the default configuration. Since in a multi-model database environment, for a given workload, the system must pay some cost to execute it, $y_{t,i}$ is always positive. In order to make the difference between positive and negative rewards to distinguish good or bad actions, the specific calculation process of rewards is as follows.

The essence of the optimization problem is to find the configuration parameters that make the throughput of the multi-model database as high as possible and the resource utilization as low as possible. Firstly, we calculate the changes in the initial and last time

based on performance metric. The external metrics of the initial time are $y_{0,i}$, and the external metrics of the last time are $y_{t-1,i}$, so the performance difference between the current moment and the initial time, and between the current moment and the previous moment is calculated according to (3) and (4) respectively.

$$\Delta_{t,0} = \begin{cases} \frac{y_{t,i}-y_{0,i}}{y_{0,i}}, & m_i \text{ is throughput} \\ \frac{y_{0,i}-y_{t,i}}{y_{0,i}}, & m_i \text{ is resource utilization} \end{cases} \tag{3}$$

$$\Delta_{t-1,t} = \begin{cases} \frac{y_{t,i}-y_{t-1,i}}{y_{t-1,i}}, & m_i \text{ is throughput} \\ \frac{y_{t-1,i}-y_{t,i}}{y_{t-1,i}}, & m_i \text{ is resource utilization} \end{cases} \tag{4}$$

We combine these two index differences into (5) to get the reward of m_i.

$$\text{reward}_{m_i} = \begin{cases} ((1+\Delta_{t-1,t})^2-1)(1+\Delta_{t,0}), & \Delta_{t-1,t} \geq 0, \Delta_{t,0} \geq 0 \\ 0, & \Delta_{t-1,t} < 0, \Delta_{t,0} \geq 0 \\ ((1-\Delta_{t-1,t})^2-1)(\Delta_{t,0}-1), & \Delta_{t,0} < 0 \end{cases} \tag{5}$$

There are three cases. Firstly, the reward will be negative if the current tuning system recommends knobs to the system with worse performance than the default knobs. Secondly, the reward will be positive if the current tuning system recommends knobs to the system with better performance than all the knobs previously recommended. Finally, if the knobs recommended by the current tuning system is better than the default, but not as good as the historically optimal knobs, the reward is 0.

Let us note that different tuning tasks may choose different tuning metrics (type or quantity), for example, throughput and latency can be tuned at the same time. Therefore, the tuning system in this paper assigns a weight coefficient w_i to the tuning indicator to indicate the direction of tuning preference, so that the tuning system can simultaneously tune multiple indicators. Then, the final total reward can be expressed as (6) below:

$$r = \sum_i \text{reward}_{m_i} * \omega_i \tag{6}$$

If the goal of the optimization is throughput, our reward function does not need to change, because the reward function is independent of changes in the hardware environment and workload and depends only on the optimization goal. Therefore, the reward function needs to be redesigned only when the optimization goal changes.

Algorithm 1 describes the specific flow of the tuning algorithm. To find the optimal strategy, we start with an arbitrary strategy μ. Before the iteration, the initial state of the multi-model database is needed, which is the internal state and external metrics of the multi-model database after the workload is executed in the default configuration. Unlike reinforcement learning in general, the multi-model database is configured so that its transition from one state to another is deterministic. Therefore, there is no need to re-measure at the beginning of each tuning cycle.

In addition, taking a random sample from the experience replay pool in a uniformly distributed manner leads to a low probability of obtaining useful data, leading to some meaningless iterations. Therefore, the tuning algorithm combines the prioritized

Initialize replay buffer R
if isExist(model) **then**
    model.load()
else
    Initialize actor network μ and critic network Q with weights θ^{μ} and θ^{Q}
    $\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'_1} \leftarrow \theta^{Q}$, $\theta^{Q'_2} \leftarrow \theta^{Q}$
    Initialize $s_0 \leftarrow \text{cost}(C_d, q)$
    for epoch $= 1, 2, \ldots, M$ **do**
        for $t = 1, 2, \ldots, T$ **do**
            $a_t \leftarrow \mu(s_t)$
            $C_t \leftarrow$ create_knobs(a_t)
            Configure multi-model database with C_t
            Perform workload q and observe new state $s_{t+1} \leftarrow \text{cost}(C_t, q)$ and $r_t \leftarrow \text{reward}(s_{t+1})$
            Push (s_t, s_{t+1}, a_t, r_t) into R
            Sample a random mini-batch (s_i, s_{i+1}, a_i, r_i) from R
            target $\leftarrow r_i + \gamma \min(Q'_1(s_{i+1}, \mu'(s_{i+1}) + \epsilon), Q'_2(s_{i+1}, \mu'(s_{i+1}) + \epsilon)$
            Update Critics
            $\theta^{Q_{m=1,2}} \leftarrow \arg\min_{\theta^{Q_m}} \frac{1}{N} \sum_i (\text{target} - Q_m(s_i, a_i))^2$
            if t mod d **then**
                Update μ by $\frac{1}{N} \sum_i \nabla Q_1(s_i, \mu(s_i)) \nabla \mu(s_i)$
                $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$,
                $\theta^{Q'_m} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'_m}$
            end if
            $s_t \leftarrow s_{t+1}$
        end for
        $P \leftarrow a_T$
    end for
end if

Algorithm 1 Parameter tuning algorithm based on TD3.

experience replay to train Actor and Critic, where Actor updates the weight of its neural network according to Q value, and uses deterministic strategy gradient iteration to calculate the optimal strategy. Critic updates the weight of its neural network based on the reward value.

