EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

RBOIRA: Integrating Rules and Reinforcement Learning to Improve Index Recommendation

Wenbo Yu[1], Jinguo You[1,2,*], Xiangyu Niu[1], Jianfeng He[1,2] and Yunwei Zhang[1,2]

[1] Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming, 650500, Yunnan, China
[2]Yunnan Key Laboratory of Artificial Intelligence, Kunming University of Science and Technology, Kunming, 650500, Yunnan, China

Abstract

INTRODUCTION: The index is one of the most effective ways to improve the database query performance. The expert-based index recommendation approach cannot adjust the index configuration in real time. At the same time, reinforcement learning can automatically update the index and improve the recommended configuration by leveraging expert experience.
OBJECTIVES: This paper proposes the RBOIRA, which combines rules and reinforcement learning to recommend the optimal index configuration for a set of workloads in a dynamic database.
METHODS: Firstly, RBOIRA designed three heuristic rules for pruning index candidates. Secondly, it uses reinforcement learning to recommend the optimal index configuration for a set of workloads in the database. Finally, we conducted extensive experiments to evaluate RBOIRA using the TPC-H database benchmark.
RESULTS: RBOIRA recommends index configurations with superior performance compared to the baselines we define and other reinforcement learning methods used in related work and also has robustness in different database sizes.

Keywords: index recommendation, heuristic rules, dynamic database, reinforcement learning

Received on 04 September 2023, accepted on 15 September 2023, published on 18 September 2023

doi: 10.4108/eetsis.3822

*Corresponding author. Email: jgyou@126.com

1. Introduction

The advent of big data has significantly impacted the query efficiency of massive data in traditional relational databases. In database optimization, there are many different ways to improve the query performance of a database, such as by creating indexes and materializing views. Creating appropriate indexes for a set of workloads in the database can effectively improve query performance [1]. The approach to index creation is no longer limited to a traditional manual approach that relies on the expertise and experience of the database administrator (DBA) to recommend index configurations for a set of workloads in a database. With the rise of machine learning in various research fields [2, 3], it is gradually being applied to the self-tuning of databases, such as using reinforcement learning to recommend the optimal index configuration [3] and join order selection for query statements for a set of workloads in a database [4, 5].

The most typical traditional index recommendation for relational databases is expert-based, with some limitations: Firstly, it targets static databases to create an index. Secondly, it cannot update the index configuration on time. Although using reinforcement learning to recommend index configurations is better than the traditional approach. Therefore, this paper optimizes the following two aspects:

(1) Many index candidates are trained, which reduces the training efficiency of the algorithm with the recommended suboptimal index configuration because many related works extract index candidates from

databases [6] or workloads [7] (such as **group by** or **order by**) without pre-processing, which reduces the training efficiency of the algorithm and increases the execution cost.

(2) Part of the related work still only targets static databases without considering the actual production environment of databases, which leads to recommending suboptimal index configuration.

Based on the above, this paper proposes the RBOIRA, which combines rules and reinforcement learning to recommend the optimal index configuration for a set of workloads in a dynamic database. The RBOIRA execution steps include two main steps: firstly, it prunes the index candidates using designed heuristic rules. Secondly, it uses reinforcement learning to update the indexes in real time, enabling the recommendation of the optimal index configuration for a set of workloads in a dynamic database.

In summary, our work contributes the following:

(1) Three heuristic rules are proposed for the pruning index candidates, significantly reducing the dimensions of action and state space, which improves execution efficiency and reduces the execution cost.

(2) RBOIRA integrates rules and reinforcement learning to configure and update indexes in real time for a set of workload recommendation indexes under a dynamic database.

(3) We conducted extensive experiments to evaluate RBOIRA's performance using the TPC-H database benchmark. Experimental results show that RBOIRA recommends index configurations with superior performance to comparison methods and it also has some robustness in different database sizes.

2. Related work

Machine learning (ML) is currently applied to many fields, such as a distributed cooperative coevolutionary genetic algorithm to optimize multi-objective data publishing [8], optimize stragglers in edge federated learning (EFL) [9], and uncertain data query [10]. In recent years, machine learning has been continuously integrated into traditional relational databases or NoSQL databases to implement components for automation and self-optimization [11-13]. The limitations of traditional tuning methods can be effectively addressed by using ML techniques, which significantly promote the development of AI4DB [11, 12]. Moreover, researchers can consider enhancing database performance using hardware devices [14].

Database tuning can be divided into external tuning and internal tuning [15, 16]. Configuring the database management system through the application programming interface (API) is called external tuning, and embedding algorithms into the database management system is called internal tuning. Integrating ML technology into index selection is part of internal tuning and is one of the most critical parts of achieving database self-tuning [17, 18]. For example, Ge et al. [19] designed a distributed prediction-randomness framework for the evolutionary dynamic multiobjective partitioning optimization of databases.

