A Query-Level Distributed Database Tuning System with Machine Learning

Xiang Fang[1,2],Yi Zou[1,2],Yange Fang[1], Zhen Tang[1], Hui Li[1], Wei Wang[1,2,3,4,*]
[1]*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences*
[2]*University of Chinese Academy of Sciences*
[3]*Nanjing Institute of Software Technology*
[4]*University of Chinese Academy of Sciences Nanjing College*
Beijing, China
{fangxiang20,zouyi19,fangyange18,tangzhen12,lihui2012,wangwei}@otcaix.iscas.ac.cn

***Abstract*—Knob tuning is important to improve the performance of database management system. However, the traditional manual tuning method by DBA is time-consuming and error-prone, and can not meet the requirements of different database instances. In recent years, the research on automatic knob tuning using machine learning algorithm has gradually sprung up, but most of them only support workload-level knob tuning, and the studies on query-level tuning is still in the initial stage. Furthermore, few works are focus on the knob tuning for distributed database. In this paper, we propose a query-level tuning system for distribute database with the machine learning method. This system can efficiently recommend knobs according to the feature of the query. We deployed our techniques onto CockroachDB, a distribute database, and experimental results show that our system achieves higher performance under typical OLAP workload. For all categories of queries, our system reduces the latency by 9.2% on average, and for some categories of queries, this system reduces the latency by more than 60%.**

***Index Terms*—query-level, knob tuning, distributed database, machine learning**

I. Introduction

With the development of cloud computing and big data, the scenario of joint-cloud is gradually popularized. Its uses include Geo-Partition, separation of hot and cold data, etc, and can be applied to various fields, such as financial analysis, intelligent transportation. Obviously, traditional relational databases are not suitable for this distributed scenario. In this background, native distributed databases emerge as the times require. Native distributed database means that the architecture design, underlying storage and query processing are all oriented to the needs of distributed data management, and the database cluster as a whole provides services to the outside world [1]. CockroachDB [22] is one of the most successful cases. It has scalability, strong consistency, and high availability. Its partition function can well solve the above scenario with both OLTP and OLAP characteristics.

In order to get better performance, database knob tuning is is a basic work. However, tuning hundreds of database knobs(or parameters) has become increasingly complex as the database grows. The industry is more and more inclined to use machine learning tuning methods instead of manual labor [4]. At present, there are many successful cases in this field [8, 10, 12, 13]. However, there are still some deficiencies. First, most of the existing mature knob tuning methods are coarse-grained. These methods are suitable for OLTP scenarios only. For OLAP scenarios, it is necessary to design more fine-grained tuning methods. Second, distributed databases have more knobs than traditional relational databases, but few tuning works can be directly migrated to distributed database scenarios.

Tuning distributed database for OLAP scenarios mainly has the following challenges: first, how to recommend different configurations according to the feature of different query statements; second, how to select effective knobs to tune among the massive knobs of the distributed database; third, how to associate query statement features, database knob configurations and tuning performance to achieve automatic knob recommendation.

To address these problems, we propose a query-level knob tuning system with machine learning , which can efficiently tune the distribute databases under OLAP workload. Our tuning process is mainly divided into three steps: the first we analyze the key operations that are critical to performance in a given query; then, we select the knobs in the database that have an important impact on key operations, and further analyze the relationship between the values of these knobs and the latency; finally, according to the relationship between the knobs and the latency, recommend configuration for the query to improve its runtime performance.

The main contributions of this paper can be summarized as follows:

(1) We propose a query-level tuning system for distributed databases, which can automatically recommend configuration for each query.

(2) We propose a QVEN model, which is used to compress the query feature vector into a more compact compressed vector without losing key features.

(3) We propose a QPPN model, a knob-aware query performance prediction model based on neural network, which is used to model the functional relationship between configurations, query features and execution latency.

(4) We conducted experiments on typical OLAP workload (TPC-H) and CockroachDB. Experimental results showed that

our system achieved higher performance than default or random configurations and other workload-level tuning system.

The remainder of this paper is organized as follows. We explain the motivation to design a query-level tuning system for distributed database and OLAP scenarios in section II. We then provide an overview of our approach in section III, followed by a description of our techniques for query feature extraction in section IV, selecting the knobs that have the most impact in section V, performance prediction in section VI and configuration recommendation in section VII. In section VIII, we present our experimental evaluation. Lastly, we conclude with related work in section IX.

