Journal of Database Management
Volume 35 • Issue 1

A Sample-Aware Database Tuning System With Deep Reinforcement Learning

Zhongliang Li, Nanjing University of Aeronautics and Astronautics, China
Yaofeng Tu, Nanjing University of Aeronautics and Astronautics, China
Zongmin Ma, Nanjing University of Aeronautics and Astronautics, China*
https://orcid.org/0000-0001-7780-6473

ABSTRACT

Based on the relationship between client load and overall system performance, the authors propose a sample-aware deep deterministic policy gradient model. Specifically, they improve sample quality by filtering out sample noise caused by the fluctuations of client load, which accelerates the model convergence speed of the intelligent tuning system and improves the tuning effect. Also, the hardware resources and client load consumed by the database in the working process are added to the model for training. This can enhance the performance characterization ability of the model and improve the recommended parameters of the algorithm. Meanwhile, they propose an improved closed-loop distributed comprehensive training architecture of online and offline training to quickly obtain high-quality samples and improve the efficiency of parameter tuning. Experimental results show that the configuration parameters can make the performance of the database system better and shorten the tuning time.

KEYWORDS

DBTune Scheme, Intelligent Tuning, Machine Learning, SA-DDPG Model

INTRODUCTION

In recent years, there has been a notable and rapid development in information and communication technologies, with a significant emphasis on the domains of Cloud-Computing, Big-Data (Eachempati et al. (2022)), Artificial intelligence (Wang et al. (2019)), and the advent of 5G networks. The continuous expansion of the volume of the data and the enrichment of data types have resulted in increasingly diverse and rapid changes in database workload. Conventional operational methods are no longer adequate to fulfill the requirements of modern database systems. According to Chen et al. (2019), Kraska et al. (2019) and Li, Zhou & Li (2019), With the recent advancement of artificial intelligence technology AI-based database operation and maintenance methods (M'barek et al. (2016)) are gradually replacing traditional database operation and maintenance methods.

Improving the performance of the database is a main concern of the intelligent operation and maintenance. The tuning of database generally refers to increasing the throughput per unit time of the

DOI: 10.4018/JDM.333519
*Corresponding Author

database or reducing the latency of a single database operation. There are a number of configurable parameters which is significantly important for the performance of the database operations. Oh and Sang (2005) and Weikum et al. (2002) showed in their work that database parameter tuning has always been a key concern of DBA. In recent years, the academic community has also made a significant improvement on database parameter tuning which impact on DBA performances (Aken, Pavlo, Gordon & Zhang, 2017; Cai et al., 2022; Cereda et al., 2021; Fekry et al., 2020; Gur et al., 2021; Ishihara & Shiba, 2020; Li, Zhou, Li & Gao, 2019; Kanellis et al., 2020; Kanellis et al., 2022; Zhang et al., 2019; Zhang et al., 2021; Zhang et al., 2022). However, researches have ignored the following three issues:

1. Database performance failures are rare and difficult to obtain in online environments. It is relatively easy to construct performance problem samples in an offline database environment. However, due to the differences between offline simulation environments and real online environments, models trained solely on sample data from offline simulation environments may perform poorly in online environments. The above two reasons require the combination of samples from both online and offline simulation environments to jointly train the model.
2. Both online and offline environments ignore the impact of hardware environment information on database tuning when generating samples. If hardware environment information is not included in the model training process, it cannot better characterize database performance characteristics, thereby affecting the accuracy of the model.
3. Not all samples in the database can be used to train the model. Some samples are invalid and need to be filtered those outliers. This part of the sample does not refer to manual input errors, null values, and other anomalies. The main factors affecting database performance are not only database parameters but also workload. When the workload is relatively low (performance has not reached the performance bottleneck), the performance of the entire database is only related to the strength of the workload, and is not related to the database parameters. Therefore, when the database system does not reach the performance bottleneck, the samples generated need to be filtered to avoid errors in evaluating the model's output parameters.

At present, the most cutting-edge intelligent parameter tuning is based on reinforcement learning (Li, Zhou, Li & Gao, 2019; Lillicrap *et al*., 2016; Zhang *et al*., 2019). The authors introduce a method for identifying whether the recommendation result is effective after configuring recommendation induced from the reinforcement learning, and proposes a new generation of database intelligent tuning system. DBtune used reinforcement learning to solve the problem of parameter tuning. The main work and contributions are as follows:

1. To solve the problem of sample scarcity, we proposed a distributed comprehensive training architecture with the Big-Small sample pool mechanism, which can supplement the high-quality samples filtered by the online environment to the comprehensive sample pool of the offline environment. Then the comprehensive sample data can be used to iteratively generate the basic model of the offline environment and send this basic model to the online environment for incremental update. This improved closed-loop system can improve the efficiency of parameter tuning.
2. The hardware resources and workload characteristics of the database operation are added to the model for training, which improves the versatility of the model. Users only need to fine-tune the target tuning environment to obtain high-quality recommended parameters. Therefore, the interaction cost of the tuning system is reduced, which is convenient for large-scale commercial applications.
3. According to the relationship between the workload and the overall system performance in the intelligent tuning process, a sample perception algorithm model based on reinforcement learning SA-DDPG (Sample-Aware-DDPG) is proposed. In the training phase, SA-DDPG improves the

accuracy of the model by filtering out sample noise caused by Client-Load fluctuations. In the actual prediction stage, the effectiveness of the tuning parameters is identified through the sample filter to filter out invalid parameters.

The remainder/remaining of this paper is organized as follows. First, we introduce the design of DBtune. Second, we propose the algorithm design of SA-DDPG, including methodology, algorithm design, database state parameter design and reward function design. Third, we present the test/simulation results and corresponding analysis. Subsequently, presents the research related to database intelligent parameter adjustment. We then demonstrate the usability of our method with a case study. Finally, we conclude this paper and prospects the research trend of database intelligent parameter adjustment.

DESIGN OF DBTUNE SCHEME

This paper proposes an end-to-end sample-aware database intelligent parameter tuning method, which can adapt the database configuration to workload and quickly adjust the database parameters to a better state, enabling the database to achieve better performance. The Big-Small sample mechanism adopted can accelerate sample collection speed, and reinforcement learning optimization based on the pre trained model can also accelerate the optimization speed, helping database operation and maintenance personnel quickly adjust database performance to the optimal level.

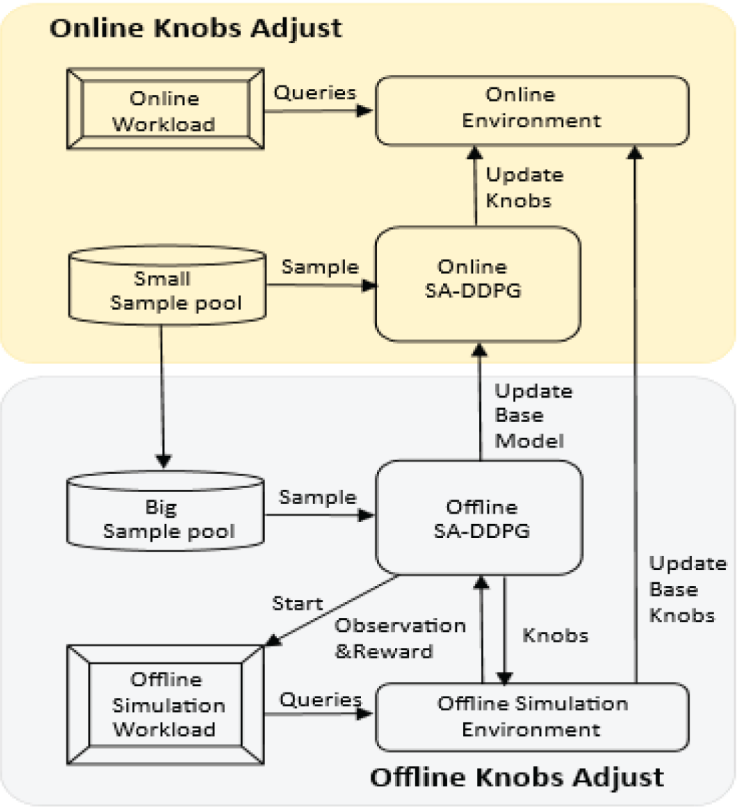
DBtune is a distributed comprehensive intelligent parameter tuning method that combines offline and online parameter tuning. The program framework is divided into two parts: Online Knobs Adjust and Offline Knobs Adjust, as shown in Figure 1.

The basic model and parameters generated by offline tuning will be regularly sequentially updated to the online tuning environment; the samples generated by the online tuning will be sent to the offline tuning environment, and a basic model with stronger generalization ability will be generated after retraining.