Creating an index will directly affect the database's query performance and transaction load-handling capabilities [20]. Especially when the database system processes transactions, creating indexes on corresponding columns can improve the query efficiency of the database [21]. Jan et al. [22] described and analyzed 8 traditional index selection methods and compared them in different dimensions, such as the time complexity of the algorithm. They also designed a system that could choose the right index for different situations. On the contrary, Ding et al. [23] proposed a method to improve the query efficiency of workload in the database by using data rules, which improved the query performance to some extent. Sadri et al. [24] designed a deep reinforcement learning algorithm to minimize the total execution cost of the workload in the cluster to recommend the best index configuration for the cluster database, where each replica database can recommend the same index configuration or a different index configuration.

Lan et al. [7] proposed five heuristic rules to extract the corresponding index candidates from the workload, which significantly reduced the dimensionality size of the action space and state space, and designed a deep reinforcement learning algorithm to recommend the index, but the shortcoming was that it only targeted at static databases and could not update the index configuration in time. Licks et al. [6] designed SmartIX that is a reinforcement learning algorithm to recommend the index configuration for the next set of workloads in a dynamic database. The limitation of this method is that it does not prune the index candidates extracted from the database, but takes all columns as the index candidates, resulting in low efficiency of algorithm training, so it recommends the second-best index configuration. Sharma. et al. [25] designed MANTIS which uses a deep reinforcement learning algorithm to implement index type recommendation and index selection. However, MANTIS still neglected to further process the index candidates, it did not screen the candidates according to the characteristics of the data in the database, which resulted in long algorithm training time and high cost.

3. Methodology

3.1. The architecture of RBOIRA

By constructing its architecture to clarify further the execution process of each functional module of RBOIRA, as shown in Figure 1. The description of each functional module is detailed as follows:

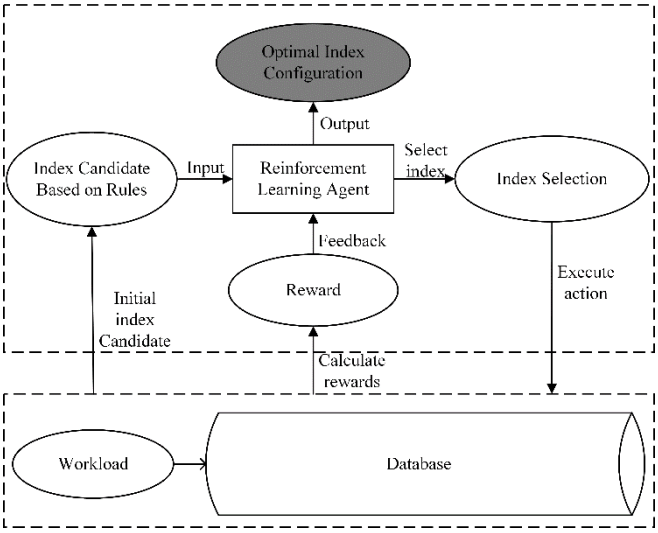

Figure 1. The architecture of RBOIRA

(1) Workload module. It is responsible for providing query statements and embedding them into the relational database for later running by script functions.

(2) Index candidate based on rules module. Heuristic rules prune the index candidates extracted from the database table and input them into the reinforcement learning agent module as an input stream.

(3) Reinforcement learning agent module. It is responsible for training the algorithm, including updating the action parameters, updating the status, and selecting the action by using strategies and other operations.

(4) Index selection module. Execute the operation of creating an index and deleting an index.

(5) Reward module. The reward for the action is calculated by the reward function and then feedback to the agent.

(6) Recommended optimal index configuration module. The module recommends the index configuration in the last training episode as the optimal index configuration.

3.2. Heuristic rules

The heuristic rules are defined as follows:

Rule 1 (R1): When the data volume is the smallest and much smaller than other tables (at least one scale is 10^3), the columns in this table are not considered index candidates.

Rule 2 (R2): The column of the longest type in the database table is not recommended to be indexed which is not considered an index candidate.

Rule 3 (R3): A column with too many duplicate values or null values is not considered an index candidate, but must meet two conditions as follows:

Condition 1: It is the minimum selectivity of a column in the table.

Condition 2: Selectivity is less than 20%. This column is not considered an index candidate if and only if both conditions are met.

The above three rules are applicable rules defined based on database index optimization experience. The reasons for selecting them are as follows:

For R1: (i) Indexes occupy storage space and require maintenance. The additional storage space and maintenance costs of establishing indexes on small data tables outweigh the performance advantages. (ii) For small tables, the database query optimizer may choose a full table scan instead of using an index to improve query performance because it may be faster.