The traditional TD3 algorithm always targets the minimum between two estimates when updating the Critic network. This update rule does not introduce any additional overestimation risk as traditional Q-Learning does, but it can also lead to underestimation bias. While underestimation does not spread during the learning process, it can have some negative performance effects. Therefore, in order to reduce overestimation while minimizing the negative effects of underestimation, the tuning algorithm uses a positive parameter α ($\alpha < 1$) to mix the minimum and maximum output of the two Critic target networks to update the target, rather than just using the minimum Q value.

$$Q(s_{i+1}, a_{i+1}) = \alpha \min_{m=1,2} Q'_m(s_{i+1}, a_{i+1}) + (-\alpha) \min_{m=1,2} Q'_m(s_{i+1}, a_{i+1}) \tag{7}$$

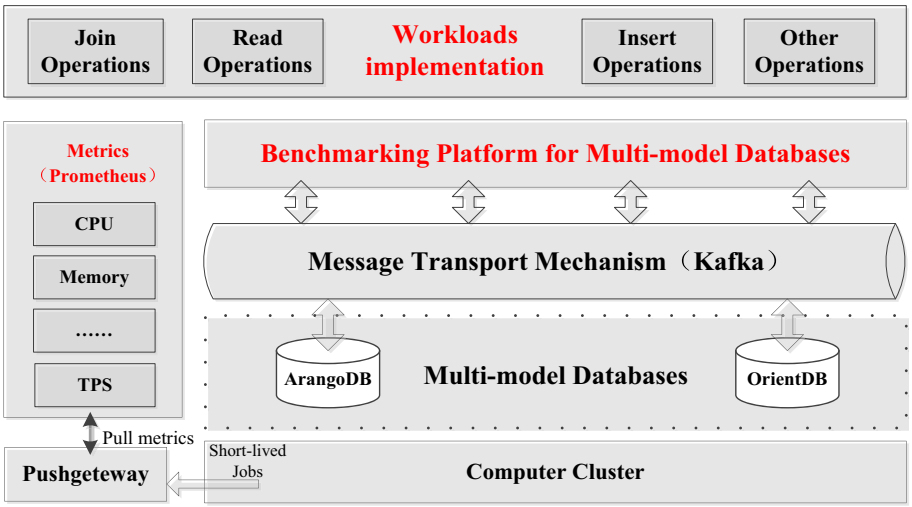

Fig. 3 The framework of the benchmarking platform

3.3 Benchmarking platform

To meet the requirements of performance monitoring in the process of multi-model database tuning, we propose a benchmarking platform for multi-model database and integrate it in MMDTune. As shown in Fig. 3, it adopts a multi-layers structure, which is mainly divided into five parts: infrastructure layer, data storage layer, message transmission layer, workload implementation layer and metrics collection layer.

For infrastructure layer, in essence, it is a computer cluster or cloud computing environment, which provides hardware foundation or virtual machine running environment for multi-model database.

Data storage layer consists of various NoSQL stores. Two well-known multi-model databases, ArangoDB and OrientDB, have been integrated with MMDTune.

In order to benchmark the different databases fairly, the core is the messaging transmission layer. To simulate the real situation of streaming data transmission, the messaging mechanism used is Apache Kafka (Dunning & Friedman, 2016). MMDTune uses Kafka to interact with a variety of multi-model databases for a variety of workload operations. When a user needs to extend a new multi-model database in the platform, the approach is to implement the corresponding services according to the standard interface approach.

In workload implementation layer, the most important aspect is to evaluate a multi-model database as comprehensively as possible. More specifically, it provides four parameters for generating workloads: multi-model database operations, how data requests are distributed, number of threads, and operands, enabling dynamic generation of variable workloads. As shown in Table 3, the platform implements the following multi-model database operations for generating workloads:

1. Inserting operation. It writes data of different models to the database, including documents, graphs, and key/values.
2. Joining query across models. The feature of multi-model database is that it can manage multiple data models at the same time, so join query is the most important function of

Table 3 Multi-model database workloads

Label	Operation
I1 ∼ I3	Insert operations for different data models: I1: insert a new JSON I2: insert a new graph I3: insert a new key/value
Q1	(1) Document query
	(2) Embedded array operation for JSON
Q2	Shortest path query
Q3	(1) Join data from JSON and graph
	(2) Embedded array operation for JSON
Q4	(1) Join data from JSON and key/value
	(2) Composited-key lookup for key/value
Q5	(1) Join data from graph and key/value
	(2) Fuzzy query
Q6	Join data from JSON, graph, and key/value
Q7	(1) Group the graph data
	(2) Find the maximum value in the result set by combining the aggregation function
	(3) Join data from JSON and sub-graph

multi-model database. Querying in a single statement by joining different data models realizes its cross-model characteristics.

3. Shortest path query. Both ArangoDB and OrientDB provide a shortest path query statement that can directly retrieve all shortest paths between two nodes.
4. Aggregating query. This operation aggregates information from multiple records using the aggregation functions unique to the multi-model database.
5. Updating/deleting records. The platform implements more workloads, such as updating documents, deleting records, and so on.

In different application scenarios, data access always meets a certain distribution mode. For example, on news sites, the most recently published news are more likely to be searched and visited. On platforms like MicroBlog, the higher the traffic to an item, the easier it is to retrieve it, regardless of freshness. Therefore, in order to achieve the fidelity of simulated workloads, different data request distributions are introduced in the design of workloads, including Zipfian, Poisson, Uniform, and Latest. Each distribution pattern determines which records to retrieve or which data to insert into the database. In particular, Zipfian and Poisson attributed data are selected according to Zipfian's law and Poisson distribution respectively. Uniform means to read data with equal probability. Finally, in the Latest distribution, the probability of data being accessed is closely related to the order in which it is inserted, that is, the most recently inserted record becomes the most popular, while previously popular data becomes less popular.