- **Offline Knobs Adjust:** Offline SA-DDPG module loads the base model initially, and then sends instructions to start the Offline Simulation workload module. The Offline Simulation workload module sends requests (Queries) to the offline simulation environment. The samples from the online environment and the local samples from the offline environment converge into a Big-Sample pool. The Offline SA-DDPG module uses the SA-DDPG algorithm to generate the recommended basic configuration based on the basic model and combined with the updated sample data. Finally, the Offline SA-DDPG module will update the base model to the Online Environment.
- **Online Knobs Adjust:** First, the Online Environment loads the basic configuration (base knobs) recommended by the Offline Knobs Adjust system, and starts to provide services. At the same time, the Online SA-DDPG module starts and loads the base model, which generated by the Offline Knobs Adjust system. The Online Environment sends status information and reward information to the online SA-DDPG module. The Online SA-DDPG module generates knobs according to the SA-DDPG algorithm and generates some sample data to append to the Small Sample pool at the same time. Finally, the online database environment will periodically update the online configuration (update knobs) based on availability.

Through the above design, on the one hand, the DBtune system uses historical sample data and small sample data generated in the online environment to form a Big-Sample pool. A general basic model can be trained by the Big-Sample pool and then fine-tuning on the basic model can greatly shorten the configuration parameter optimization time. On the other hand, we can get an improved

Figure 1. DBtune scheme framework



model that matches the actual business production environment by constantly fine-tuning the basic model. Subsequently, We can achieve a higher-performance configuration for the live network.

SA-DDPG MODEL

Necessity of Sample Filters

To study intelligent tuning, we need to consider a question first: whether adjusting parameters will definitely affect the performance of the overall database system? After the extensive research, the answer is no. When the client load pressure does not reach the server performance bottleneck, the overall system performance depends only on the client load and has nothing to do with the quality of the database parameters.

The overall performance of the database system depends on three factors: the client load pressure Workload, the server's hardware environment and the capacity of server processing Sever-Side Capabilities, as shown in Formula (1):

$$P = F\left(Workload, Hardware, Sever - Side - Capabilities\right) \tag{1}$$

Here, P is the overall performance of the database system; Workload is the client load, including the frequency of database operation, operation type and the size of the operation data; Hardware is the

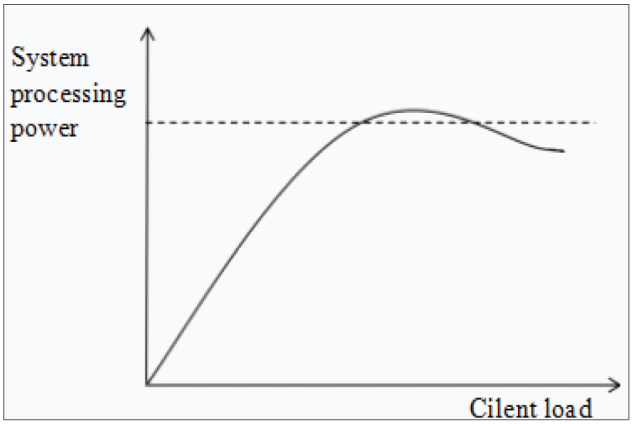
hardware resource data, including CPU, memory, disk, network bandwidth and other factors; Server-Side-Capabilities are server capabilities, including database server type (MySQL or Postgre-SQL) and database configuration (knobs), etc.; F is a mapping function. In the process of intelligent parameter adjustment on the server side, the hardware resources can be considered to be temporarily unchanged and the server type is considered to be constant, as shown in Formula (2). From Formula (2), it can be seen that the performance of the system is related to client load and server configuration. The load of the client generally cannot be kept constant in an online production environment. When the client load changes, it will affect the result of intelligent tuning. In the industry, intelligent parameter adjustment method often ignores the influence of the load pressure of the client on the performance of the system while training the server-side parameter adjustment model:

$$P = F\left(Workload, Knobs\right) + C_{Hardware} + C_{ServerType} \tag{2}$$

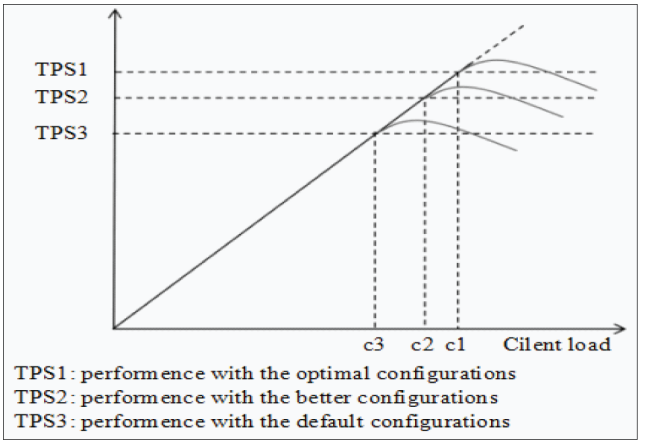
The relationship between client load pressure and system performance is shown in Figure 2, the horizontal axis represents the load capacity of the client, and the vertical axis represents the performance value of the overall system. The system performance of the database generally uses TPS (Throughput Per Second) or the average response time (Average Latency) to characterize. Theoretically, the client load in the offline simulation environment can be increased infinitely by means of distributed expansion. If the processing capacity of the database server does not have a performance bottleneck, the overall performance of the system and the load capacity of the client will approach a linear relationship. However, in the actual production environment, even if the database server adopts distributed expansion, due to the influence of database synchronization between the various nodes of the server and communication between data nodes, there must be a performance bottleneck in the processing capacity of the database server. It is observed on the premise that the server hardware resources are sufficient, the performance bottleneck of the server is determined by the database type (MySQL or PostgreSQl) and database configuration parameters. During the database tuning process, the database type is generally considered to remains unchanged and only the database configuration parameters can be adjusted. Through a large number of experiments in an offline environment, we came to the following conclusions (as shown in Figure 2 (a)). When the load pressure of the client exceeds the performance bottleneck of the server, as the client load capacity continues to improve, the overall system performance will rise slowly, then drop slightly, and finally stabilize. Therefore, it is meaningful to adjust database parameters only when the client load reaches the performance bottleneck of the server.

We take Figure 2 (b) as an example to illustrate this issue. Assume that the server processing capacity corresponding to the database default configuration can make the system performance reach TPS3, and its corresponding client load is c3. In this case, only when the load capacity of the client is greater than c3, by adjusting the database parameters, it is possible to make the overall performance of the system reach TPS2. Otherwise, if the load capacity of the client has been lower than c3, no matter what algorithm is used, it will not be able to attain the system performance to reach TPS2. Similarly, if you want to achieve the system's optimal performance TPS1, the load on the client should be at least greater than c1. As shown in Figure 2 (c), when the client load is relatively small, the average delay of the system is comparatively stable. When the client load continues to increase to the database performance bottleneck, the average latency of the system begins to rise rapidly. At this time, by adjusting the database configuration parameters, the average latency of the system can continue to remain stable. Our goal is to find the best database configuration so that the average latency remains stable even when the client load is as large as possible. For example, we can find the database configsadauration corresponding to the client load c1 in Figure 2 (c). In summary, the necessary condition for the database tuning system to find the optimal solution is that the client load

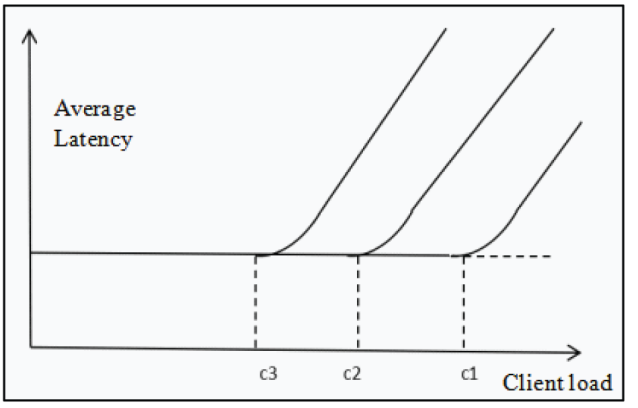
Figure 2. Relationship between client load and system performance in intelligent tuning scenarios



(a)Relationship between client load and processing power



(b) Relationship between Client-Load and throughput



(c) Relationship between client load and average latency

needs to exceed the maximum client load corresponding to the server performance bottleneck, which is ignored by the existing intelligent tuning system.

For the above analysis, we propose a method of sample filtering to filter out samples which generated when client load does not reach the performance bottleneck of the server. Only after this method, it makes sense for us to recommend parameters through the reinforcement learning model.