II. Motivation

CockroachDB [22] is an open source native distributed relational database. Its original design goal is to achieve scalability, strong consistency and high availability, and to support standard SQL interface externally. With its detailed documentation, active community and excellent performance, CockroachDB is used in all walks of life, and has an excellent reputation among Internet companies at home and abroad. Baidu, eBay, SPACEX, and DOORDASH all use CockroacShDB to build their infrastructure. Red Hat, VMware, and DAPTURE have all established good cooperative relations with the CockroachDB RD team. As an excellent open source project, CockroachDB is also a pioneer in the field of distributed database systems. Many later distributed database systems refer to its design concept and system architecture.

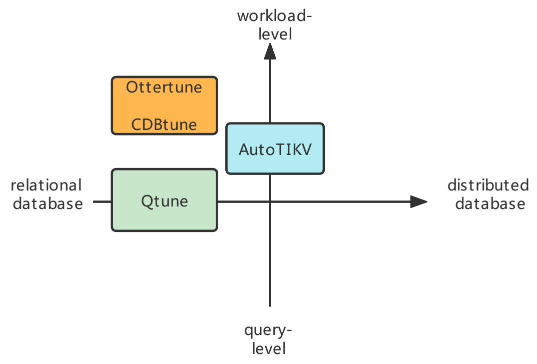

Fig. 1. Comparison of existing tuning systems

Our previous work is to extend Ottertune to CockroachDB. The experimental results show that under OLTP workload(including TPC-C, YCSB), Ottertune has a better improvement effect than the default configuration. But under OLAP workload(TPC-H), the effect is not very significant. As the Figure 1 shows, most existing tuning work are workload-level tuning, whose tuning goal is to improve the performance of the database system running for a period of time, and recommend the same configurations for all queries. In contrast, the tuning goal of the query-level knob tuning system is to improve the performance of each query within a period of time when the database system is running, that is, to recommend configuration for each query before it is executed.

Therefore, it is representative and meaningful to design a query-level tuning system and conduct experiments on CockroachDB.

III. System Overview

In this section, we present the system overview of our tuning work. Figure 2 shows the architecture of our System, which contains four main components according to functions. Figure 3 shows the workflow. The controller is used to interact with the target database (such as executing queries, applying configurations and importing data, etc.) and processing user requests (such as generating training data requests, configuration recommendation requests). The parser is used to parse the response of the query, which includes the query vector, execution plan and latency. The configuration generator is used to generate configurations, including the knob sampling method and the recommendation method, which is called by the controller. The trainer is used for data persistence and neural network training. The data that needs to be persisted includes the training data and the weights of the neural network model.

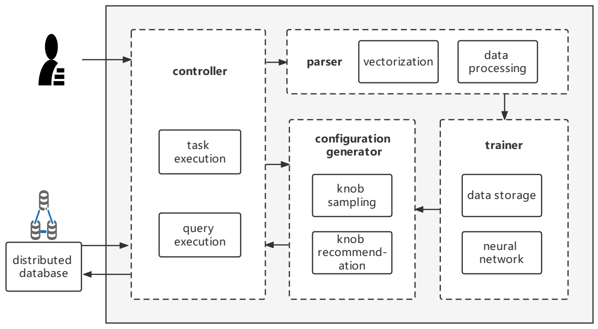

Fig. 2. Architecture of the system

The process of this system is divided into a training process and a recommendation process. Therefore, we next explain how each part of the system interacts through the training data generation process and the configuration recommendation process.

A. QVEN Model Training Data Generation Process

The training data of the QVEN model is the feature vector obtained by parsing the query statement and execution plan. First, the user initiates a QVEN model training data acquisition request to the controller. The controller then interacts with the target database, initializes the workload, and inputs the query statement and execution plan returned by the database into the parser. Finally, the parser parses, vectorizes, and preprocesses the input, outputs the feature vector of the query, and saves it on disk for training the QVEN model.

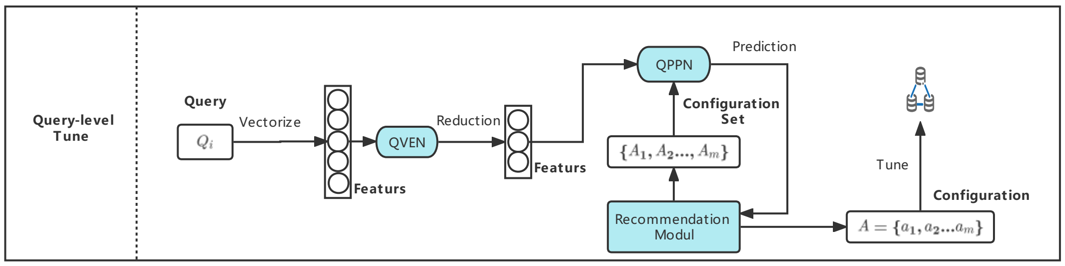

Fig. 3. Workflow of the system

B. QPPN Model Training Data Generation Process

The training data of the QPPN model is a set of tuples consisting of query vectors, knob vectors, and latency. First, the user initiates a QPPN model training data acquisition request to the controller. Then the controller calls the knob sampling method in the knob configuration generator to obtain the knob vector, and obtains the delay after applying the configuration to the database, and executing the query statement. Then call the parser to obtain the feature vector and input it into the trained QVEN model. Save the output query vector and the obtained knob vector and latency, that is, the QPPN model training set.