In the indicator collection layer, Prometheus is selected as the fine-grained performance indicator mechanism. It allows us to obtain the resource consumption of the machine over a specific period of time through a simple expression. It uses carefully designed data structures and algorithms to achieve very low per-node overhead and high concurrency, so that it has little impact on the machine. Therefore, this study used a series of functions provided by

Table 4 Monitoring metrics

Metric	Function
CPU	100 − (avg by (instance) (irate (node_cpu {instance = "xxx", mode = "idle"} [5s])) ∗ 100)
Memory	((node_memory_MemTotal_bytes − node_memory_Buffers_bytes − node_memory_Cached_bytes − node_memory_MemFree_bytes − node_memory_Slab_bytes) / node_memory_MemTotal_bytes) ∗ 100
Throughput	n/t (the number of operations executed by the database is n and the execution time is t)

Prometheus to obtain the desired measurements indirectly. Table 4 lists the corresponding calculations for CPU and memory.

Existing benchmarking tools, such as YCSB (Cooper et al., 2010; Matallah et al., 2017), provide throughput and other metrics. Throughput reflects the number of operations processed by the database system in a fixed amount of time. This performance metric is also added to the platform. Table 4 also lists the calculation methods of throughput.

4 Experiments and result analysis

To verify the effectiveness and adaptability of MMDTune, we take OrientDB as the specific research object and apply it to different experimental scenarios to carry out tuning experiments. MMDTune is similarly and easily applied to other multi-model databases. The performance changes of OrientDB are tested by setting different workloads, tunable parameters, optimization objectives and operating environments. Then, it is compared with the existing works, and the tuning effect of MMDTune is investigated through various experiments. Finally, the method is extended to ArangoDB for experiments to verify that the method can be effectively applied to other multi-model database tuning objects.

4.1 Experimental environment

The experimental environment consists of four Ali cloud servers, one of which is the client node, and the other three servers are used to build OrientDB cluster. Their hardware and software versions and configurations are completely consistent, as shown in Table 5.

Table 5 The experimental environment

Attribute	Information
CPU	Intel Xeon Platinum 8269@2.6GHz, 4 core
Memory	16GB
OS	CentOS 7.6
OrientDB version	3.1.3
ArangoDB version	3.7.3
Prometheus version	2.21.0

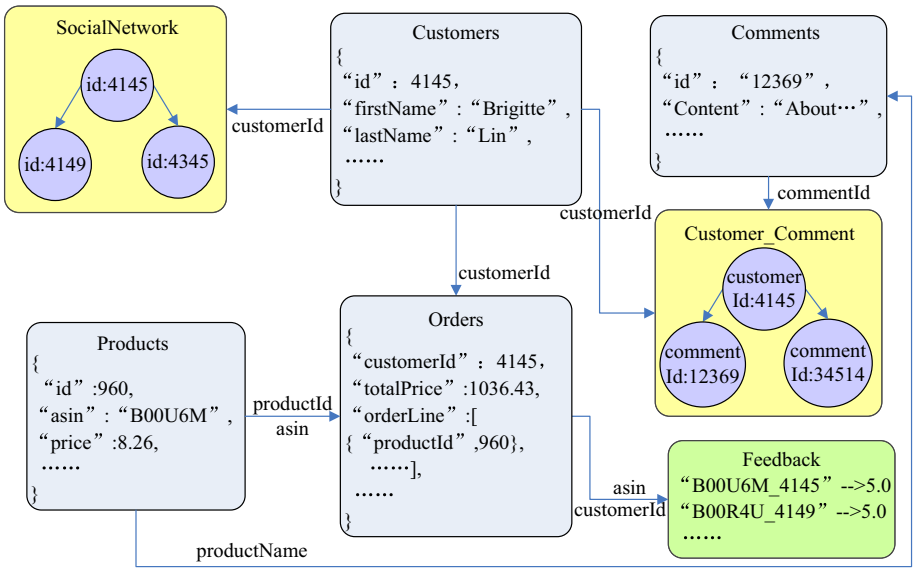

Fig. 4 An example of a multi-model dataset

4.2 Experimental dataset

To evaluate the performance of multi-model databases, we need to generate and use large-scale multi-model data.

MMDTune uses seed datasets from Unibench (Zhang et al., 2018), and generates large-scale multi-model data. It simulates a scenario combining social network with e-commerce, and contains four data models (key-value, document, graph, relationship) that can be supported by OrientDB and ArangoDB. Figure 4 shows an example of each entity and the relationships between them. The customer is the core of this dataset, and most other entities are related to it. For example, the relationships between customers form a social network, and the publishing relationships between customers and posts form another network. Orders are document-type data that contains an embedded array of the order row records within an order data. The product information in the order record together with the customer information forms the key to the feedback data and is used to indicate the customer's rating of the product purchased.

4.3 Important parameter identification experiment

This section starts with different tuning goals and uses multiple rounds of sensitivity analysis to identify critical parameters. Here, without loss of generality, we take the multi-model database operation Q1 as example (See Table 3). According to OrientDB's actual situation, there are some knobs that don't need to be considered, including those that are obviously not directly related to performance (such as pathnames) or those that are not allowed to be tuned (which can cause serious problems), so the experiment ended up with 187 adjustable sorted knobs. For each tuning target, 3 rounds of sensitivity analysis are performed separately.