SA-DDPG Model Design

To remove the impact of client load on the overall system performance, we design the SA-DDPG model. The model uses a sample filter to filter out invalid samples and also filters out samples generated by the system when the client load does not reach the server performance bottleneck. In addition, we also add hardware resources and workload information consumed by the database to the model for training as the state features of the database to enhance the generalization ability of the model. The internal structure of the SA-DDPG model is shown in Figure 3.

The components of the SA-DDPG model are shown in Table 1, specifically, environment contains four parts: States (metrics), States (Hardware), States (Workload), and Actions (knobs). States (metrics) are performance-related indicators collected from the database system and are the non-adjustable part of the database. States (Hardware) is the resource consumption information of the database environment. These two parts constitute monitoring indicators. Actions (knobs) are the configurations of the database and are the adjustable part of the database. States (Workload) is the workload that the server bears in this training iteration, including the read/write ratios, queries, software and hardware environments and amount of user data, etc. Sample filter is a module for determining the validity of samples. Agent contains two modules, Critic and Actor. The input of Actor is S', the output is a set of action vectors, which can be converted into database configurations to be adjusted. The input of Critic is Observation S', Reward, Action and it outputs a score (Q-value), which can evaluate whether this action has a beneficial effect on database performance. Critic updates the weights of its neural network on the basis of the reward value and Actor updates the weights of its neural network by the score value. The configurations recommended by Actor is loaded to database in the Environment, and then a reward and a new Observation S' are generated after the load is pressurized and are sent

Figure 3. SA-DDPG model

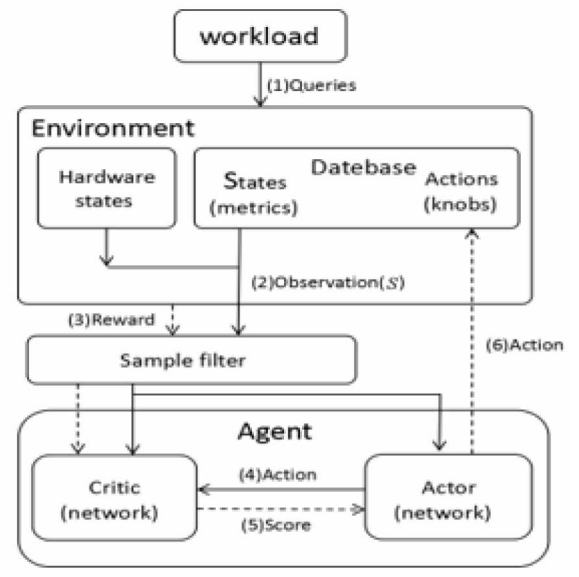


Table 1. The components of the SA-DDPG algorithm

Module	Description
Environment	Database environment
Actions (knobs)	Database configuration information
States (metrics)	Database state information
States (Hardware)	Hardware resource consumption information
States (Workload)	Workload information
Sample filter	Sample filter
Critic	Neural network to evaluate Actor
Actor	Neural network that generates actions

to the sample filter to check the validity. The sample filter is a neural network. Its input is the number of requests received per second counted by the PG database in the environment, the time between receiving requests, reward, internal statistical indicators of PG and the hardware resource consumption values. Its output is a valid flag. When the valid flag shows that the sample is valid, the saved reward and status value Observation S' will be sent to the Critic module of the Agent.

The SA-DDPG model uses the performance indicators of the database as States, and also adds the hardware resources and load consumed by the database as features to the model for training, which can more comprehensively characterize the impact of actions on the environment. The addition of a sample filter to the SA-DDPG model can filter out invalid samples, reduce the processing cost of the model, improve the sample quality, and also make the trained model more accurate. Furthermore, the optimal solution can be found in a shorter time.

SA-DDPG Model Training

The SA-DDPG model contains three networks: Sample filter, Critic and Actor. DS-DDPS first featurizes the SQL queries by considering rich features of the SQL queries. Then DS-DDPS feeds the query features into the DRL model to choose suitable configurations. We propose a Sample-Aware Deep Deterministic Policy Gradient (SA-DDPG) model to enable sample-aware database configuration tuning, which utilizes the actor-critic networks to tune the database configurations based on the hardware vector, the workload vector, and database states.SA-DDPG also adds SampleFilter to filter the samples. The pseudo code of the three networks is shown in Algorithm 1.

Sample Filter Training

The main function/goal of the sample filter is to filter out invalid samples, including the samples generated by the system when the client load does not reach the performance bottleneck of the server, so as to improve the quality of the samples.

- **Input data:** The input of the sample filter in the training phase is as S= $\left\{S_H, S_I, S_W\right\}$. We take PG database as an example. S_H is the information of the hardware resource consumed by PG, including CPU frequency, number of CPU cores, CPU utilization, memory size, memory utilization, disk IO utilization, etc. S_I is internal statistical data of PG, which contains total 86 indicators information abtaining from tables like pg_stat_database, pg_stat_user_tables, pg_stat_user_indexes, pg_statio_user_tables, pg_stat_bgwriter, pg_stat_replication, pg_stat_statement in the PG database. S_W is workload information, including the number of requests

Algorithm 1. Training SA-DDPG

Input:U:Sample set $\left\{(S_1,A_1,R_1,S_1^{'},L_1),(S_2,A_2,R_2,S_2^{'},L_2),...,(S_{|U|},A_{|U|},R_{|U|},S_{|U|}^{'},L_{|U|})\right\}$

Output: π_S : *The policy of SampleFilter*;
π_A : *The policy of Actor*;
π_C : *The policy of Critic.*

1: Init w_S, w_A, w_C;
2: *For* U_{item} in U do :
3: $\pi_S \leftarrow TrainSampleFilter\left(U_{item}, w_S\right)$
4: $\left(\pi_A, \pi_C\right) \leftarrow TrainCriticActor\left(U_{item}, w_A, w_C\right)$
5: end

Function TrainSampleFilter(U,w_S)

Input:U:{S,A,R,S',L};
w_S : *The weights of SampleFilter.*
Output: π_S : *The policy of SampleFilter*;
1: G=FC(S);
2: Accumulate the backward propagation error : $E = E + \frac{1}{2}||G - L||^2$;
3: Compute gradient ∇_{w_S};
4: Update w_S : $w_S = w_S + \varsigma\nabla_{w_S}$
5: Return

Function TrainCriticActor$\left(\mathrm{U}, w_A, w_C\right)$

Input:U:{S,A,R,S',L};
w_A : *The weights of Actor*;
w_C : *The weights of Critic*;

Output: π_A : *The policy of Actor*;
π_C : *The policy of Critic*;

1: Update the weights of target networks:
$w_A^{'} \leftarrow \varepsilon w_A + \left(1-\varepsilon\right) w_A^{'}$
$w_C^{'} \leftarrow \varepsilon w_C + \left(1-\varepsilon\right) w_C^{'}$
2: Estimation an Action-Value:Y = R + η $\pi_C^{'}(S', \pi_A^{'}(S' \mid w_A^{'}) \mid w_C^{'})$
3: *Accumulate the backward propagation error* : $\mathrm{E} = \frac{1}{N}\sum(Y - \pi_C(S, A \mid w_C))^2$
4: Compute gradient ∇_{w_C};
5: Update w_C : $w_C = w_C + \xi\nabla_{w_C}$
6: Update the weigthts w_A using the sampled policy gradient:
$\nabla_{w_A} J \approx \frac{1}{N}\sum \nabla_a \pi_C(s, a \mid w_C)\mid_{s=S, a=\pi_A(S)} \cdot \nabla_{w_A} \pi_A(S \mid w_A)\mid_S$
7: Update w_A : $w_A = w_A + \xi\nabla_{w_A}$
8: Return

received by the PG server per second, the average interval time between requests to reach the server and the average size of requests.