C. Configuration Recommendation Process

After the training is completed, the user initiates a knob tuning request of the specified query to the controller. Then the controller invokes the configuration recommendation method in the configuration generator to generate several candidate knob vectors, which are input into the QPPN model together with the query vector. Finally, the candidate knob vector is updated according to the output value of QPPN. After running several rounds, the recommended configuration is returned to the controller, and the controller applies the knob configuration on the target database, executes the query, and obtains the latency.

IV. Query Feature Extraction

The purpose of query-level knob tuning is to recommend configurations according to the feature of the query itself, so effective query feature extraction is the basis for solving the problem of query-level knob tuning. Query feature extraction mainly faces the following problems. The first is how to parse the features in the query statement. The second is how to resolve performance-related features of query execution. The third is how to reduce the dimension of the feature vector. Next we discuss how to deal with these information in the following sections.

A. Query Statement Parsing

The query statement contains query type (such as SELECT, INSERT, UPDATE, DELETE) information, data table information, and operation information (such as JOIN, ORDER BY, GROUP BY, LIMIT). Among them, the query type encoding method is a one-hot vector of SELECT, INSERT, UPDATE, DELETE. That is, if it is a SELECT statement, the position is 1. The part of the data table information involved in the query is related to the specific workload. For example, the TPC-C workload contains 9 data tables, Then use at least 9 bits to encode it, and for new workloads, you need to import all data table names. The encoding method is that if a certain data table appears in the query, it will represent the position 1 of the table. The operation information part corresponds to keywords (such as SUM, SORT, ORDER BY). We count the occurrences of each keyword in the query statement and use it as the value of the corresponding bit of the keyword.

B. Execution Plan Parsing

In the database, user can execute a specific statement such as EXPLAIN to view a query's execution plan. The execution plan is a tree structure composed of nodes. Each node corresponds to a step in the database processing the query, which contains several attributes, such as the number of data rows, the estimated cost, and the use of indexes. In this paper we chooses to encode three node attributes (estimated number of rows, data table, estimated cost), and an additional attribute is added to calculate the number of occurrences of the same type of node. For nodes of the same type, their attributes are merged, numeric type attributes (such as estimated cost) are summed, and collection types (such as data tables) are unioned.

C. Query Feature Vector Dimension Reduction

The query statement and the execution plan respectively contain the static and runtime features of the query. However, the query feature vector obtained at present has the characteristics of high-dimensional sparseness. Sparse feature vectors often mean that a large number of high-quality data samples are required for model fitting. Therefore, in order to learn the relevant patterns of query features, configurations and latency from the data, it is also necessary to solve the problem of high-dimensional sparseness of query feature vectors. To address the above problem, we propose a Query Vector Embedding Network (QVEN).

1) QVEN Model:

The input is the feature vector parsed from the query statement and execution plan. The output is a compressed vector which is as compact as possible. Finally we use the compressed query vector (hereinafter referred to as query vector) to represent query features.

2) Training QVEN:

Train Data. The QVEN training set is a set of v, v is a floating-point vector representing query features obtained after parsing the query statement and its execution plan. In order to obtain training data, given a certain workload W, W is a set of query statements q. For each q, obtain its execution plan p on the target database, and then input q and p into the parser respectively. The statement parser and the execution plan parser concatenate the two encoded vectors to obtain a floating-point vector v representing the query feature, that is, a piece of QVEN training data. Process all the query statements in the above way, and then the initial data set of QVEN on the workload can be obtained.

Training. As Figure 4 shows, The encoder consists of a three-layer network, and the purpose of the encoder is to compress the original input. First, map the input between 0 and 1 through standard deviation normalization and max-min normalization to prevent gradient explosion and speed up the convergence of the model. The normalized feature vector is input into the first layer of neural network, and the first compressed feature vector is output through a linear layer and activation function Tanh. The second layer is also composed of a linear layer and activation function, and outputs the feature vector after the second compression. Finally, through the linear layer of the third layer of neural network, the feature vector after the third compression is output as the compression result of the input vector by the encoder. The decoder is also composed of a three-layer network. The purpose of the decoder is to restore the feature vector compressed by the original encoder. Therefore, the input vector is expanded once after each layer of the network, and the Tanh activation function is used in the first two layers of the decoder, the purpose of which is also to introduce nonlinearity. The last layer uses the Sigmod activation function, which outputs a value between 0 and 1, and its purpose is to map the output to the same range of values as the input.