For R2: (i) Building indexes on long fields consumes larger storage space and increases index maintenance costs. (ii) It may hurt query performance and does not work with all database storage engines.

For R3: (i) Selectivity measures the number of distinct values in the index column relative to the total number of rows. If the selectivity is very small, that is, most of the values in the index columns are the same, then the index will provide a limited filtering effect and cannot bring about performance improvement. (ii) The query optimizer may not be able to use this index, and it may be difficult to maintain in a highly concurrent database environment.

In addition, the index candidates are pruned by each rule, as shown in Table 1.

Table 1. Pruned Index candidates under each rule

Rule	Pruned index candidates
R1	n_name, n_comment, r_name, r_comment
R2	p_name, ps_comment, n_comment, r_comment, c_comment, s_comment, o_comment, l_comment
R3	p_mfgr, ps_availqty, l_linestatus, c_mktsegment, o_shippriority

3.3. The algorithm of RBOIRA

The basic design of the pseudo-code of the RBOIRA is shown in algorithm 1.

Algorithm 1: Index recommendation of RBOIRA

Input: Index candidate set DBS, a set of workloads W.

Output: Optimal Indexing Configuration OIC.

1. Initializing the environment Env
2. While $episode > 0$ **do**
3. Initializing the environment Env
4. Extract index candidates from the database based on rules and map to the initial state of S
5. While $step > 0$ **do**

6. The agent selects action A based on the current state S_t and ε -greedy policy
7. Execute A to get the next state S_{t+1} and reward R
8. Update parameters θ of A
9. Store (S_t, R, A, S_{t+1}) in the experience pool E
10. Extract mini-batch data from E to perform experience playback
11. $S_t = S_{t+1}$
12. **end**
13. $\varepsilon = \varepsilon - \varepsilon * 0.1$
14. end

The execution steps are as follows: Step 1, initializes the environment, including the initialization state and action parameters, in which α, γ and ε are set as 0.01, 0.85, and 0.9. Step 2 to step 12 into each episode and set it to 30. Step 3, initialize the environment again to delete all indexes. Step 4, prune index candidates based on heuristic rules. Step 5 to step 11 into each step and set it to 100. Step 6, select an action based on ε -greedy and the current state S_t. Step 7 executes the action to update the next current state and get a reward. Step 8 updates the parameters of action [6]. Step 9 store (S_t, R, A, S_{t+1}) in the experience pool. Step 10, random extract mini-batch data used to train. Step 11 updates the current state. Step 12 end of steps. Step 13 update ε . Step 14 end of episodes and output OIC.

4. Experiments

4.1. Experimental setup

Experimental environment. All experiments are conducted in an 8-core AMD R7-5800X CPU @ 3.80 GHz and 32 GB of RAM running Ubuntu Linux 18.04 and Python 3.6.6.
Dataset and workload. We use the TPC-H tool to generate a 1 GB dataset (containing 8 relation tables and 8,661,245 records as shown in Table 2 stored in MySQL and generate a workload consisting of 22 queries with different levels of complexity.
Index candidates. All index candidates are extracted from the database tables and pruned by our three heuristic rules. The number of index candidates is pruned from the initial 45 to the final 30 index candidates for model training.

Table 2. The description of 8 relational tables

Table	Record	Description
CUSTOMER	150000	Stores information about customers who purchase parts from the SUPPLIER, including name, address, etc.
LINEITEM	6001215	Stores all the customer's order details and component information, including order number, part number, etc.
NATION	25	Stores information about some countries in the world, including country names, etc.
ORDERS	1500000	Store order information, including the order number, customer number, price, etc.
PART	200000	Stores part information provided by the SUPPLIER, including part number, part name, part type, and so on.
PARTSUPP	800000	Stores the supply information between the PART and the SUPPLIER, including the part number, supplier name, etc.
REGION	5	It stores information about the world's five continents, including their names.
SUPPLIER	10000	Stores the supplier information of the parts, including the supplier name, supplier address, etc.

4.2. Evaluation metric

The TPC-H is a well-known non-profit organization that creates database performance benchmarks [26]. We get outputs from three metrics based on the TPC-H benchmark: $Power@Size$, $Throughput@Size$, and $QphH@Size$. While $QphH@Size$ is obtained by computing $Power@Size$ and $Throughput@Size$ metrics. The $Power@Size$ evaluates how fast the DBMS computes the answers to single queries. This metric is computed using formula (1):

$$Power@Size = \frac{3600}{\sqrt[24]{\prod_{i=1}^{22} QI(i,0) \times \prod_{j=1}^{2} RI(j,0)}} \times SF \quad (1)$$

Where SF is the scale factor or database size, $QI(i,0)$ is a set of query streams, $RI(j,0)$ is a refresh function, and 3600 is the number of seconds per hour.