- **Model details:** Our solution uses the TPS during the training iteration. The sample filter network π_S is a fully connected neural network, including an input layer, two hidden layers and an output layer. The number of nodes in the three hidden layers are respectively 128,64 and 32. Specific parameter details are shown in Table 2. The output layer has only one dimension represented by G (shown in Formula (3)), in which G signifies the validity of the sample:

$$G = \begin{cases} 1, & valid \\ 0, & invalid \end{cases} \tag{3}$$

In order to make the neural network nonlinear, we add the ReLU activation function. The weight initialization uses the He initialization method We use Adam algorithm (Kingma & Ba, 2015) to accelerate the convergence speed of network training and the error function is designed as Formula (4):

$$E = E + \frac{1}{2} || G - L ||^2 \tag{4}$$

Table 2. The configurations of the policy network

	Input	Algorithm Function	Output
Actor: π_A and $\pi_A^{'}$	n_S	FC [n_states,128], LeakyReLU, BatchNorm1d [128]	n_actions
		FC [128,128], Tanh,Dropout, BatchNorm1d [128]	
		FC [128,64], Tanh, BatchNorm1d [64]	
		FC [64, n_actions]	
		Sigmoid	
Critic: π_C and $\pi_C^{'}$	n_S, n_actions	FC1 = (FC [n_states,128], Tanh)	1
		FC2 = (FC [n_actions,128], Tanh)	
		Temp=Cat (FC1, FC2)	
		FC [Temp, 256], LeakyReLU, BatchNorm1d [256]	
		FC [256, 64], LeakyReLU, BatchNorm1d [64]	
		FC [64,1]	
SampleFilter	n_S	FC [n_states, 128], Sigmoid	1
		FC [128, 64], LeakyReLU, Dropout, BatchNorm1d [64]	
		FC [64, 32], Tanh,BatchNorm1d [32]	
		FC [32, 1]	
		softmax	

Note: n_S is the sum of the quantities of S_H, S_I, and S_W dimensions.

- **Data annotation:** In order to train the sample filter network, we need to label the states $S = \left\{\mathrm{S,A,R,S'}\right\}$. First, we increase the workload from zero gradually and find the "inflection point" of the server performance by drawing the relationship curve between the workload and the overall system performance TPS, as shown in Figure 2(a). When training the SampleFilter offline, the workload pressure is appropriately increased at each iteration. If ones observe TPS grows linearly as the workload increases, it means that the sample is invalid. Otherwise, we will increase the workload to further judge whether the sample is valid. And if the TPS does not increase anymore after fine-tuning, the sample is valid. In this way, we get the label of the training sample of the neural network π_S: $L = \left\{L_1, L_2, ..., L_{|U|}\right\}$. During the training phase, label L and state data S are used as input data U: $Sample\,set\{(S_1, A_1, R_1, S_1^{'}, L_1), (S_2, A_2, R_2, S_2^{'}, L_2), ..., (S_{|U|}, A_{|U|}, R_{|U|}, S_{|U|}^{'}, L_{|U|})\}$, which is used to train the SA-DDPG model. Finally, we get the training model of the sample filter π_S after several iterative training until the model converges.

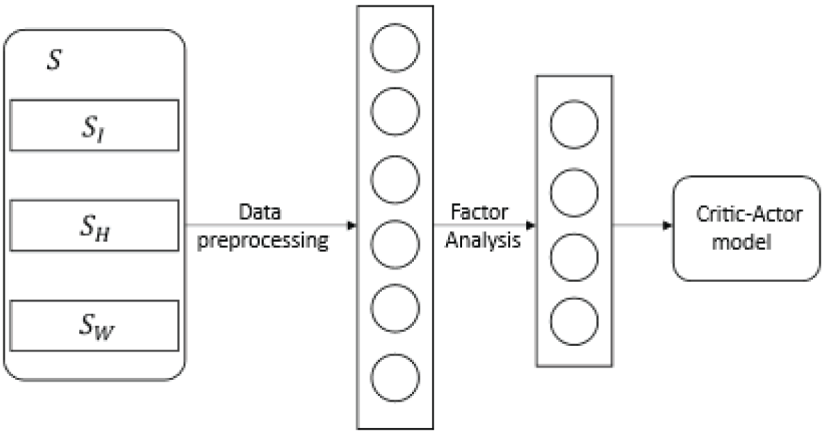
Critic Network and Actor Network Training

The two network Critic and Actor in the Agent are used to find a set of high-performance configuration parameters for the database environment. We use PG as an example for the description of the following data.

- **Input data:** When the client sends a request to the database system, it will use the PG statistical instructions to obtain the performance status indicators inside the PG database as $S_I^k = \{S_{new\_order}^k$, $S_{stock}^k, S_{customer}^k, S_{district}^k$, ..., $S_{item}^k\}$, where new_order, stock, customer, district, and item are all indicators of the internal state of the database. Meanwhile, by collecting and calculating, the state indicators of the hardware resources consumed by the environment is $S_H^k = \left\{S_{cpu}^k, S_{disk}^k, S_{mem}^k\right\}$, in which the status indicators of hardware resources are CPU resources consumed, hard disk resources consumed and memory resources consumed. Also, data indicators of the current load are collected as $S_W^k = \left\{S_{tps}^k, S_{inner}^k, S_{size}^k\right\}$, where the data indicators of the load are the current average number of requests per second, the average interval of the number of request and the average data size of the number of requests. Last, the three types of state indicators are combined as the input of Critic and Actor: $S = S_I + S_H + S_W$.
- **Data preprocessing:** Before entering/feeding the data into the model, we first preprocess these performance index data from three aspects, as shown in Figure 4:
  - Remove the non-numerical parameters in the performance indicators.
  - Accumulate the data with multiple values in multiple tables in a cumulative way.
  - Use factor analysis method (Pedregosa *et al*, 2011) to reduce the dimensionality of performance data in order to reduce the data correlation.
- **Training process:** In each training iteration, we choose a sample $\mathrm{U:\{S,A,R,S',L\}}$ from the sample pool. Note that training Actor and Critical networks does not require label data L. The sample U as the input of Critic and Actor network. We will first update the latest parameters to target networks $\pi_A^{'}$ and $\pi_C^{'}$. As shown in the following formula:

$$w_A^{'} \leftarrow \varepsilon w_A + \left(1-\varepsilon\right) w_A^{'} \tag{5}$$

Figure 4. The processing flow of DBtune system status



$$w_C^{'} \leftarrow \varepsilon w_C + \left(1-\varepsilon\right) w_C^{'} \tag{6}$$

We use this gradient result to update the parameters of the Actor network. We finally calculate the real action value according to the Bellman equation (Sutton, 2018), using Formula (7):

$$Y = R + \eta \pi_C^{'}(S', \pi_A^{'}(S' \mid w_A^{'}) \mid w_C^{'}) \tag{7}$$

Here η is a discount factor that can strengthen short-term rewords, weaken forward rewords, and also ensure the convergence of the Critic-Network π_C. We use this output Y as the true value for evaluating network π_C. Then we use Y and the output of evaluation network Critic to calculate the error and gradient, as shown in Formula (8):

$$E = \frac{1}{N} \sum (Y - \pi_C(S, A \mid w_C))^2 \tag{8}$$

We finally update the evaluation network Critic with Formula (9):

$$w_C = w_C + \xi \nabla_{w_C} \tag{9}$$

The parameters of the Critic network are updated by minimizing the loss function, and iterated continuously until the end of the training. Using Formula (10), we then calculate the gradient of the Actor network according to the output Q value of the Critic network:

$$\nabla_{w_A} J \approx \frac{1}{N} \sum \nabla_a \pi_C(s, a \mid w_C) \mid_{s=S, a=\pi_A(S)} \cdot \nabla_{w_A} \pi_A(S \mid w_A) \mid_S \tag{10}$$

$$w_A = w_A + \xi \nabla_{w_A} \tag{11}$$

- **Reward function design:** The reward function is used in deep reinforcement learning to evaluate the return brought by the current action on the environment. The reward function can determine the search direction and convergence speed of the model to a certain extent, that is, the model will search along the direction of maximum reward. The reward function of DBtune can be combined with user needs, using TPS or latency.

Taking the performance indicator TPS as an example, the difference change rate $\Delta perf_{t\to 0}$ between the performance indicator value $perf_t$ at time t and the initial performance indicator value $perf_0$ are calculated as shown in Formula (12). Also, the difference change rate $\Delta perf_{t\to t-1}$ between the performance indicator value $perf_t$ at time t and the performance indicator value $perf_{t-1}$ in the previous iteration are calculated as shown in Formula (13):

$$\Delta perf_{t\to 0} = \frac{perf_t - perf_0}{perf_0} \tag{12}$$

$$\Delta perf_{t\to t-1} = \frac{perf_t - perf_{t-1}}{perf_{t-1}} \tag{13}$$

Based on the above two formulas, we define the reward function based on performance indicators in Formula (14):

$$R_{perf} = \begin{cases} \left(\left(1+\Delta perf_{t\to 0}\right)^2 - 1\right)\left|1+\Delta perf_{t\to t-1}\right|, \Delta perf_{t\to 0} > 0 \ and \ \Delta perf_{t\to t-1} >= 0 \\ 0, \Delta perf_{t\to 0} > 0 \ and \ \Delta perf_{t\to t-1} < 0 \\ \left(\left(1-\Delta perf_{t\to 0}\right)^2 - 1\right)\left|1+\Delta perf_{t\to t-1}\right|, other \end{cases} \tag{14}$$

SA-DDPG Model Tuning Process

After the model is trained, we use Algorithm 2 to choose database config parameters. We first vectorize the Workload-Information, Hardware-Information, and Internal-database-States-Information. Then, we input the result S into SampleFilter to determine whether it is valid. If it is valid, we generate database configuration and generate samples to add to the SmallSamplePool. If the S judgment is invalid, it indicates that there is no need to regenerate the database parameters and exit directly.