The weights of the QVEN model are initialized using the standard normal distribution. Given a training set $\{v_1,v_2...,v_n\}$ whose training objective is to minimize the mean squared error L:

$$L = \frac{1}{n}\sum_{i=1}^{n} ||o_i - v_i|| \tag{1}$$

where o_i represents the output of the decoder, i.e. the reconstruction result of the feature vector for the query v_i. In this paper, the Adam optimizer is used to update the gradient, and the equal interval strategy (StepLR) is used to update the learning rate. The pseudocode of the training process is shown in Algorithm 1.

After the training of the QVEN model is completed, for the input query statement and the corresponding execution plan, the query vectorization method parses and reduces the dimension to obtain a compressed query vector. The pseudocode of the query vectorization process is shown in Algorithm 2.

Algorithm 1 QVEN Model Training

```
Input: $W$: The weight of QVEN; $S$: The training set; $epoch$: The training epoch
 1: Initiate $W$ by standard normal distribution;
 2: for $i$ for 0 to $epoch$ do
 3:     for each batch $V$ in $S$ do
 4:         Generate the output $O$ of $V$;
 5:         Calculate the backward propagation error:
 6:         $L = \frac{1}{n}\sum_{i=1}^{n} ||o_i - v_i||$
 7:         Calculate gradient $\bigtriangledown(E)$, update $W$;
 8:     end for
 9:     Update learning rate;
10: end for
```

Algorithm 2 Query Vectorization

```
Input: $Q$: The queries; $P$: Plan of the queries; $E$: The QVEN model encoder
 1: Parse query and plan:
 2: $V$ = Parse($P$,$Q$);
 3: Compress query vector:
 4: $C$ = $E(V)$;
 5: return $C$;
```

V. Important Knob Selection and Sampling

In this section, we present the principles of knobs selection in database and knobs' sampling method. The knob selection is a sub-issue of database knob tuning, which means to selecting important knobs from all optional knobs firstly. Then the following knob tuning is to select appropriate values for these selected knobs to improve database performance. Therefore, the basic principle of knob selection is that these knobs have an impact on performance, and modifying them will not cause database service exceptions. According to the above principles, we selects 23 cluster parameters on CockroachDBV21.1 (modifications take effect immediately without restarting the database service) as the set of knobs to be tuned.

Knobs to be tuned can be divided into numeric types (floating point numbers, integers) and enumeration types (boolean values, enumeration values). For numeric types, the sample space is generated according to the default value set for it by the database. Assuming that the default value of the knob is d, the sampling space is the interval $[\alpha * d, \beta * d]$, where α and β are the proportional coefficients between the minimum and maximum values of the sampling space and the default values. These coefficients can be set according to the tuning duration, but need to satisfy $\alpha < 1$ and $\beta > 1$. For an enumeration type, its sample space is all its values. Given the knob sampling space, this paper uses Latin Hypercube Sampling (LHS) to generate knob samples. LHS can make the distribution sufficiently uniform even when the number of

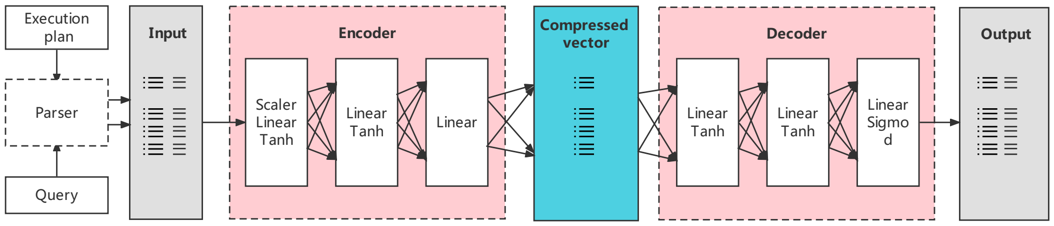

Fig. 4. Query vector embedding network(QVEN)

samples is small, so that the probability of the tuning algorithm to explore a more reasonable knobs combination is increased.

VI. Knob-aware Performance Prediction

This section introduces how to use query feature extraction technology, knob selection and sampling technology to construct a knob-aware query performance prediction model, the goal of which is to predict the performance of a query running under a certain knob configuration.

A. *QPPN Model*

For an input query and knob configuration, the goal of the Query Performance Predict Network (QPPN) is to predict the time it takes to execute the query after applying the knob configuration on the target database.