The $Throughput@Size$ measures the ability of the system to process the most queries in the least amount of time, taking advantage of I/O and CPU parallelism. It evaluates the system's performance against a multi-user

workload that is completed in a set amount of time, using the formula in (2):

$$Throughput@Size = \frac{S \times 22}{T_s} \times 3600 \times SF \quad (2)$$

Where S is the number of query streams executed, and T_s is the total time required to run the throughput test for s streams.

Equation (3) depicts the Queries-per-Hour Performance $QphH@Size$ metric, calculated by taking the geometric mean of the previous two metrics and reflecting various aspects of a database's query processing capability.

$$QphH@Size = \sqrt{Power@Size \times Throughput@Size} \quad (3)$$

The ACT is used to compare the time consumption of the algorithm, which reflects the efficiency of the algorithm, as equation (4). Where $The\ Total\ Costing\ Time$ represents the total training time of the algorithm, and $The\ Number\ Of\ Episodes$ represents the total training episodes.

$$ACT = \frac{The\ Total\ Costing\ Time}{The\ Number\ Of\ Episodes} \quad (4)$$

The $Selectivity$ is used to control the selectivity of attributes, as shown in equation (5). Col represents the corresponding column; $Count(*)$ represents the total number of rows in the column in the table. $Distinct\ Count(*)$ indicates the cardinality of the column.

$$Selectivity = \frac{Col(Distinct\ Count(*))}{Col(Count(*))} \times 100\% \quad (5)$$

4.3. Baseline

To make a full and comprehensive experimental comparison of the proposed RBOIRA method, this section describes in detail the self-defined comparison baseline and the related method of using reinforcement learning algorithm to implement database index recommendation, as follows:

1. Initial_State: It is the default TPC-H configuration and contains only the indexes on the primary and foreign keys.
2. Expert-Based: Indicates the index configuration based on expert suggestions.
3. ALL-S: Indicates the index configuration in which all columns in the database are indexed.
4. ALL-R: Using heuristic rules prune single-attribute index candidates, and indexes are built on the remaining index candidates.
5. SmartIX [6]: A real-time creation and deletion of single-attribute index candidates in a dynamic database to recommend the optimal index configuration.
6. DQN-S [7]: The deep reinforcement learning algorithm in the Index Advisor method is used to recommend the single-attribute index configuration for a set of workloads in the database.
7. NoDBA [27]: A system based on cross-entropy deep reinforcement learning method used to recommend the best index configuration for a given workload in a set of databases.
8. POWA [28]: Index configuration recommended by PostgreSQL Workload Analyser which is an open-source index recommendation tool.
9. ITLCS [29]: An index tuning and learning classifier, which combines a learning classifier and genetic algorithm to make efficient index configuration recommendations.

5. Experimental results and analysis

5.1. Model Training

The RBOIRA model was trained based on algorithm 1, where the training episodes of RBOIRA are set to 30, and the training steps for each episode are set to 100. The index configuration corresponding to the maximum reward of the last training episode is taken as the optimal index configuration and its convergence is shown in Figure 2.

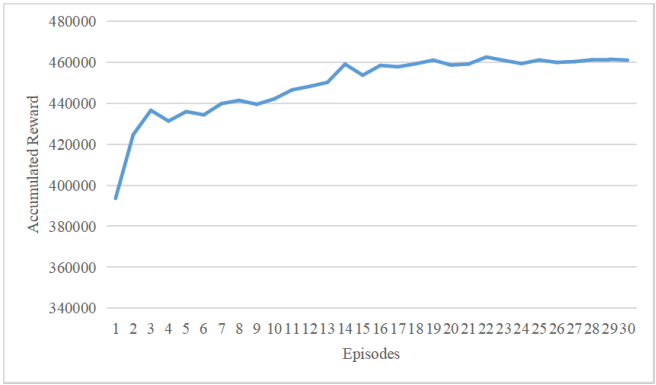

Figure 2. Accumulated reward per episode

Besides, we also compare the training time consumption of SmartIX and RBOIRA under the same experimental environment and the same training episodes and calculate their respective ACT, as shown in Figure 3. The ACT of SmartIX and RBOIRA are 3894.48s and 2839.74s, respectively. RBOIRA is about 1.37 times more effective than SmartIX. By analyzing the experimental results of Figure 3, it can be found that there is one main reason.

RBOIRA's index candidates only account for the total number of index candidates in SmartIX, which greatly reduces the dimensions of state space and action space, improving agent learning efficiency and reducing cost.