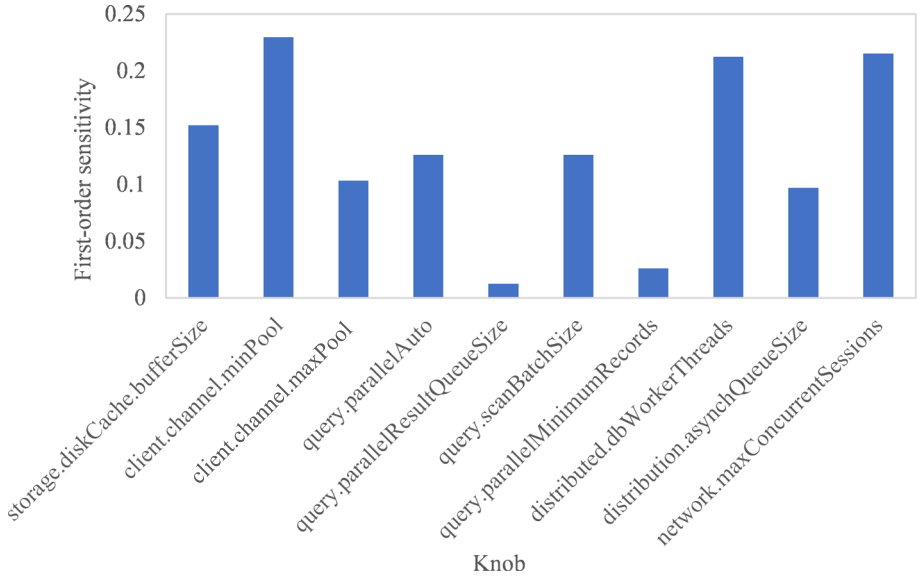

Fig. 5 Multi-round sensitivity related to throughput

4.3.1 Identification of important parameters related to throughput

Throughput means the number of operations the database can handle per second. For the throughput task, the parameters shown below are finally filtered, and Fig. 5 lists the first-order sensitivities of these parameters.

As shown in Fig. 5, the first-order sensitivities of "query.parallelMinimumRecords" and "query.parallelResultQueueSize" are not high, but the second-order sensitivities formed by them and query.parallelAuto are 0.726354 and 0.702862 respectively, so they are also included in the optional range. In addition, among these parameters, the database connection pool and the number of concurrent sessions have a greater impact on the throughput. This is because OrientDB needs to establish a communication session between the client and the server through a remote connection when executing the workload. At the same time, the workload in the experiment consists of the order of 100000 operations multi-model database operations and multi-threads. Increasing the number of concurrent sessions and the size of the connection pool causes establishing multiple database connections at the same time, thereby reducing the execution time. Creating too many sessions can also stress the system and slow down operations.

4.3.2 Identification of important parameters related to memory

Memory tuning is the process of determining optimal cache parameter values for the multi-model database OrientDB. For example, OrientDB with a large working set benefits from a large cache; a small cache incurs excess cost as data is swapped in and out. Conversely, OrientDB with a small working set benefits from a small cache; a large cache is a waste of resources due to the high cost of each fetch and unnecessary static power. Therefore, in this experiment, with memory as the tuning task, the important parameters shown in Fig. 6 are selected.

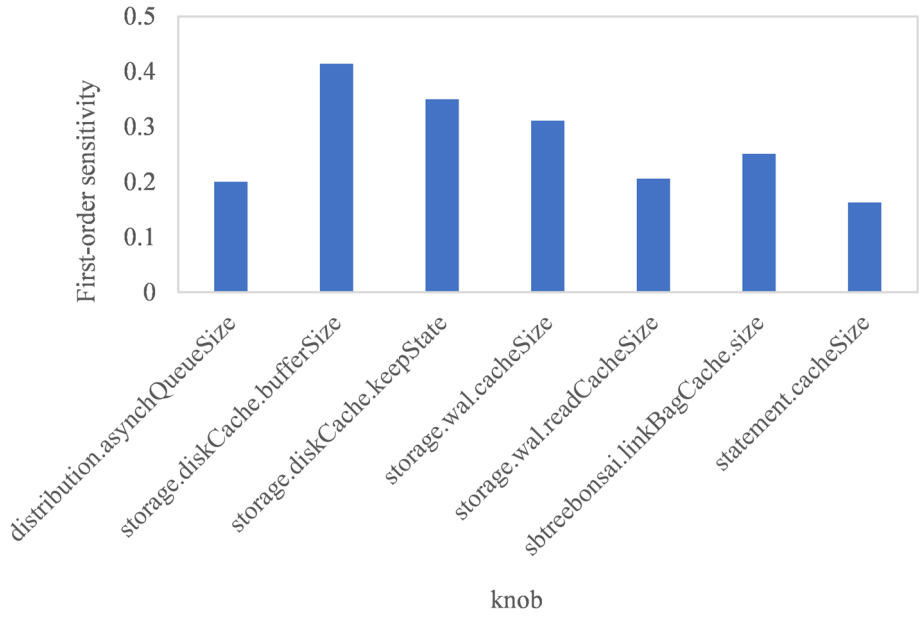

Fig. 6 Multi-round sensitivity related to memory

As can be seen from Fig. 6, there are some settings in OrientDB that can make it run on systems with limited resources, and they are mainly concentrated in the two areas of cache and log. Among them, "storage.diskCache.bufferSize" has the highest first-order sensitivity among all parameters, which means that it plays an important role in OrientDB's memory tuning.

4.3.3 Identification of important parameters related to the CPU

When the number of threads set in the workload is too high, OrientDB may take up to 100% of the CPU, which puts a serious burden on the system. If there are other running programs, it will cause multiple applications to compete for the CPU, resulting in extremely slow operation or even crash. Therefore, it is very important to try to reduce the CPU utilization of the storage system for applications with limited operating environment. The user cannot really reduce the CPU usage of OrientDB by modifying the configuration parameters, but one can reduce the number of threads running in parallel. This is also verified by the execution results of multiple rounds of sensitivity analysis. Table 6 lists the configuration parameters related to the CPU.