B. *Training QPPN*

Train Data. The training data is a set of tuples(c,k,t), where c represents the query vector, encoding the feature information of the query; k represents the knob vector, encoding the current knob information; t represents the latency in executing the query under the current knob configuration. The training goal of the performance prediction network is to make the output o of the network get closer and closer to t for any input c and k.

The steps to obtain QPPN training data are as follows: (1) Given a workload W={q_1,q_2...,q_3} and a knob space K={K_1,K_2...,K_m}, K has m dimension, which means that there are m knobs to be tuned, and K_m represents the sampling space of the m$_{th}$ knob. E represents the encoder of the QVEN model trained on the same type of load. (2) Run a round of workload on the target database with query granularity, and process q_1, q_2..., q_i in turn. (3) For the query q_i, obtain the execution plan p_i on the target database, input q_i, p_i and the encoder E into the query vectorization method , and obtain the query vector c_i. (4) Sampling in the knob space K by the knob sampling method to obtain a knob vector K_i of length m, and decode the knob vector K_i to obtain the corresponding knob configuration. (5) After applying the knob configuration on the target database, execute the query q_i, wait for the response from the target database, and obtain the latency t_i. Since then, a piece of training data (c_i, k_i, t_i) is obtained. (6) After the end of this round, get a piece of training data, go back to step (2), and perform the next round of training data acquisition. For a workload W containing n query statements, run S rounds according to the above method, and obtain $S * n$ pieces of QPPN training data.

Training. QPPN consists of a three-layer neural network whose structure is shown in the Figure 5. Before training the model, the input query vector and knob vector are normalized. The normalized input is passed through two consecutive neural network layers, each of which in turn consists of a linear layer, a BatchNorm layer, and the activation function ReLU. Among them, the purpose of the linear layer is to combine the input features, the purpose of the BatchNorm layer is to normalize the vector to prevent overfitting, and the purpose of the ReLU activation function is to introduce nonlinearity and avoid the neural network only learning simple linear transformations. After the output of the penultimate layer is subjected to the linear transformation of the last layer and the ReLU activation function, the output vector is compressed to one dimension by the SUM function and then output. The last layer uses the ReLU activation function because the purpose of the model is to predict query latency, which is always a value greater than zero.

The weights of the QPPN model are initialized using a standard normal distribution. Given a training set {(c_1,k_1,t_1),(c_2,k_2,t_2)...,(c_n,k_n,t_n)}, the training objective is to minimize the mean squared error L, which is defined as Equation 2.

$$L = \frac{1}{n}\sum_{i=1}^{n} ||o_i - log_2(t_i)|| \tag{2}$$

where O_i represents the output of QPPN, which is used to predict the logarithm of query latency. Because the latencys of different queries may differ by several orders of magnitude, the logarithm is taken, which is more convenient for the optimization of the neural network. Like QVEN, QPPN uses the Adam optimizer to update the gradients and uses the equal interval strategy (StepLR) to update the learning rate. The pseudo-code of the process is shown in Algorithm 3.

VII. Knob Configurations Recommendation

This section will introduce how to use query feature extraction methods and query performance prediction models for configuration recommendation. After the training phase, the performance prediction model predicts the latency of the query under the given knob configuration according to the query feature. On this basis, the gradient-based algorithm or the non-gradient-based algorithm can be used to search for

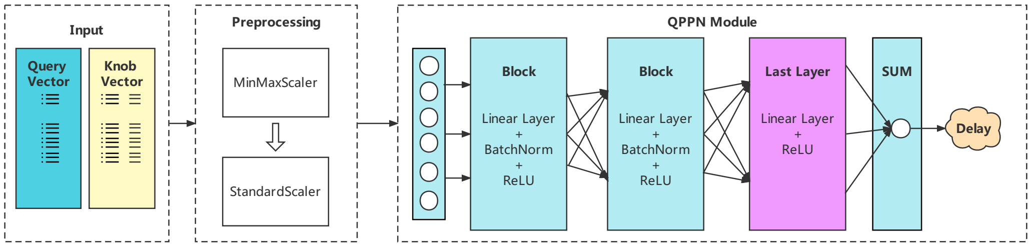

Fig. 5. Query performance prediction network(QPPN)

Algorithm 3 Query Performance Prediction

```
Input: c: The compressed vector of a query; K: The vectors of knob; M: The QPPN model
1: Calculate QPPN model output:
2: $O = M(c,K)$;
3: Calculate latency predict:
4: $L = 2^O$;
5: return $L$;
```

the optimal configuration of the given query. In this paper we choose genetic algorithm to avoid the problem of gradient descent falling into a local minimum, thereby obtaining sub-optimal knob configuration.