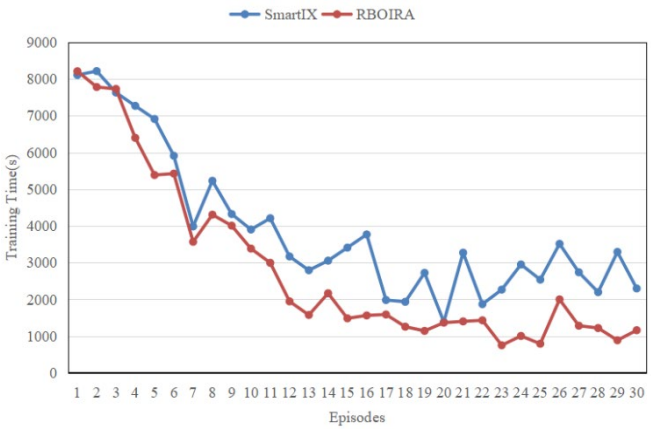

Figure 3. Algorithm time efficiency comparison

As can be seen from Figures 2 and 3, although the reinforcement learning algorithm 1 finally successfully converged and the improved algorithm was more efficient, it still consumed huge time costs and hardware costs. In addition, the dimensional space explored by the agent is still relatively large, which increases the complexity of reinforcement learning to explore the optimal solution.

5.2. Incremental data experiments

It conducts an incremental data experiment to prove the robustness of RBOIRA in different data sizes. Data description, query performance, and TPC-H benchmark are shown in Table 3, Figure 4, and Table 4, respectively. Table 3 includes 10 levels of data sizes to simulate data sizes. The data size ranges from 100MB to 1GB, and the experimental data is increased by 100MB each time. A detailed description of the number of records for tables is shown in Table 3.

By analyzing the experimental results in Figure 4 and Table 4, we can find that the query performance gradually stabilizes as the data size gradually increases where the high performance at the beginning is caused by small-volume data. This is mainly because when the size of data in the database is too small, even creating primary and foreign keys in the database still brings significant performance improvement to the database. On the contrary, as the size of database data gradually increases, reinforcement learning dynamically adjusts the index configuration according to the current data changes in the database, ensuring that the query performance of the database does not fall dramatically and keeps the database query performance stable. Therefore, this incremental data analysis experiment further illustrates the robustness of RBOIRA in different data sizes.

Table 3. Description of the number of data records in different data sizes

Data Size	CUSTOMER	LINEITEM	NATION	ORDERS	PART	PARTSUPP	REGION	SUPPLIER
100MB	15000	600572	25	150000	20000	80000	5	1000
200MB	30000	1199969	25	300000	40000	160000	5	2000
300MB	45000	1800093	25	450000	60000	240000	5	3000
400MB	60000	2399740	25	600000	80000	320000	5	4000
500MB	75000	2999671	25	750000	100000	400000	5	5000
600MB	90000	3601036	25	900000	120000	480000	5	6000
700MB	105000	4200337	25	1050000	140000	560000	5	7000
800MB	120000	4800841	25	1200000	160000	640000	5	8000
900MB	135000	5400556	25	1350000	180000	720000	5	9000
1GB	150000	6001215	25	1500000	200000	800000	5	10000

Table 4. TPC-H performance of RBOIRA under different data sizes

Data Size	Power@Size	Throughput@Size	QphH@Size
100MB	11258.71	5582.72	7928.06
200MB	8760.91	5299.35	6813.75
300MB	8435.62	4683.21	6285.36
400MB	8157.86	4017.93	5725.18
500MB	7629.05	3966.45	5500.93
600MB	7767.71	3999.58	5573.83
700MB	7835.42	3939.52	5555.88
800MB	7720.88	3944.81	5518.83
900MB	7609.50	3890.23	5440.84
1GB	7616.37	3836.23	5405.38

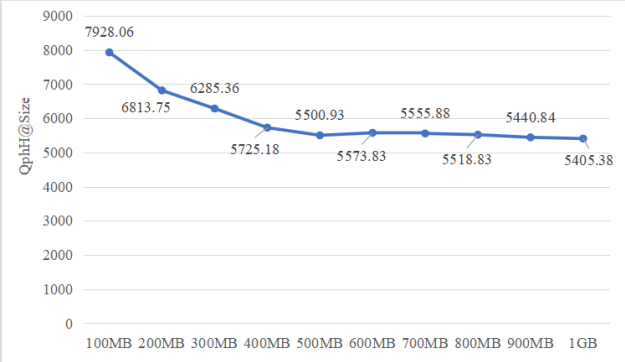

Figure 4. Query performance of RBOIRA in different data sizes

5.3. Performance comparison experiment

We conduct a query performance comparison experiment and index size experiment, and the results are analyzed in comparison. Experiment results are shown in Figures 5 and 6 respectively. It also gives the detailed experimental results of each baseline configuration in the TPC-H standard database, shown in Table 5.