Algorithm 2 is the pseudo code of the parameter tuning process of the SA-DDPG model. When the Query request is sent to the database, the data generated in the database environment is as the formula: $S = \left\{S_H, S_I, S_W\right\}$. S_H is the information of the hardware resource consumed by PG, including CPU frequency, number of CPU cores, CPU utilization, memory size, memory utilization, disk IO utilization, etc. S_I is internal statistical data of PG. S_W is workload information, including the number of requests received by the PG server per second, the average interval time between requests to reach the server and the average size of requests.

These data are vectorized and sent to the sample filter for predictive filtering. If the filtering result is shown as valid, the status data S is sent to the Actor network to generate configuration

Algorithm 2. Tuning with SA-DDPG

Input: U:Query set$\{q_1, q_2, ..., q_{|U|}\}$

Output: $A : Action knobs$

Generate S_H, S_I, S_W $from U$;

$S = \{S_H, S_I, S_W\}$

For q in U do :

    $G = \pi_S(S)$

    if $G == valid$:

        $A = \pi_A(S)$

        Deplpoy A;

        Generate S', R;

        SmallSamplePool $\Leftarrow \{S, A, R, S'\}$

    else

end

parameter knobs, and the database configuration file is updated by the knobs, and the Q value is calculated. After the configuration of the database Deployed, the performance evaluation data R of the database, the new metric data S' of the database, the configuration parameters A of the database, and the metric data S before A deployed are packaged into samples and sent to the Small-Sample-Pool.

EXPERIMENTAL RESULTS AND ANALYSIS

Experimental Environment

In this experiment, the database environment adopts the CPU specification of Intel(R) Xeon(R) Gold 6240L CPU @ 2.60GHz ×8, the memory is 16G, the disk is 200G and the database adopts postgreSQL10.11. The workload information is shown in Table 3.

We first use the Losso algorithm to sort the relevance of configuration items to the performance and take the top 10 configuration items. After the identification of professional DBA, we finally focused on making intelligent recommendations for 12 configuration parameters. The specific configuration items are shown in Table 4.

Table 3. The workload information

Name	Mode	Table	Cardinality	Size(G)	Query
Sysbench	RW	3	4000000	11.5	19190636
JOB	RO	21	7305123	13.1	113
TPC-H	RO	8	163120155	50.1	22

Table 4. Key configuration items to be adjusted

max_wal_size	max_parallel_workers_per_gather
checkpoint_timeout	effective_cache_size
bgwriter_lru_maxpages	work_mem
default_statistics_target	temp_buffers
shared_buffers	random_page_cost
autovacuum_vacuum_cost_delay	checkpoint_completion_target

Ablation Experiment

We conducted two ablation experiments, comparing whether to use SampleFilters and whether to use additional-states. Our experiment chose PG database as the parameter tuning target. The workload adopts a load that simulates financial transaction business. The two algorithms adjusted their parameters 50 times respectively. The results are shown in Figure 5 (a) and Figure 5 (b).

As shown in Figure 5(a) and Figure 5(b), both Used-SampleFilter and Used-additional-states alone can achieve good parameter tuning results, which compared to simply using the DDPG algorithm and internal database parameters. Used-SampleFilter is more effective than Used-additional-states. Used-SampleFilter can stabilize the TPS of the database at around 13500. Used-additional-states can stabilize the TPS of the database at around 13000. The TPS of the database without our method ultimately stabilized at 11000, and our method improved by approximately 18%.

Online Testing

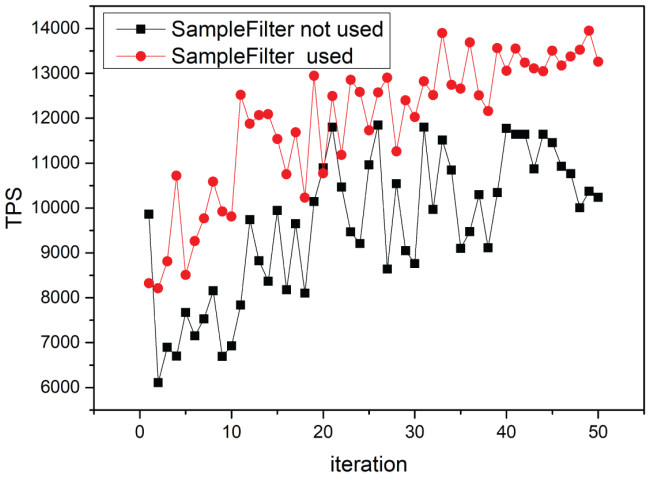
We conducted online testing on SA-DDPG. In a production environment of a domestic southern operator, the database system deployment framework is shown in Figure 6(a). The physical devices of the two nodes Slave1 and Slave2 are exactly the same and their business load can be considered exactly the same. At 18:45 on a certain day, due to the query business volume suddenly increased, two slave nodes raised performance alarms at the same time. The DBtune tuning system was deployed on Slave1, and the reinforcement learning recommendation algorithm with-out sample filter, DBtune-non-sample-filter, was deployed on Slave2. Both algorithms were configured that a training iteration was 5 minutes.

As shown in Figure 6 (b) and Figure 6 (c), at 18:47, the query TPS of the two slave nodes reached about 7900, the query delay increased and exceeded the threshold and the alarm was triggered. The alarm triggered DBtune and DBtune-not-using-the-scheme at the same time and the goal was to increase TPS by 10% (Equal to increase to more than 8900). Also, the data collection and incremental training were started. At 19:04, DBtune recommended the optimal configuration results for this business scenario with 9297TPS and achieved the goal first. The service returned to normal at 19:06 after the recommended configurations took effect in the production environment and the performance alarm was eliminated. However, DBtune-not-using-the-scheme achieved the system performance improvement goal by the recommended configuration at 19:43. In summary, through the above experiments, it can be seen that the DBtune system with sample filter performs better in online parameter adjustment, and it also proves that the effect of parameter adjustment recommendation can be improved after sample filtering.

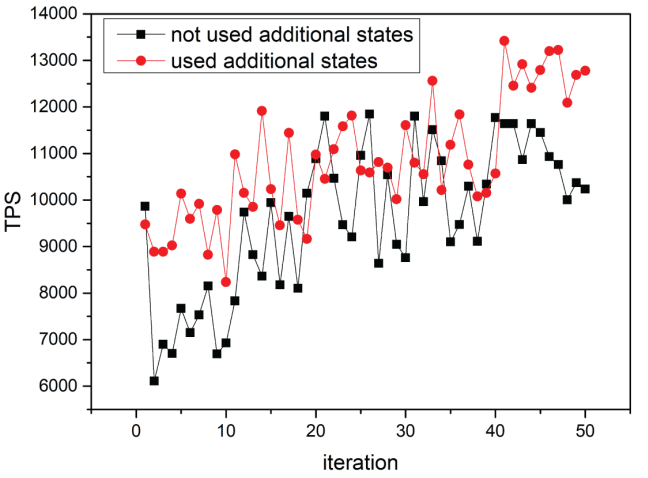
Comparison With Existing Techniques

In order to verify the effect of the scheme proposed, we use the standard data set to do experiments and compare with the performance of other similar systems in the industry.