The workflow of configuration recommendation using genetic algorithm is shown in Algorithm 4. The user initiates a tuning request, gives a query q and obtains its execution plan p. q and p are inputed into the query vectorization method, and output the compressed query vector c. First, we initialize the current population as P, and for each iteration, encode the individuals in the current population as the corresponding knob vector K. The the knob vector k and the query vector c are input into the query performance prediction method, and the fitness function is calculated. After that, the population is selected according to the fitness function, and the excellent individuals are preserved. Then we crossover and mutate the individuals in the population, explore new individuals, obtain the next generation population P' and update the current population to P'. After T rounds of iterations, the knob vector corresponding to the best individual is decoded and configured as a recommended configuration.

Algorithm 4 Configuration Recommendation

```
Input: q: The query; M: The QPPN model; E: The QEVN model encoder; T: Total generation;
1: Initiate the first generate population $P$ by discrete uniform distribution;
2: Initiate generation count $t = 0$;
3: Get plan $p$ of query $q$;
4: Extract query feature from $p$ and $q$:
5: $c$ = QueryVectorize($q$,$p$,$E$);
6: while !$converged$ and $t < T$ do
7:     Encode population to knob vectors:
8:     $K$ = Binary2Digit($P$);
9:     Predict query latency by QPPN model:
10:    $L$ = LatencyPredict($c$,$K$,$M$);
11:    Calculate fitness values:
12:    $V = 1 / L$;
13:    Rank individuals by $V$:
14:    $R = P$[ArgSort($V$)];
15:    Generate next generation by $V$:
16:    $P'$ = Select($P$,$V$);
17:    Cross($P'$);
18:    Mutation($P'$);
19:    $P = P'$;
20: end while
21: return $R[0]$;
```

VIII. Experimental Evaluation

The target database used in this paper is CockroachDBV21.1. The environment for running the database service consists of four physical machines, three physical machines form a CockroachDB cluster, and one physical machine acts as a load balancer. Among them, each physical machine is configured with CentOS8 operating system, 128GB memory, 7TB solid state drive, and 2.50GHz CPU. The machine running the knob tuning system is configured with Ubuntu 18.04 operating system, 8GB memory, 250GB solid state drive, and 2.50GHz CPU. The load balancer is used to send query requests on behalf of users and distribute the pressure on the nodes in the cluster, while the database cluster provides services to the outside world as a whole. This paper uses the load balancing open source software HAProxy officially recommended by CockroachDB.

A. Workload.

The evaluation workload used in our experiments is TPC-H, and the data size is 10GB. The TPC-H benchmark test is modeled according to the real production running environment, which contains 22 kinds of queries (Q1-Q22). The main evaluation metric is the latency of each query, that is, the time from submitting the query to getting the response.

B. Training Data Collection

We generated 98,366 query statements and execution plans on TPC-H for the QVEN model, parsed them into query feature vectors through the parser, and saved them as initial

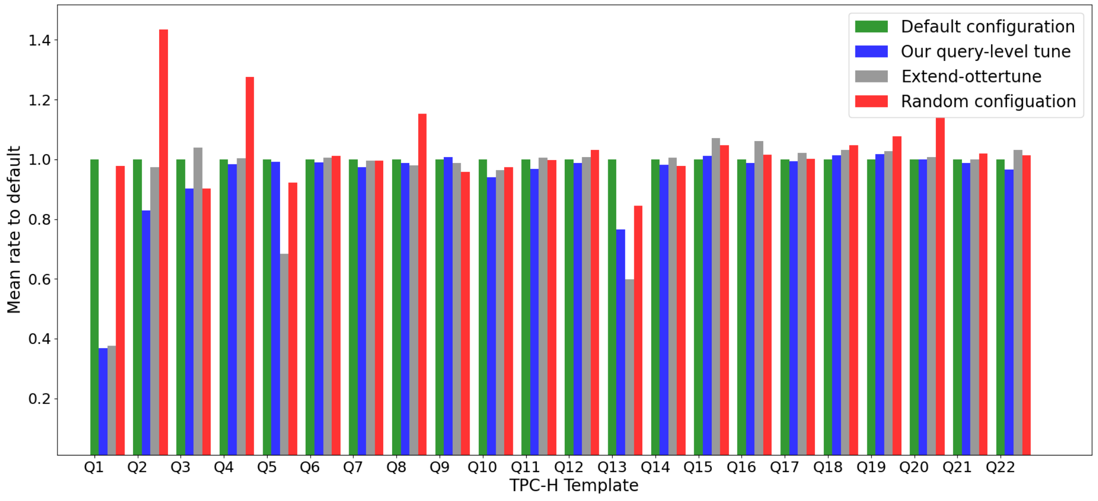

Fig. 6. Comparison of execution latency

data. The initial data was divided into training set and test according to the ratio of 9 to 1. set. For the QPPN model, we regenerated 765 query statements on TPC-H. Before executing a query, we used the knob sampling method to generate a new random knob configuration and applied it to the target database. The parameters of the LHS sampling algorithm were 100, and a total of 25 rounds were executed. , each round takes about 8 hours to obtain 19125 initial data. Similarly, the initial data is divided by a ratio of 9 to 1 as a training set and a test set. In order to verify the overall effect of the knob tuning tool, we generated 269 new queries on TPC-H, and these queries to be tuned have no intersection with the above training model queries.