In the experiment, due to the small variance of the measured CPU utilization, the importance of each parameter is high. In theory, when "environment.concurrent" is set to false, the database will turn off its internal lock management, so that the multi-model database OrientDB executes in a single-threaded environment, which can reduce the CPU utilization of the system. The parameter "distributed.dbWorkerThreads" has an important impact on both throughput and CPU utilization of OrientDB. Analysing it, one can find that this parameter is mutually exclusive for different tuning goals. Increasing the number of parallel worker threads will inevitably increase the CPU utilization of the database and thus shorten the execution time. On the contrary, it will reduce the CPU used by the system and prolong

Table 6 Parameters related to CPU

Knob	Describe	Sensitivity
environment.concurrent	Specifies whether to run in a multithreaded environment	0.951842
distributed.localQueueSize	Size of the thread queue for distributed messages	0.739933
distributed.dbWorkerThreads	Number of parallel worker threads processing distributed messages per database	0.837039

the running time. Therefore, one cannot meet the demands of increasing throughput and reducing resource cost at the same time.

4.4 Tuning experiment and analysis

To verify that MMDTune can adapt to different tuning tasks, we use the workloads shown in Table 7. Five workloads involve typical multi-model database operations, including single-model read-only, cross-model join queries, and a combination of read-write operation, which can fully test the tuning effect under different workloads to avoid the contingency of tuning results.

To verify the effectiveness of the tuning algorithm and the multi-round sensitivity analysis method, the experiment uses the random method and the multi-round sensitivity analysis method to select the tunable parameters respectively, and performed tuning operations on different workloads with throughput as the tuning objective. At the same time, as the reinforcement learning algorithm explores the parameter space, and in order to find the optimal configuration more likely, random noise is added to the output of the model to enhance its exploration ability. In order to obtain the correlation between the number of tuning steps and tuning results, we observe the performance of the system in increments of five tuning steps. The experimental results are shown in Fig. 7.

Figure 7(a)∼(e) respectively show the optimal performance of different workloads after different number of tuned steps, where the horizontal coordinate represents the number of tuned steps and the vertical coordinate represents the maximum throughput after tuning with MMDTune. When the number of steps is 0, it indicates the performance of OrientDB under the default configuration. Curve If means using the multi-round sensitivity analysis method to select adjustable parameters, and Curve RC means adjusting the knobs of random selection.

Firstly, for all workloads, OrientDB performs better than the default configuration after five tuning steps, indicating that the tuning algorithm can learn from past experience and

Table 7 Workloads

Label	Operation	Distribution	Thread	Operand
W1	Q1	Uniform	10	100000
W2	Q4	Poisson	10	100000
W3	30%I1 + 70%Q6	Latest	10	100000
W4	60%Q1 + 40%I1	Uniform	20	100000
W5	50%I2 + 50%Q7	Uniform	20	100000

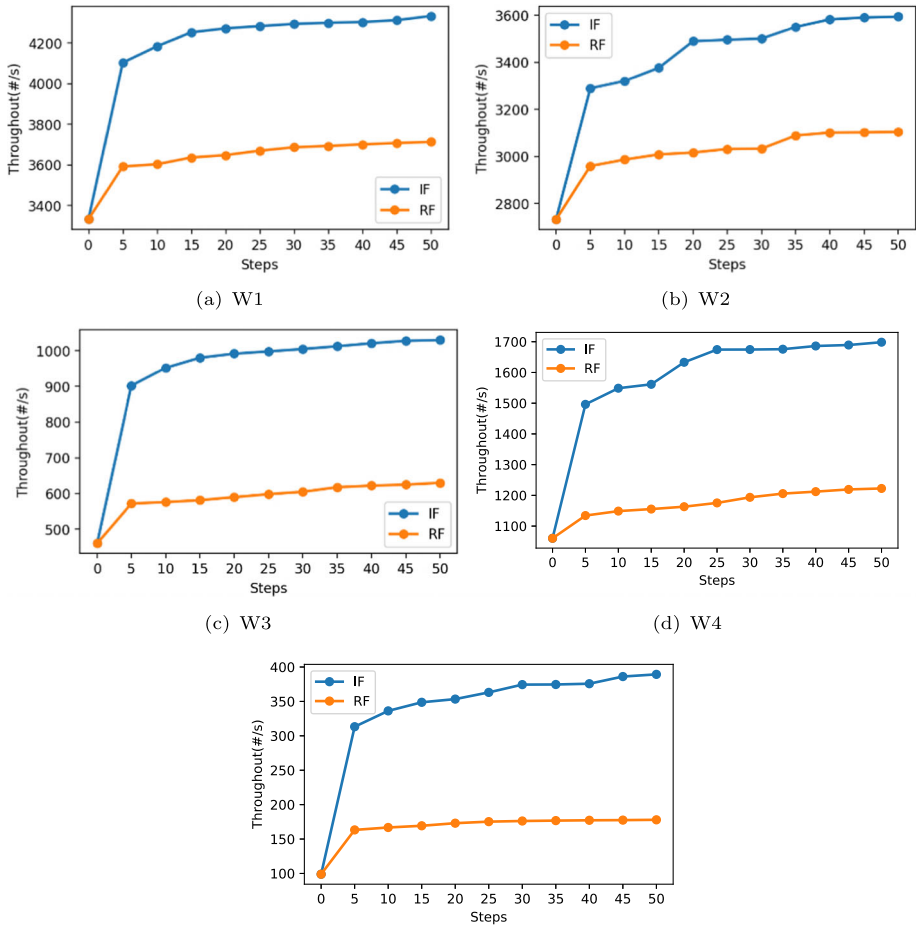

Fig. 7 Comparison of TPS with different tuning steps

achieve high efficiency. As the number of tuning times increases, the agent can fine-tune action to gradually adapt to the current workload, thereby continuously improving the system throughput. Secondly, if we accept longer tuning times, users will get better configuration to achieve higher performance. However, as the number of tuning steps continues to increase, the throughput gain does not gradually increase, but tends to stabilize. This is because the prioritized experience replay method is used in the algorithm, which leads to the fast convergence speed of the algorithm. Thirdly, by comparing the two curves of If and RC in Fig. 7, the two parameter selection strategies have the same performance trend and can achieve better performance, but the multi-round sensitivity analysis method has much higher performance gain than the random method. For example, by looking at the If curve in Fig. 7(a), it can be found that the throughput of OrientDB increased by 30% when the tunable parameter was selected using multi-round sensitivity analysis, while the throughput of OrientDB increased by only 11.42% when the random knob was adjusted. This is because the former needs to extract the parameters with high correlation, which can help the

learning of the neural network very effectively. In contrast, randomly selected parameters may have little effect on the performance of OrientDB, or combine with other parameters to create complex relationships between database performance. As a result, it can lead to longer study times and less noticeable improvement in grades. According to the above conclusions, the validity of multi-round sensitivity analysis is verified, and it can truly select the parameters related to performance.