Figure 5. Ablation experiment



(a) Comparison between Not-used-SampleFilter and Used-SampleFilter

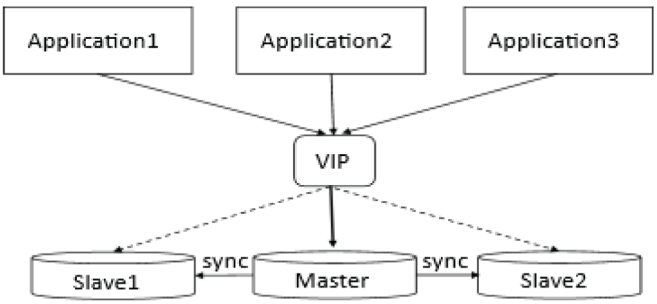


(b) Compare Not-used-additional-states and Used-additional-states

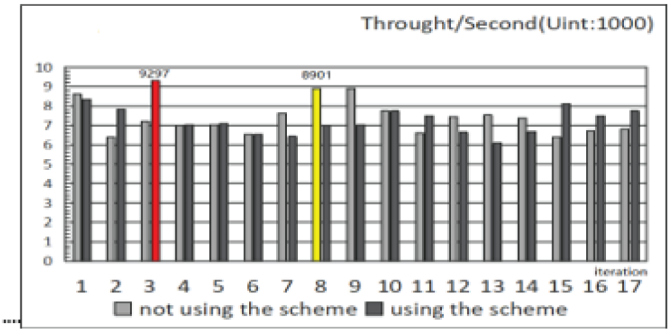
We modified the database compression tool program to control the database workload pressure by controlling the number of client connections of the compression tool. In the training process, half of the samples are fully pressurized samples, and the other half are insufficient pressurized samples. Our DBtune solution uses the SA-DDPG algorithm to obtain a reinforcement learning inference model and other methods use the samples to train and obtain their respective models. Each method uses the trained model to infer the respective configuration parameters by using the standard data set to load Sysbench (RW), JOB (RO), TCP-H (RO), and then uses the standard data set to perform performance test comparison after the configuration parameters take effect.

The basic model is trained and generated using two months of historical sample data. OtterTune is installed and deployed with the latest open-source version to train and debug for three days (Zhang

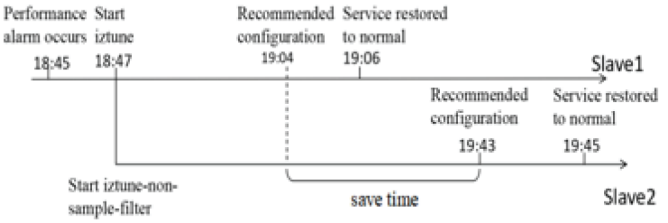
Figure 6. Comparison of parameter adjustment between using this scheme and not using this scheme



(a)·Production·environment·



(b)·Comparison·of·parameter·adjustment·effects·based·on·our·method·or·not



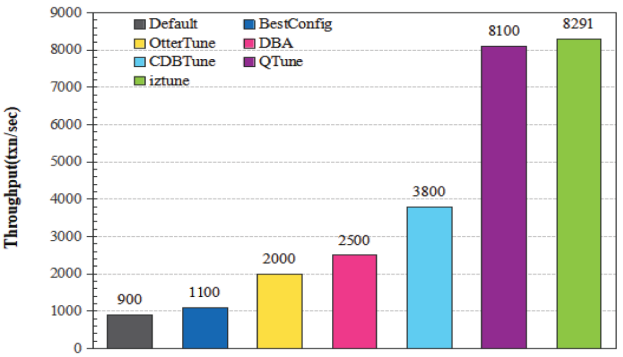
(c)Fault·handling·timeline

et al., 2018). This experiment invites a DBA with 5 years of work experience in ZTE to debug the PG parameters for a week. CDBTune (Zhang et al., 2019) has conducted online debugging for nearly twenty hours. The experiment uses three business load models of Sysbench (RW), TPC-H (RO) and JOB (RO) for comparative testing. SysBench is a modular, cross-platform and multi-threaded benchmark test tool, which is mainly used to evaluate and test the database load under various system parameters. We use SysBench for OLTP (Online transaction processing) benchmark test. TPC-H is a decision support system benchmark test for the retail industry. It defines 8 tables and 22 queries to reflect the system's ability to handle multiple queries, such as the size of the database selected to execute the query, the query processing power when a single stream submits the query and the query throughput when multiple concurrent users submit the query. JOB(RO) is a workload proposed in recent years and it defines 21 tables and constructs 113 analytical SQL queries based on the IMDB database. Focusing more on connection order is an extremely important workload method in query optimization problems. The query is designed to have 3 to 16 connections with an average

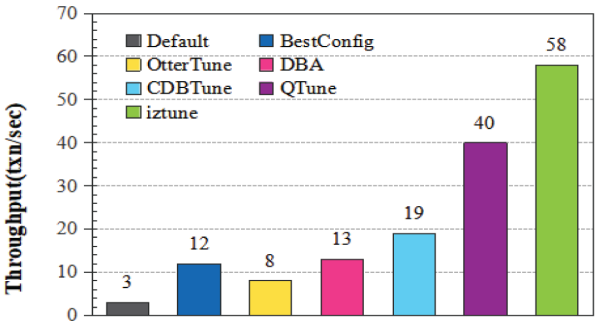
of 8 connections. Compared with the existing methods OtterTune (Zhang et al., 2018), CDBTune (Zhang et al., 2019), BestConfig (Zhu et al., 2017) and QTune (Li, Zhou, Li & Gao, 2019), the PG parameters recommended by DBtune with these three models have the best effect compared to other tuning parameters and perform best in the TPC-H benchmark test.

The experimental results of specific throughput are shown in Figure 7 (a), Figure 7 (b), and Figure 7 (c). The experimental results of latency are shown in Figure 8 (a), Figure 8 (b), and Figure 8 (c), respectively.

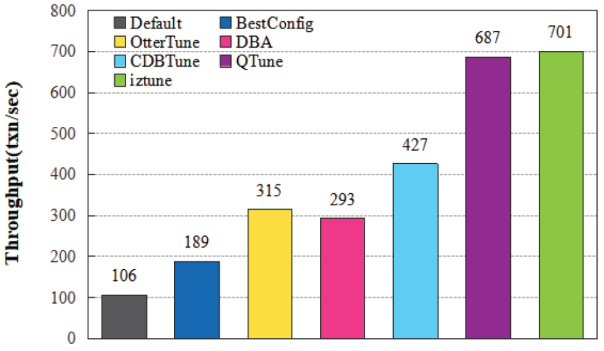
Figure 7. Throughput comparison between existing methods Default settings, BestConfig, Sysbench, OtterTune, DBA (RW), CDBTune, and QTune



(a) Sysbench (RW)

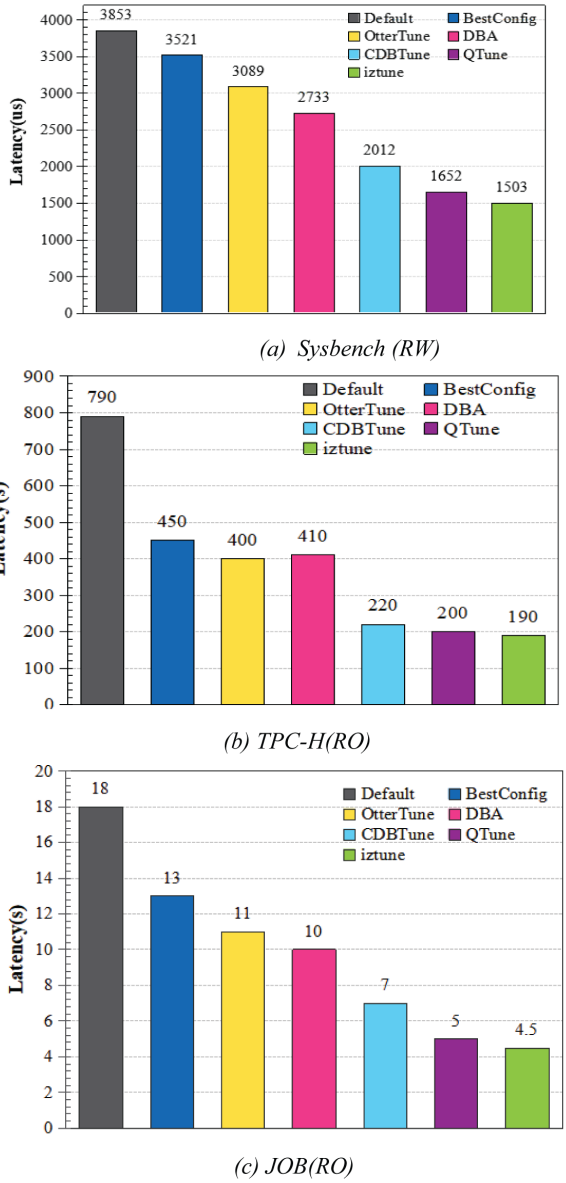


(b) TPC-H(RO)



(c) JOB(RO)

Figure 8. Latency comparison between existing methods Default settings, BestConfig, Sysbench, OtterTune, DBA (RW), CDBTune, and QTune



(a) Sysbench (RW)

(b) TPC-H(RO)

(c) JOB(RO)

The DBA debugs performance required rich expert experience and needs to manually debug repeatedly. Once a problem occurs, it is necessary to troubleshoot whether it is a parameter problem or a hardware resource problem. It takes long time and is difficult to find a set of better configuration parameters in a high dimensional space. The BestConfig method uses interval mapping to find the optimal solution, so it can only find the local optimal solution in a specific interval. There is only a certain probability that the better configurations will be found, so the performance is not very stable. The CDBTune (Zhang et al., 2019) system is also based on reinforcement learning to train the

Actor-Critic network and uses Actor output recommended configuration, but the samples used are not filtered and the sample quality is poor. Thus, it takes a lot of time to find better configurations and it is difficult to train a good network with a poor-quality sample to recommend. QTune (Li, Zhou, Li & Gao, 2019) uses the features of query sentence to train the model, and to a certain extent, strengthens the performance characterization capabilities of model. However, there are also sample quality problems, and the hardware resources and load characteristics that the database depends on are not added to the model training, resulting in the lack of performance characterization capabilities of the trained model. Our experiments also fully prove the above analysis.