C. Evaluation on Tuning Performance

we verified the superiority of this method by comparing and analyzing the query latency under the knob configuration generated by other methods. Including the default configuration of CockroachDB V21.1, randomly generated configurrtion, and our previous work of extend-Ottertune method. Figure 6 shows the results and we can see the following points: (1)Using the knob configuration recommended by our tuning system, its query performance is significantly better than the default configuration or random configuration, and the average reduction ratio of all types of query latency relative to the default configuration is 9.2%; (2) Among all kinds of queries, Q1 type queries have the most obvious latency under our recommended configuration, which is 63%; Q1, Q2, Q3 and Q13 have the latency reduction of more than 10%; a few types of queries have slightly increased latency (no more than 2%), such as Q9 and Q19; (3) Compared with the workload-level tuning method, the average reduction ratio of all types of query latency relative to the default configuration is 6.5%, which is lower than 9.2% of our system. In addition, among all 22 types of queries, extend-Ottertune can only improve the performance of 7 types, which is far lower than the 18 types of our method.

For all kinds of queries, our system recommends a configuration with an overhead between 106 and 230ms.

IX. Related Work

A. Knob Tuning System

Database knob tuning has been a difficult task since databases often provide hundreds of knobs to control various modules in the system [9]. Traditional DBAs often spend a lot of manpower and time on tuning [10], and cannot satisfy increasingly complex databases. Therefore, experts are increasingly inclined to use machine learning methods to deal with this difficult task [6, 7]. The first to appear is the knob tuning method based on experience and heuristic algorithm. For example, IBM provides performance monitoring tools for its database DB2 [2], Oracle develops similar performance monitoring tools for its databases [3], and Microsoft develops tools for SQL Server databases [4]. However, the above experience-based knob tuning methods have obvious limitations, cannot be adapted to other databases and are not automated enough. To solve this problem, the research on applying machine learning methods to knob tuning has attracted more and more attention. Ottertune [10] is the first large-scale automated knob tuning system using machine learning methods proposed by Carnegie Mellon University, and supports a variety of traditional relational databases, including PostgreSQL, MySQL, and Oracle. CDBTune [5, 12] is an end-to-end DBMS configuration automatic adjustment system proposed by Tencent, which uses deep reinforcement learning for database knob tuning and can recommend better knob configurations in complex cloud environments. On the basis of CDBTune, Tsinghua University and Huawei proposed QTune [11]. QTune also uses the deep reinforcement learning method, but unlike CDBTune, it supports three granularity knob tuning of query-level, cluster-level and workload-level. The above work has strong generalization ability, but it is mainly aimed at knob tuning tasks in traditional relational database scenarios under workload level. Among them, although QTune supports query level, it still has two shortcomings: the number of encoded query features is small; it does not support knob tuning for distributed databases.

B. Query Performance Prediction

The ability to estimate query execution time is crucial for many tasks in database systems. From a technical perspective, query performance prediction is mainly divided into methods based on statistical machine learning [13–17] and methods based on deep neural networks [18, 21]. Archana Ganapathi et al. proposed a method for predicting multiple indicators during query execution using a clustering model [14]. The performance indicators of the target query were calculated using the performance indicator information of the nearest neighbors. Li et al. proposed a robust resource estimation method, whose statistical model [17] is used to replace the labor cost model used in the query optimizer. Ryan Marcus et al. proposed a neural network-based method [18], which introduced a new neural network structure specifically to solve the query performance prediction problem. However, the above methods do not fully consider the impact of hardware configuration and knob changes on query latency, so they cannot be applied to our query-level knob tuning work.

X. Conclusion

We propose a query-level knob tuning system for distributed database, which can recommend different knob configurations for different queries. Experimental results show our system reduces latency by an average of 9.2% under TPC-H workload and perform better than other tuning systems. We believe it lays the foundation for future joint-cloud tuning scenarios. We will adapt more databases to our system, and optimization work of tuning performance is under way.