Assume that the running environment of the application system is a server with a small amount of kernel and memory. Then, users will want to limit the resource utilization of the database system. By default, OrientDB will try to use as much memory as possible, which can easily cause the database to fail or even crash. Here, W3 is used as the workload to perform tuning, which includes both the read-write operation of the database and the connection query of the multi-model database, which can effectively avoid the particularity of tuning.

Figure 8 describes the changes of resource utilization after different training steps. Similar to the throughput task, the resource degradation of the system tends to be stable as the number of tuning times increases. Compared to database performance in standard configuration, memory utilization of OrientDB decreased by 36.88%, while CPU utilization was reduced by two-thirds. Thus it can be seen that MMDTune can effectively act on the combination of metrics. This is due to the reward function in reinforcement learning, which fully considers the importance of each metric and makes the tuning results meet the target requirements as far as possible.

In a cloud environment, different users have their own database memory size and disk capacity. As models migrate to different hardware environments, the knowledge about disk size, memory size, and computing power needs to be updated. Therefore, it is a challenge to adapt to the new hardware environment. To verify that MMDTune can greatly optimize the performance of the multi-model database for different hardware configurations, this experiment optimizes OrientDB under different hardware environments.

Table 8 shows the two operating environments used in the experiment. Table 9 lists the tuning results of OrientDB running different workloads in both hardware environments. Looking at both configurations, instance B is better than instance A in every respect. Therefore, the performance of the database on instance B is better. As can be seen from Table 9, MMDTune performs better when instance B is used as the running environment. For example, for workload W3, in environment B, the database throughput increased by 71.22%.

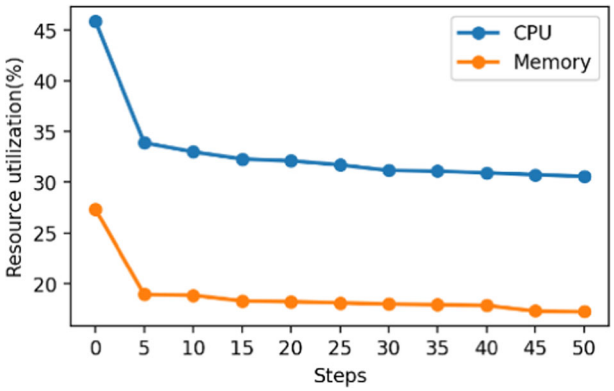

Fig. 8 Comparison of resource utilization with different tuning steps

Table 8 Environment configurations

Instance	SSD (G)	Memory (G)	CPU
A	50	32	8
B	50	48	12

In environment A, the increase was only 65.08%. This is because the policy network has a lot of configuration space to explore in the running environment B, and there is more room for performance improvement. In general, the experimental results have verified that MMDTune can adapt to different hardware environments, and a better configuration is recommended.

To verify the efficiency of MMDTune, the online tuning efficiency of OtterTune and CDBTune are compared. Since CDBTune does not process database parameters, the method of multi-round sensitivity analysis is adopted in the experiment to select adjustable parameters. In addition, because the benchmark tools cannot generate workloads suitable for multi-model databases, our proposed multi-model database benchmark platform is integrated into the system to support the tuning of OrientDB.

By blindly reducing the cache configuration and the number of working threads, we can limit the memory consumption and reduce the CPU utilization, but this will sacrifice the execution speed of the system. This is not reasonable in modern application system. Users hope that the multi-model database can not only meet certain throughput requirements, but also make use of system resources as little as possible. In most cases, only when the multi-model database system makes full use of CPU, can it achieve faster execution speed, and cannot meet the requirements of high throughput and low CPU utilization at the same time. Therefore, the throughput and memory utilization are set as the tuning goals. When the rewards stabilize, tuning stops. The final experimental results are shown in Fig. 9 below. Among them, OtterTune uses 1000 pieces of tuning data stored in the MMDTune tuning process as training data.

According to Fig. 9, all tuning methods achieve better performance than the default configuration. MMDTune has the best tuning effect. For workload W1, the throughput obtained using MMDTune is 7.35% higher than CDBTune, and the memory utilization is reduced by 11.54%. The reasons are as follows: firstly, CDBTune adapts DDPG algorithm to recommend configuration, which has the problem of overestimation when updating the network, resulting in cumulative error. On the basis of TD3 algorithm, it effectively alleviates the above problems by mixing two Critic network estimates. Secondly, CDBTune relies on the policy network every time it selects parameters. However, at the initial stage of training, the

Table 9 Comparison of TPS in different environments

Label	Instance A		Instance B	
	Before tuning	After tuning	Before tuning	After tuning
W1	5018.03	6195.41	6077.64	7561.18
W2	4016.52	4916.53	5172.37	6182.51
W3	801.01	1322.37	955.36	1635.80
W4	1908.45	2956.89	2279.53	3548.37
W5	178.02	521.31	212.63	790.48