The analysis of the performance comparative experimental results in Figure 5, shows that RBOIRA outperforms the other comparison methods on query performance. By analyzing the experimental results in Figure 5, it is clear that the index configuration recommended by RBORIA outperforms the other baselines in terms of QphH. The following two reasons mainly explain this:

(1) Compared to traditional methods such as Expert-Based, ALL-S, and POWA, RBOIRA maximizes the query performance of a set of workloads in the dynamic database by using reinforcement learning methods to constantly update index configurations, which helps reinforcement learning agents find better index configurations.

(2) Compared to machine learning methods such as SmartIX, DQN-S, and ITLCS, RBOIRA uses rules to prune the initial index candidates. This operation effectively improves the learning efficiency of the agent and reduces the execution cost of the algorithm, thus helping the agent explore better index configuration.

Besides, by comparing and analyzing the experimental results in Figure 6, it is found that although RBOIRA does not have the smallest of all methods on index size, it's still much smaller than other baselines like ALL-S and SmartIX. Although the index size of RBOIRA is somewhat larger than that of POWA, DQN-S, and NoDBA, RBORIA improves query performance by sacrificing less space, and the swap of space for performance is adequate.

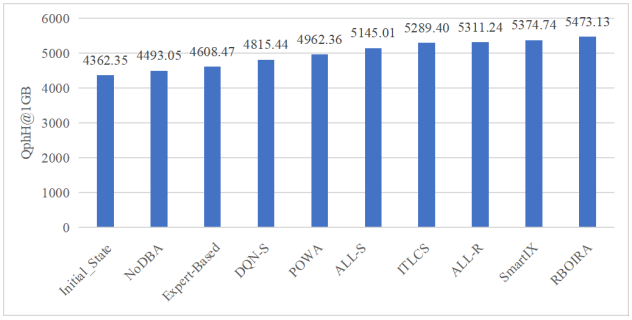

Figure 5. Hourly query performance

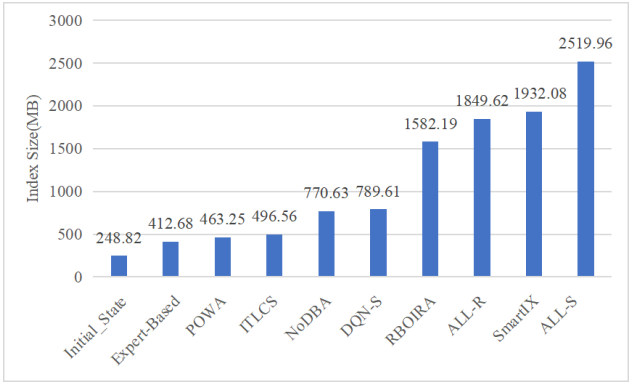

Figure 6. Index size

Table 5. TPC-H performance comparison

Index Config	Power@1GB	Throughput@1GB	QphH@1GB	Index Size
Initial_State	5640.16	3374.04	4362.35	248.82
NoDBA	5827.56	3464.15	4493.05	770.63
Expert-Based	6190.36	3430.81	4608.47	**412.68**
POWA	6826.56	3607.24	4962.36	463.25
DQN-S	6515.25	3559.16	4815.44	789.61
ALL-S	7252.57	3650.08	5145.00	2519.96
ITLCS	7261.62	3852.82	5289.40	496.56
ALL-R	7440.80	3791.19	5311.24	1849.62
SmartIX	7583.89	3809.10	5374.74	1932.08
RBOIRA	**7700.02**	**3890.26**	**5473.13**	**1582.19**

5.4. Ablation experiment and selectivity experiment

We conducted the TPC-H performance test experiments for each rule to verify the effectiveness of a single rule and analyzed its experimental results. The experimental results are shown in Table 6, including index size.

Analyzing the experiment results in Table 6 gives the following conclusions:

(1) Compared with ALL-S, R1 has no significant change in TPC-H query performance and index size, which indicates that when the capacity of data in the table is too small, the create index or not create index does not have a significant impact on the database query performance and space consumption proving the rationality of R1.

(2) Although R2 has decreased performance compared with ALL-S, it has significantly decreased index size, indicating that less storage space is consumed. R2's index size was significantly reduced when it slightly affected query performance degradation, indicating that this replacement operation is worthwhile, which proves the rationality of the R2.

(3) Compared with ALL-S, R3 outperforms ALL-S in both TPC-H performance and index size, which proves the validity of R3.