Acknowledgment

This work was supported by the National Key RD Program of China No. 2021YFB2600301 and Youth Innovation Promotion Association at CAS.

References

[1] A. Pavlo and M. Aslett, "What's really new with newsql?" ACM Sigmod Record, vol. 45, no. 2, pp. 45–55, 2016.

[2] E. Kwan, S. Lightstone, A. Storm, and L. Wu, "Automatic configuration for ibm db2 universal database," Proc. of IBM Perf Technical Report, 2002.

[3] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood, "Automatic performance diagnosis and tuning in oracle." in CIDR, 2005, pp. 84–94.

[4] D. Narayanan, E. Thereska, and A. Ailamaki, "Continuous resource monitoring for self-predicting dbms," in 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. IEEE, 2005, pp. 239–248.

[5] J. Zhang, K. Zhou, G. Li, Y. Liu, M. Xie, B. Cheng, and J. Xing, " CDBtune$^+$: An efficient deep reinforcement learningbased automatic cloud database tuning system, The VLDB Journal, vol. 30, no. 6, pp.959987, 2021.

[6] I. Trummer, J. Wang, Z. Wei, D. Maram, S. Moseley, S. Jo, J. Antonakakis, and A. Rayabhari, "Skinnerdb: Regret-bounded query evaluation via reinforcement learning," ACM Transactions on Database Systems (TODS), vol. 46, no. 3, pp. 1–45, 2021.

[7] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi, "Learning state representations for query optimization with deep reinforcement learning," in Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, 2018, pp. 1–4.

[8] S. Duan, V. Thummala, and S. Babu, "Tuning database configuration parameters with ituned," Proceedings of the VLDB Endowment, vol. 2, no. 1, pp. 1246–1257, 2009.

[9] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang, "Bestconfig: tapping the performance potential of systems via automatic configuration tuning," in Proceedings of the 2017 Symposium on Cloud Computing, 2017, pp. 338–350.

[10] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in Proceedings of the 2017 ACM international conference on management of data, 2017, pp. 1009–1024.

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

[12] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu et al., "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in Proceedings of the 2019 International Conference on Management of Data, 2019, pp. 415–432.

[13] M. Akdere, U. Cetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, "Learningbased query performance modeling and prediction," in 2012 IEEE 28th International Conference on Data Engineering. IEEE, 2012, pp. 390–401.

[14] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in 2009 IEEE 25th International Conference on Data Engineering. IEEE, 2009, pp. 592–603.

[15] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacig¨um¨us, and J. F. Naughton, "Predicting query execution time: Are optimizer cost models really unusable?" in 2013 IEEE 29th International Conference on Data Engineering (ICDE). IEEE, 2013, pp. 1081–1092.

[16] N. Zhang, P. J. Haas, V. Josifovski, G. M. Lohman, and C. Zhang, "Statistical learning techniques for costing xml queries," in Proceedings of the 31st international conference on Very large data bases, 2005, pp. 289–300.

[17] J. Li, A. C. K¨onig, V. Narasayya, and S. Chaudhuri, "Robust estimation of resource consumption for sql queries using statistical techniques," arXiv preprint arXiv:1208.0278, 2012.

[18] R. Marcus and O. Papaemmanouil, "Plan-structured deep neural network models for query performance prediction," arXiv preprint arXiv:1902.00132, 2019.

[19] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," Biological cybernetics, vol. 59, no. 4, pp. 291– 294, 1988.

[20] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," science, vol. 313, no. 5786, pp. 504–507, 2006.

[21] X. Zhou, J. Sun, G. Li, and J. Feng, "Query performance prediction for concurrent queries using graph embedding," Proceedings of the VLDB Endowment, vol. 13, no. 9, pp. 1416–1428, 2020.

[22] Cockroach Labs, the company building CockroachDb. [Online]. Available: https://www.cockroachlabs.com/

[23] A. J. Storm, C. Garcia-Arellano, S. S. Lightstone, Y. Diao, and M. Surendra, "Adaptive self-tuning memory in db2," in Proceedings of the 32nd international conference on Very large data bases, 2006, pp. 1081–1092.

[24] D. A. Stewart, "Proceedings of the 2003 conference of the centre for advanced studies on collaborative research," 2003.

[25] R. Marcus and O. Papaemmanouil, "Deep reinforcement learning for join order enumeration," in Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2018, pp. 1–4.

[26] X. Yu, G. Li, C. Chai, and N. Tang, "Reinforcement learning with tree-lstm for join order selection," in 2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, 2020, pp. 1297–1308.

[27] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica, "Learning to optimize join queries with deep reinforcement learning," arXiv preprint arXiv:1808.03196, 2018.

[28] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," arXiv preprint arXiv:1809.00677, 2018.