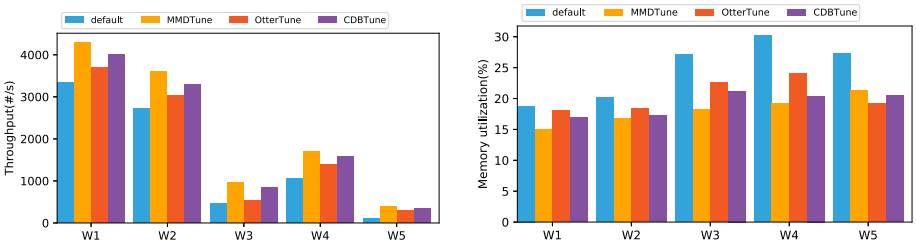

Fig. 9 Experimental results of different tuning methods

policy network can only recommend parameter values according to the local optimal solution, thus limiting the exploration space of CDBTune. So the tuning effect of CDBTune is worse than that of MMDTune. But because CDBTune can effectively learn from past experience, it has better tuning capabilities than OtterTune.

Of all the methods, OtterTune achieves the lowest performance gain. Similarly, for workload W1, the database throughput is increased by 16.21% with MMDTune compared to OtterTune, and the memory utilization is reduced by 16.55%. This is because OtterTune uses Gaussian processes to map configurations. Although it can learn from history, this regression model is still too simple to explore new knowledge to refine itself compared with neural network, so the performance gain is very limited. The deep reinforcement learning method enables the neural network to simulate the human brain, learn in the direction of optimization, and recommend reasonable parameter settings corresponding to the current workload and hardware environment.

Different multi-model databases have different system parameters, including different meanings, types, names and value ranges. To verify that MMDTune can be effectively applied to different multi-model databases, ArangoDB is also used to verify the performance of MMDTune. To avoid the contingency of tuning results, throughput, memory and CPU utilization are taken as optimization objectives, and W3 is used as workload. The MMDTune is compared with the tuning results of OtterTune and CDBTune, and the experimental results are shown in Tables 10 and 11.

From Table 10, when workload W3 is executed, ArangoDB's throughput increases by 126.15% with MMDTune compared to the default configuration, and from Table 11, memory utilization and CPU utilization decrease by 41.57% and 31.33%, respectively. Because it is not limited by the throughput, the memory and CPU utilization of ArangoDB are significantly reduced by the three tuning methods. This is because ArangoDB provides a large number of parameters for modifying buffers and worker threads, so the algorithm has a good chance to adjust resource utilization. By comparing the tuning effects of different methods in the table, it can be seen that MMDTune always achieve better performance. In summary, MMDTune can efficiently adapt to different multi-model database systems while maintaining relatively good performance.

Table 10 Tuning results of different tuning methods with throughput as the target

Metrics	Default	MMDTune	OtterTune	CDBTune
TPS	302.389	683.879	487.785	632.892

Table 11 Tuning results of different tuning methods with resource utilization as the target

Metrics	Default	MMDTune	OtterTune	CDBTune
Memory utilization (%)	17.343	10.132	11.325	11.241
CPU utilization (%)	59.849	41.094	41.213	41.908

5 Conclusion

As more and more applications are proposed to deal with multi-model data, the task of managing and tuning multi-model databases becomes very important. In practice, the research of multi-model database tuning based on deep reinforcement learning is not only helpful to adjust parameter values according to different tuning objectives and workloads, but also helpful to set the best parameter configuration for the database under different hardware environments to improve the stability and reliability of the system. To solve the problem of parameter selection and adjustment in the process of tuning, we propose a parameter tuning method MMDTune for multi-model database, which can recommend excellent configuration scheme in complex environment. It uses multi-round sensitivity analysis and improved TD3 algorithm to improve the tuning results of the database. At the same time, it uses the benchmarking platform of the multi-model database to generate the workload and collect the performance metrics to meet the performance monitoring requirement during the tuning process. We modify workloads, optimizing targets, and running environments to carry out the tuning experiment, and the results showed that MMDTune has a strong adaptability regardless of how the tuning tasks change.

Acknowledgements This work was partly supported by National Key R&D Program of China (2019YFE0109900); Fundamental Research Funds for the Central Universities, 2018 (B200202185); Jiangsu Province Key Research and Development Program (Modern Agriculture) Project under Grant no. BE2018301, 2017; Jiangsu Province Postdoctoral Research Funding Project under Grant no. 1701020C, 2017; and Six Talent Peaks Endorsement Project of Jiangsu under Grant no. XYDXX-078.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest. Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

References

Basu, D., Lin, Q., Chen, W., & et al. (2016). Regularized cost-model oblivious database tuning with reinforcement learning. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXVIII.* (pp. 96–132). Berlin, Heidelberg.

Borgonovo, E., & Plischke, E. (2016). Sensitivity analysis: a review of recent advances. *European Journal of Operational Research*, *248*(3), 869–887. https://doi.org/10.1016/j.ejor.2015.06.032.

Braun, G., Fillottrani, P. R., & Keet, C. M. (2022). A framework for interoperability between models with hybrid tools. Journal of Intelligent Information Systems. https://doi.org/10.1007/s10844-022-00731-7.

Cooper, B. F., Silberstein, A., Tam, E., & et al. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing, ACM SIGMOD.* (pp. 143–154). https://doi.org/10.1145/1807128.1807152.

Dankwa, S., & Zheng, W. (2019). Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing, Association for Computing Machinery.* (pp. 1–5). https://doi.org/10.1145/3387168.3387199.

Davoudian, A., Chen, L., & Liu, M. (2018). A survey on nosql stores. *ACM Computing Surveys*, *51*(2), 1–43. https://doi.org/10.1145/3158661.

Duan, S., Thummala, V., & Babu, S. (2009). Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment*, *2*(1), 1246–1257. https://doi.org/10.14778/1687627.1687767.