From the experimental data in Table 5, it can be seen that our method demonstrate improved performance to varying degrees compared to similar methods in all three load types. When the workload type is Sysbench (RW), our method improves the Throughput (txn/sec) metric by at least 2% and reduces the latency metric by at least 9%; When the workload type is TPC-H (RO), our method improves the Throughput (txn/sec) metric by at least 45% and reduces the latency metric by at least 5%; When the workload type is TPC-H (RO), our method improves the Throughput (txn/sec) metric by at least 45% and reduces the latency metric by at least 5%.

RELATED WORK

Database intelligent tuning mainly divided into three classes: traditional intelligent tuning, intelligent tuning based on machine learning and intelligent tuning based on reinforcement learning:

- **Traditional intelligent tuning:** Traditional intelligent tuning methods are generally based on expert experience. For example, the MySQL optimization tool, MySQL-Tuner, can provide configuration suggestions based on information such as workload. Duan et al. introduced iTuned (Duan, Thummala & Babu, 2014), a practical tool for database parameter tuning using statistical experiments. It uses statistical rules to find the configurations that have the most impact on the database and can achieve the highest performance. Wei, Ding & Hu (2014) presented a database tuning method based on fuzzy rules. Zheng, Ding & Hu (2014) proposed a statistical method to identify key systems and utilized a neural network to tune parameters to match specific business workloads. BestConfig (Zhu et al, 2017) grids the parameters of the system based on expert experience, and continuously searches and reduces the grid space through recursive

Table 5. Comparison of experimental results between our method and similar methods

Workload		Default	Bestconfig	Ottertune	DBA	CDBTune	QTune	DBTune	Improve
Sysbench (RW)	Throughput(txn/sec)	900	1100	2000	2500	3800	8100	8291	2%+ ↑
	Latency(s)	3853	3521	3089	2733	2012	1652	1503	9%+ ↑
TPC-H (RO)	Throughput(txn/sec)	3	12	8	13	19	40	58	45%+ ↑
	Latency(s)	790	450	400	410	220	200	190	5%+ ↑
JOB (RO)	Throughput(txn/sec)	105	189	315	293	427	687	701	2%+ ↑
	Latency(s)	18	13	11	10	7	5	4.5	10%+ ↑

iteration method to find the optimal sample. Traditional intelligent tuning methods generally have shortcomings, such as strong dependence on expert experience, only local optimum, time consuming tuning process and complicated system operation and maintenance.

- **Intelligent tuning based on machine learning:** Students and researchers in the database research group of Carnegie Mellon University have developed a database intelligent tuning tool called OtterTune (Aken et al., 2017). OtterTune chooses the important and independent database state indicators by Factor Analysis (Ghahramani & Hinton, 1996) algorithm, and uses the K-means (Chinrungrueng & Sequin, 1991) algorithm to reduce the dimension, denoise, and improve the robustness for the filtered state indicators. It also utilizes Lasso (Tibshirani, 1996) technology to filter out the effective knob configurations. Meanwhile, the configurations are optimized and a configuration scheme that suit the actual data collected by the current statistical model will be selected. To better understand ML-based tuning, Aken et al. (2021) conducted a thorough evaluation of ML-based DBMS knob tuning methods on an enterprise database application. They use the OtterTune tuning service to compare three state-of-the-art ML algorithms on an Oracle installation with a real workload trace. A method for accelerating automatic tuning was proposed in (Konstantinos et al., 2020), which can automatically filter out configurations that have little impact on performance according to the current hardware environment. The drawback of this method is that it does not consider the effect of client load, so the tuning result may not give the optimal solution due to the sample problem. Tan et al. (2019) proposed a prediction model based on data-driven and neural network. In accordance with the changes of database load and performance, it dynamically tunes the database Buffer Pool to fully make use the database memory resources and then reduces costs. Kanellis et al. (2022) proposed a database tuning device called LlamaTune, which leverages domain knowledge to improve the sample efficiency of existing optimizers. LlamaTune employs an automated dimensionality reduction technique based on randomized projections, a biased-sampling approach to handle special values for certain knobs, and knob values bucketization, to reduce the size of the search space. Zhang et al. (2021) design ResTune to automatically optimize the resource utilization without violating SLA constraints on the throughput and latency requirements. ResTune leverages the tuning experience from the history tasks and transfers the accumulated knowledge to accelerate the tuning process of the new tasks. The prior knowledge is represented from historical tuning tasks through an ensemble model. The model learns the similarity between the historical workloads and the target, which significantly reduces the tuning time by a meta-learning based approach. The limitation of intelligent tuning based on machine learning manifests is that, on the one hand, it requires a large number of training samples and the time cost is relatively high; on the other hand, it can only offer recommendation for a small number of configurations and is hard to find an optimal solution in the high-latitude parameter space.
- **Intelligent tuning based on reinforcement learning:** Reinforcement learning (Lillicrap et al, 2016) is the way that the agent continuously interactively learns in the environment through trial and explore, maximizes the cumulative rewards obtained in the process of interacting with the environment and finds the best strategy to solve the problem. Almost all reinforcement learning problems can convert to Markov decision process (MDP) (Sutton & Barto, 2018), and the common algorithms include Q-learning (van Hasselt, Guez, & Silver, 2016; Watkins & Dayan,1992), Policy Gradients (Lin et al, 2018) and so on. Zhang et al. (2019) proposed an end-to-end tuning system, CDBTune, which used the deep reinforcement learning methods. Specially, they used the Deep Deterministic Policy Gradient (DDPG) method for training, which can be applied to more complex situations with large action space and continuous action space. Li, Zhou, Li & Gao (2019) proposed a query-aware database tuning system QTune based on reinforcement learning. This system can provide three different granularities of debugging, namely query statement level, workload level and overall system level. At the same time, the system will use the database SQL query sentence to extract features, and then the extracted features are input into

the Action-Critic (Li, Zhou & Li, 2019) model of the DDPG algorithm together with the state of the database environment for training to obtain recommended parameters. Cai et al. (2022) proposed the hybrid tuning system Hunter, which utilizes the genetic algorithm GA to collect relatively excellent training samples in a short period of time. By compressing indicators and selecting knobs in the spatial optimizer, the data dimension is reduced to reduce computational complexity, thereby reducing the computational complexity. The shortcoming of this system is that the recommended parameters are still unable to distinguish whether they are effective, and these parameters may be local optimal rather than global optimal.

CONTRIBUTIONS

Database intelligent parameter tuning is an adaptive adjustment of parameters based on different business loads. It allows for the full utilization of limited database resources. It also provides the best technology for database performance. Intelligent database tuning not only improves database performance but also reduces database costs. The intelligent parameter tuning of databases is increasingly receiving attention from both academia and industry.

The paper proposes a database intelligent parameter tuning method based on sample perception. This method uses a large and small sample pool mechanism to quickly accumulate database fault samples. It incorporates the business load of the database and the database hardware environment as features into the model training, while using neural network filters to filter out invalid samples. The paper finally uses deep reinforcement learning algorithm to iteratively optimize the database parameter configuration, making the database performance continuously approach the global optimal solution.

CONCLUSION

The current database intelligent parameter adjustment system cannot confirm the validity of sample, and it is easy to make the result locally optimal instead of the global optimal. In order to solve this problem, in this paper, we develop a new comprehensive AI database tuning system, which combines offline tuning and online tuning, and propose the SA-DDPG algorithm to train the model. In this algorithm, a method of screening and eliminating invalid samples is used to improve the sample quality and the validity of recommended configurations of the model. By adding the hardware resources data consumed to the model training, the performance representation capabilities of model are enhanced and the robustness of models under different business loads is improved. On the one hand, with the development of cloud-enabled database, the future intelligent tuning will consider more distributed intelligent parameter tuning methods in the cloud environment. On the other hand, with the development of artificial intelligence technology, especially the development of reinforcement learning, it can be believed that more new solutions will be applied to the intelligent database tuning.