(4) Compared with R1, R2, R3, and ALL-S, ALL-R has better query performance and a smaller index size, which indicates the rationality of the integration of the three rules.

Table 6. TPC-H performance comparison and index size under different rules

Index Config	Power@1GB	Throughput@1GB	QphH@1GB	Index Size
ALL-S	7252.57	3650.08	5145.00	2519.96
R1	7185.83	3668.22	5135.24	2512.88
R2	7001.01	3629.27	5040.66	1992.43
R3	7397.14	3750.99	5267.47	2373.18
ALL-R	**7440.80**	**3791.19**	**3791.19**	**1849.62**

Furthermore, we also conducted a selectivity threshold experiment for R3, with experimental results as shown in Figure 7, where selectivity is calculated by Formula 5. As the selectivity increases, more and more index candidates are pruned, leading to the deterioration of database query performance. As the selectivity gradually increases, the index candidates extracted from the database have a higher selectivity. However, because the attributes themselves are too long (they do not meet R1), taking the index scan consumes more time than taking the full table scan, thus reducing the query efficiency. As Figure 7 shows, if the selectivity is within the range of 0.2, the query performance is still close to the Initial_State.

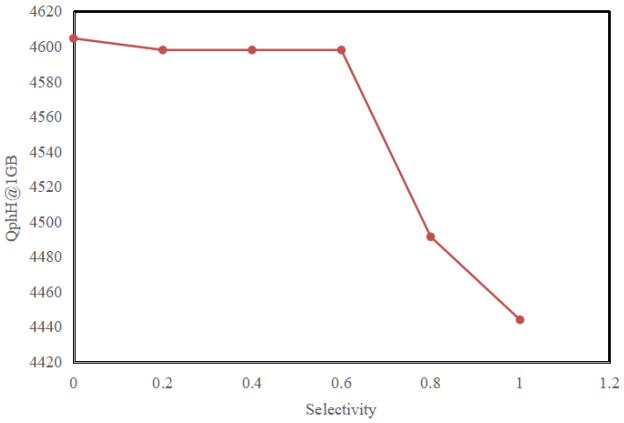

Figure 7. Comparison experiment of selectivity threshold

This situation indicates that the selectivity of 0.2 can reduce the number of index candidates and ensure that the database query performance does not change significantly.

6. Conclusion

In this paper, we propose RBOIRA which is a practical and flexible index advisor that integrates three heuristic rules and reinforcement learning to recommend the optimal index configuration for a set of workloads in a dynamic database. We have designed extensive experiments to prove the superiority of RBOIRA, and the experimental results show that RBOIRA outperforms other existing related methods.

The RBOIRA still has some areas that can be optimized, such as improving the algorithm efficiency of reinforcement learning and reducing the resource usage required for intelligent index recommendation. In the future, we will improve the efficiency and reduce the cost of the reinforcement learning recommendation index.

Declarations

Ethical Approval.
Not Applicable

Data Availability Statements.
The data of TPC-H that support the findings of this study are openly available at: http://www.tpc.org/, reference number [26].

Competing interests.
The authors have no competing interests to declare that are relevant to the content of this article.

Funding.
This work was partially supported by the Natural Science Foundation of China (NO.62062046), CCF Opening Project of Information System (NO.HZ2021F0055A).

Author Contributions.
Yu - Conceptualization, Methodology, Software, Writing Original draft preparation and Editing. You - Conceptualization, Supervision, Reviewing, and Editing. He, Zhang, and Niu - Reviewing.

Acknowledgements.
Thanks to all the authors for their help in creating the revised manuscript.