Dunning, T., & Friedman, E. (2016). *Streaming architecture: new designs using apache kafka and mapr streams*. Sebastopol, CA: O'Reilly Media, Sebastopol.

Fekry, A., Carata, L., Pasquier, T., & et al. (2020). To tune or not to tune? in search of optimal configurations for data analytics. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM SIGKDD.* (pp. 2494–2504). https://doi.org/10.1145/3394486.3403299.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *Proceedings of 35th International Conference on Machine Learning, ICML 2018, IMLS.* (pp. 2587–2601).

Gordon-Ross, A., & Vahid, F. (2007). A self-tuning configurable cache. In *Proceedings of 44th ACM/IEEE Design Automation Conference.* (pp. 234–237). https://doi.org/10.1109/DAC.2007.375159.

Huang, X., Wang, J., Yu, P. S., & et al. (2017). An experimental study on tuning the consistency of nosql systems. *Concurrency and Computation: Practice and Experience*, *29*(12), e4129. https://doi.org/10.1002/cpe.4129.

Li, G., Zhou, X., Li, S., & et al. (2019). Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment*, *12*(12), 2118–2130. https://doi.org/10.14778/3352063.3352129.

Lu, J., & Holubová, I. (2019). Multi-model databases: a new journey to handle the variety of data. *ACM Computing Surveys (CSUR)*, *52*(3), 1–38. https://doi.org/10.1145/3323214.

Lu, J., Liu, Z. H., Xu, P., & et al. (2018). Udbms: road to unification for multi-model data management. In *International Conference on Conceptual Modeling, Springer,* (pp. 285–294). https://doi.org/10.1007/978-3-030-01391-233.

Matallah, H., Belalem, G., & Bouamrane, K. (2017). Experimental comparative study of nosql databases: Hbase versus mongodb by ycsb. *Comput. Syst. Sci. Eng*, *32*(4), 307–317.

Płuciennik, E., & Zgorzałek, K. (2017). The multi-model databases–a review. In *Proceedings of 13th International Conference on Beyond Databases, Architectures and Structures.* (pp. 141–152). https://doi.org/10.1007/978-3-319-58274-0 12.

PN, S., Sivakumar, A., Rao, S., & et al. (2013). D-tunes: self tuning datastores for geo-distributed interactive applications. *ACM SIGCOMM Computer Communication Review*, *43*(4), 483–484. https://doi.org/10.1145/2534169.2491684.

Prometheus Team (2022). Prometheus. https://prometheus.io/, Accessed 22 July 2022.

Sawadogo, P., & Darmont, J. (2021). On data lake architectures and metadata management. *Journal of Intelligent Information Systems*, *56*(1), 97–120. https://doi.org/10.1007/s10844-020-00608-7.

Schaul, T., Quan, J., Antonoglou, I., & et al. (2015). Prioritized experience replay. arXiv:1511.05952.

Sobol, I. M. (2001). Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and computers in simulation*, *55*(1-3), 271–280. https://doi.org/10.1016/10.1016/S0378-4754(00)00270-6.

Storm, A. J., Garcia-Arellano, C., Lightstone, S. S., & et al. (2006). Adaptive self-tuning memory in db2. In *Proceedings of the 32nd international conference on Very large data bases, VLDB.* (pp. 1081–1092). https://doi.org/10.5555/1182635.1164220.

Tian, W., Martin, P., & Powley, W. (2003). Techniques for automatically sizing multiple buffer pools in db2. In *Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative Research, CASCON'03.* (pp. 294–302). https://doi.org/10.5555/961322.961367.

Tran, D. N., Huynh, P. C., Tay, Y. C., & et al. (2008). A new approach to dynamic self-tuning of database buffers. *ACM Transactions on Storage (TOS)*, *4*(1), 1–25. https://doi.org/10.1145/1353452.1353455.

Van Aken, D., Pavlo, A., Gordon, G. J., & et al. (2017). Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data, ACM SIGMOD.* (pp. 1009–1024). https://doi.org/10.1145/3035918.3064029.

Van Aken, D., Yang, D., Brillard, S., & et al. (2021). An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *Proceedings of the VLDB Endowment*, *14*(7), 1241–1253. https://doi.org/10.14778/3450980.3450992.

Wei, Z., Ding, Z., & Hu, J. (2014). Self-tuning performance of database systems based on fuzzy rules. In *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD).* (pp. 194–198) IEEE. https://doi.org/10. 1109/FSKD.2014.6980831.

Wu, J., Wang, R., Li, R., & et al. (2018). Multi-critic ddpg method and double experience replay. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 165–171) IEEE. https://doi.org/10.1109/SMC.2018. 00039.

Zadeh, F. K., Nossent, J., Sarrazin, F., & et al. (2017). Comparison of variance-based and moment-independent global sensitivity analysis approaches by application to the swat model. *Environmental Modelling & Software*, *91*, 210–222. https://doi.org/10.1016/j.envsoft.2017.02.001.

Zhang, C., Lu, J., Xu, P., & et al. (2018). Unibench: A benchmark for multi-model database management systems. In *Technology Conference on Performance Evaluation and Benchmarking.* (pp. 7–23) Springer Verlag. https://doi.org/10.1007/978-3-030-11404-6 2.

Zhang, J., Liu, Y., Zhou, K., & et al. (2019). 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). https://doi.org/10.1145/3299869.3300085.

Zhang, J., Zhou, K., Li, G., & et al. (2021). Cdbtune+: An efficient deep reinforcement learning-based automatic cloud database tuning system. *The VLDB Journal*, *30*, 1–29. https://doi.org/10.1007/s00778-021-00670-9.

Zhu, Y., Liu, J., Guo, M., & et al. (2017). Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing.* (pp. 338–350). https://doi.org/10.1145/3127479.3128605.