Note that different database types may have different numbers of parameters although they contain the same major parameters. But their principles of parameter tuning are the same. In this paper, we take PG as an example to demonstrate our proposed method. In our future work, we will widely verify our proposed method over other database types to further demonstrate that the proposed method is not limited to a particular database type.

REFERENCES

Aken, D. V., Pavlo, A., Gordon, G. J., & Zhang, B. (2017). Automatic database management system tuning through large-scale machine learning. *Proceedings of the 2017 ACM International Conference on Management of Data*, 1009-1024. doi:10.1145/3035918.3064029

Aken, D. V., Yang, D., Brillard, S., Fiorino, A., & Pavlo, A. (2021). An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *14*(7), 1241–1253. doi:10.14778/3450980.3450992

Cai, B. (2022). HUNTER: An online cloud database hybrid tuning system for personalized requirements. *Proceedings of the 2022 ACM International Conference on Management of Data*, 646-659. doi:10.1145/3514221.3517882

Cai, B., Liu, Y., Zhang, C., Zhang, G., Zhou, K., Liu, L., Li, C., Cheng, B., Yang, J., & Xing, J. (2022). HUNTER: An Online Cloud Database Hybrid Tuning System for Personalized Requirements. *Proceedings of the 2022 SIGMOD Conference*, 646–659. doi:10.1145/3514221.3517882

Cereda, S., Valladares, S., Cremonesi, P., & Doni, S. (2021). CGPTuner: A contextual gaussian process bandit approach for the automatic tuning of IT configurations under varying workload conditions. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *14*(8), 1401–1413. doi:10.14778/3457390.3457404

Chen, J., Chen, Y., Chen, Z., Ghazal, A., & Zhou, M. (2019). Data management at huawei: Recent accomplishments and future challenges. *Proceedings of the 2019 IEEE 35th International Conference on Data Engineering*, 13-24. doi:10.1109/ICDE.2019.00009

Chinrungrueng, C., & Sequin, C. H. (1991). Optimal adaptive k-means algorithm with dynamic adjustment of learning rate. *Proceedings of the 1991 International Joint Conference on Neural Networks*, 855-862. doi:10.1109/IJCNN.1991.155291

Duan, S., Thummala, V., & Babu, S. (2009). Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *2*(1), 1246–1257. doi:10.14778/1687627.1687767

Eachempati, P., & Srivastava, P. R. (2022). Applications of Big Data Analytics in Investment Management: A Review and Future Research Agenda Using TCM Framework. *Journal of Database Management*, *33*(1), 1–32. doi:10.4018/JDM.299557

Fekry, A., Carata, L., Pasquier, T., Rice, A., & Hopper, A. (2020). To tune or not to tune?: In search of optimal configurations for data analytics. *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2494-2504.

Ghahramani, Z., & Hinton, G. E. (1997). *The EM algorithm for mixtures of factor analyzers.* Technical Report CRG-TR-96-1. University of Toronto.

Gur, Y., Yang, D., Stalschus, F., & Reinwald, B. (2021). Adaptive multi-model reinforcement learning for online database tuning. *Proceedings of the 24th International Conference on Extending Database Technology*, 439-444.

Kanellis, K., Alagappan, R., & Venkataraman, S. (2020). Too many knobs to tune? Towards faster database tuning by pre-selecting important knobs. In *Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems*. USENIX Association.

Kanellis, K., Ding, C., Kroth, B., Müller, A., Curino, C., & Venkataraman, S. (2022). LlamaTune: Sample-Efficient DBMS Configuration Tuning. *Proc. VLDB Endow*, *15*(11), 2953–2965.

Kanellis, K., Ding, C., Kroth, B., Müller, A., Curino, C., & Venkataraman, S. (2022). LlamaTune: Sample-efficient DBMS configuration tuning. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *15*(11), 2953–2965. doi:10.14778/3551793.3551844

Kingma, D., & Ba, J. (2015). Adam: a method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations*.

Konstantinos, K., Ramnatthan, A., & Shivaram, V. (2020). Too many knobs to tune? Towards faster database tuning by pre-selecting important knobs. *Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems*.

Kraska, T. (2019), SageDB: A learned database system. *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research.*

Li, G., Zhou, X., Li, S., & Gao, B. (2019). Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *12*(12), 2118–2130. doi:10.14778/3352063.3352129

Li, G. L., Zhou, X. H., & Li, S. (2019). Xuanyuan: An ai-native database. *IEEE Data Eng. Bull.*, *42*(2), 70–81.

Lillicrap, T. P. (2016). Continuous control with deep reinforcement learning. *Proceedings of the 4th International Conference on Learning Representations*.

Lin, R., Stanley, M. D., Ghassemi, M. M., & Nemati, S. (2018). A deep deterministic policy gradient approach to medication dosing and surveillance in the ICU. *Proceedings of the 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. doi:10.1109/EMBC.2018.8513203

M'barek, S., Baccouche, L., & Ben Ghezala, H. (2016). Model Driven Engineering for Quality of Service Management: A Research Note on the Case of Real-Time Database Management Systems. *Journal of Database Management*, *27*(4), 24–38. doi:10.4018/JDM.2016100102

Oh, J. S., & Sang, H. L. (2005). Resource selection for autonomic database tuning. *Proceedings of the 21st International Conference on Data Engineering Workshops*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., & Grisel, O. (2012). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tan, J., Zhang, R., Zhang, T., Li, F., & Cao, W. (2019). Ibtune: Individualized buffer tuning for large-scale cloud databases. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *12*(10), 1221–1234. doi:10.14778/3339490.3339503

Tibshirani, R. (2011). Regression shrinkage and selection via the lasso: A retrospective. *Journal of the Royal Statistical Society. Series B, Statistical Methodology*, *73*(3), 267–288. doi:10.1111/j.1467-9868.2011.00771.x

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2094-2100.

Wang, W., & Siau, K. (2019). Artificial Intelligence, Machine Learning, Automation, Robotics, Future of Work and Future of Humanity: A Review and Research Agenda. *Journal of Database Management*, *30*(1), 61–79. doi:10.4018/JDM.2019010104

Watkins, C., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, *8*(3-4), 279–292. doi:10.1007/BF00992698

Wei, Z., Ding, Z., & Hu, J. (2014). Self-tuning performance of database systems based on fuzzy rules. *Proceedings of the 2014 International Conference on Fuzzy Systems and Knowledge Discovery*, 194-198. doi:10.1109/FSKD.2014.6980831

Weikum, G., Mönkeberg, A., Hasse, C., & Za Bb Ack, P. (2002). Self-tuning database technology and information services: from wishful thinking to viable engineering. *Proceedings of the 28th International Conference on Very Large Data Bases*, 20-31.

Yaniv, Y., & Shiba, M. (2020). Dynamic configuration tuning of working database management systems. *Proceedings of 2nd IEEE Global Conference on Life Sciences and Technologies*, 393-397.

Zhang, B., Van Aken, D., Wang, J., Dai, T., Jiang, S., Lao, J., Sheng, S., Pavlo, A., & Gordon, G. J. (2018). A demonstration of the OtterTune automatic database management system tuning service. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *11*(12), 1910–1913. doi:10.14778/3229863.3236222

Zhang, J. (2019). An end-to-end automatic cloud database tuning system using deep reinforcement learning. *Proceedings of the 2019 International Conference on Management of Data*, 415-432. doi:10.1145/3299869.3300085

Zhang, X. (2021). ResTune:Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. *Proceedings of the 2021 International Conference on Management of Data*, 2102-2114. doi:10.1145/3448016.3457291

Zhang, X., Wu, H., Li, Y., Tan, J., Li, F., & Cui, B. (2022). Towards dynamic and safe configuration tuning for cloud databases. *Proceedings of the 2022 International Conference on Management of Data*, 631-645. doi:10.1145/3514221.3526176

Zheng, C., Ding, Z., & Hu, J. (2014). Self-tuning performance of database systems with neural network. *Proceedings of the 10th International Conference on Intelligent Computing*, 1-12. doi:10.1007/978-3-319-09333-8_1

Zhu, Y., Liu, J., Guo, M., Bao, Y., & Yang, Y. (2017). Bestconfig: tapping the performance potential of systems via automatic configuration tuning. *Proceedings of the 2017 Symposium on Cloud Computing*, 338-350. doi:10.1145/3127479.3128605

Zhongliang Li is a PhD candidate at Nanjing University of Aeronautics and Astronautics, China. His research interests include intelligent database systems.

Yaofeng Tu received his PhD degree from Nanjing University of Aeronautics and Astronautics, China. His research interests include Big Data and artificial intelligence.