References

[1] Ramakrishnan R, Gehrke J. Database management systems (3. ed.)[M]. DBLP,2003.
[2] Li Y. Deep Reinforcement Learning: An Overview[J]. 2017.
[3] Lahdenmaki T, Leach M. Relational Database Index Design and the Optimizers: DB2, Oracle, SQL Server. John Wiley \& Sons, 2005.
[4] Tan J, Zhang T, Li F, et al. ibtune: Individualized buffer tuning for large-scale cloud databases[J]. Proceedings of the VLDB Endowment, 2019, 12(10): 1221-1234.
[5] Marcus R, Papaemmanouil O. Deep reinforcement learning for join order enumeration[C]//Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management. 2018: 1-4.
[6] Paludo Licks G, Colleoni Couto J, de Fátima Miehe P, et al. SMARTIX: A database indexing agent based on reinforcement learning [J]. Applied Intelligence, 2020, 50(8):2575-2588.
[7] Lan Hai, Bao Zhifeng, Peng Yuwei. An Index Advisor Using Deep Reinforcement Learning. CIKM '20: The 29th ACM International Conference on Information and Knowledge Management. ACM, 2020.
[8] Sultana K, Ahmed K, Gu B, et al. Elastic Optimization for Stragglers in Edge Federated Learning[J]. Big Data Mining and Analytics, 2023, 6(4): 404-420.
[9] Ge Y F, Bertino E, Wang H, et al. Distributed Cooperative Coevolution of Data Publishing Privacy and Transparency[J]. ACM Transactions on Knowledge Discovery from Data, 2023, 18(1): 1-23.
[10] Wang Bin, Zhu Rui, Luo Shiying, et al. H-MRST: A Novel Framework for Supporting Probability Degree Range Query using Extreme Learning Machine[J]. Cognitive Computation, 2017, 9(1): 68-80.
[11] Li Guoliang, Zhou Xuanhe, Cao Lei. AI Meets Database: AI4DB and DB4AI. Proceedings of the 2021 International Conference on Management of Data. 2021: 2859-2866.
[12] Li GL, Zhou XH. XuanYuan: An AI-native Database Systems[J]. Journal of Software, 2020, 31(3): 831-844.
[13] Yan Yu, Yao Shun, Wang Hongzhi, et al. Index Selection for NoSQL Database with Deep Reinforcement Learning[J]. Information Sciences, 2021, 561: 20-30.
[14] Pei W, Li Z H, Pan W. Survey of key technologies in GPU database system. Ruan Jian Xue Bao[J]. Journal of Software, 2021, 32(3): 859-885.
[15] Van Aken D, Pavlo A, Gordon G J, et al. Automatic Database Management System Tuning Through Large-scale

Machine Learning. Acm International Conference on Management of Data. ACM, 2017:1009-1024.
[16] Pavlo A, Butrovich M, Joshi A, et al. External vs. Internal: An Essay on Machine Learning Agents for Autonomous Database Management Systems[J]. IEEE bulletin, 2019, 42(2).
[17] Welborn J, Schaarschmidt M, Yoneki E. Learning Index Selection with Structured Action Spaces[J]. arXiv preprint arXiv:1909.07440, 2019.
[18] Basu D, Lin Q, Chen W, et al. Regularized cost-model oblivious database tuning with reinforcement learning[J]. Transactions on Large-Scale Data and Knowledge-Centered Systems XXVIII: Special Issue on Database-and Expert-Systems Applications, 2016: 96-132.
[19] Ge Y F, Wang H, Bertino E, et al. Evolutionary dynamic database partitioning optimization for privacy and utility[J]. IEEE Transactions on Dependable and Secure Computing, 2023.
[20] Lan Hai, Bao Zhifeng, Peng Yuwei. A Survey on Advancing the DBMS Query Optimizer: Cardinality estimation, cost model, and plan enumeration[J]. Data Science and Engineering, 2021, 6(1): 86-101.
[21] Gani A, Siddiqa A, Shamshirband S, et al. A Survey on Indexing Techniques for Big Data: Taxonomy and Performance Evaluation[J]. Knowledge and information systems, 2016, 46(2): 241-284.
[22] Kossmann J, Halfpap S, Jankrift M, et al. Magic Mirror in My Hand, Which is The Best in the Land? An Experimental Evaluation of Index Selection Algorithms[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 2382-2395.
[23] Ding Bailu, Das S, Marcus R, et al. Ai Meets Ai: Leveraging Query Executions to Improve Index Recommendations. Proceedings of the 2019 International Conference on Management of Data. 2019: 1241-1258.
[24] Sadri Z, Gruenwald L, Lead E. DRLindex: Deep Reinforcement Learning Index Advisor for A Cluster Database. Proceedings of the 24th Symposium on International Database Engineering and Applications. 2020: 1-8.
[25] Sharma V, Dyreson C, Flann N. MANTIS: Multiple Type and Attribute Index Selection using Deep Reinforcement Learning. 25th International Database Engineering and Applications Symposium. 2021: 56-64.
[26] Thanopoulou A, Carreira P, Galhardas H. Benchmarking with TPC-H on off-the-shelf hardware[J]. ICEIS (1), 2012: 205-208.
[27] Graefe G. B-tree Indexes for High Update Rates[J]. ACM SIGMOD Record, 2005, 35(1): 39-44.
[28] POWA (2019) PostgreSQL workload analyzer. https://powa.readthedocs.io/
[29] Pedrozo W G, Nievola J C, Ribeiro D C. An adaptive approach for index tuning with learning classifier systems on hybrid storage environments[C]//Hybrid Artificial Intelligent Systems: 13th International Conference, HAIS 2018, Oviedo, Spain, June 20-22, 2018, Proceedings 13. Springer International Publishing, 2018: 716-729.