Analysis and Practice of Automatic Indexing in Big Data

Bo Zhao[1,*], Hailin Liao[2,] Zebin Wen[3], Wanting Wen[4]

{746392365@qq.com[1*],1599508639@qq.com[2],2079331865@qq.com[3],1145972524@qq.com[4]}

School of Computer Science, Guangdong University of Science and Technology, Dongguan, Guangdong 523083, China

Abstract: In the digital age, the explosive growth of data and the need for instant performance pose challenges to database management systems. In a big data environment, query performance has become a key issue, and index optimization is a key means to improve performance. Automatic Indexing It dynamically selects, creates and adjusts indexes through machine learning models, evaluates the need for new indexes and the need for existing indexes, creates a new index when needed, and deletes it when it is no longer needed. This article introduces the working principle and research methods of automatic indexing, conducts simulation experiments, and analyzes the experimental results to complete the application exploration and practice of automatic indexing technology in a big data environment. At the end of the text, the challenges and limitations of automatic indexing are presented.

Keywords: Automatic Indexing; Big Data; Database

1 Preface

The index problem is a long-standing problem in the database field. Researchers have discovered that indexes may improve the efficiency of data queries, and also recognized the difficulty and challenges of index selection [1-2]. Traditional manual index management has always been one of the main methods for database performance optimization, including B-Tree indexes, bitmap indexes, partitioned indexes, function-based indexes, etc. Mainstream commercial database vendors have developed relevant tuning tools to assist DBAs in tuning, such as Microsoft's SQL Server database tuning advisor (DTA) [3] and IBM's DB2 design advisor (DB2Advis) [4-5] and Oracle's SQL access advisor (SQL access advisor) [6], etc., the dynamically changing query patterns in the big data environment make it difficult for traditional manual methods to track and adapt, and the speed of manual intervention often cannot keep up with the data. and changes in query patterns. With the development of machine learning technology and the expansion of database application scenarios, related research on index selection issues has regained the attention of more and more researchers. For example, the index selection scheme that introduces machine learning technology [7], some researchers have proposed Adaptive indexing technology [8-10], which automatically generates an index structure based on query statements and makes adaptive adjustments to avoid the problem of index selection, such as database cracking[8] and index adaptive merging[9] technology, automatic indexing technology provides an unprecedented automation method, providing an intelligent and efficient index management solution in big data.

PMBDA 2023, December 15-17, Nanjing, People's Republic of China

DOI 10.4108/eai.15-12-2023.2345391

2 Introduction

The development of automatic indexing technology provides new ways to solve the complexity of index management in big data environments. Auto-indexing automatically creates indexes, making them unusable, and making them invisible and visible if they need to be rebuilt. Based on the query, it can also create SQL baselines. Very importantly, it can learn from previous implementations and their impact on performance and automatically change the policy using reinforcement learning algorithms. The core idea of this technology is to use machine learning algorithms to learn query patterns from database workloads, predict potential performance improvements, and automatically create or adjust indexes. The introduction of automatic indexing makes index management no longer dependent on manual labor, but is continuously optimized through real-time analysis and adaptive learning. In a big data environment, research on how automatic indexing works involves multiple key aspects to ensure its adaptability to large-scale, high-complexity data sets. Detailed working steps see figure 1 below:

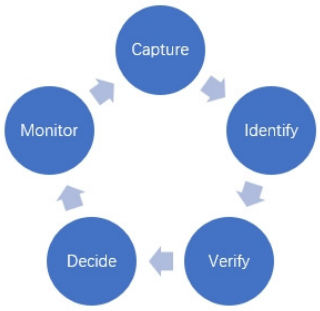

Figure 1 Schematic diagram of automatic indexing artificial intelligence cycle

The first is the capture phase. During this phase, it captures all SQL statements from the application workload and creates an automatic SQL tuning set.It then goes through the identification phase, where all query performance is analyzed and candidate indexes are suggested. These indexes are created and made invisible.During the validation phase, query performance with and without candidate indexes is measured. If an index is required, materialize the index and measure performance again.Next comes the decision stage. At this stage, if there are performance improvements, the index will become visible. If there is no improvement, the index will remain invisible. If certain queries have a performance impact on the index, create a SQL plan baseline using those affected queries. This way the performance of these queries does not change.The final phase is monitoring where query performance is continuously monitored and the cycle is repeated. Invisible indexes that have exceeded the retention period will be deleted.

3 Research methods for automatic indexing in big data

3.1 Analysis of data set characteristics

First, automatic indexing performs in-depth analysis of the characteristics of large-scale data sets, including data distribution, data relationships, and data types. This analysis helps determine which columns might be good candidates for an index, as well as the type and ordering of the index. With complete visibility into your data set, Automated Indexing can more accurately select and create adaptable indexes to improve query performance.

For example, for a customer information table, automatic indexing can perform data analysis, query analysis, partition analysis, etc. through in-depth analysis of the national customer information data set. Automatic indexing will first conduct a comprehensive data analysis of the data set, including analysis of the data distribution, data relationships, and data types of each column. For example, you can view the unique value distribution, frequency distribution, etc. of columns such as region, customer number, and insurance type.

3.1.1. Analyze commonly used query patterns

Which columns are frequently used in conditional filtering, sorting, or join operations. This helps determine the best indexing strategy to support frequently executed queries and improve query performance. Suppose there is a sales order database. Automatic index analysis finds that most queries involve the order_date, customer_id, and product_id columns. Through the machine learning model, it was found that for frequent range queries, such as retrieving orders by date range, creating a range index based on the order date order_date may improve performance. At the same time, for the case of equivalence query of frequent customer number customer_id, it may be recommended to create an equivalence index based on customer number customer_id. In this way, the automatic indexing system generates index strategies suitable for different query needs based on query patterns to optimize database performance.

3.1.2. Select ideal candidates for the index

Based on the results of the analysis, automatic indexing will select which columns are ideal candidates for the index. This may include fields such as region, customer number, etc. to support common query operations.

3.1.3 Determine the type of each index

Based on data distribution and query requirements, automatic indexing will determine the type of each index (such as B-tree index, hash index, etc.) and the sorting method (ascending or descending order). For example, a B-tree index might be chosen for range queries, while a hash index might be chosen for equality queries.

3.2 Automatic index selection

In practical applications, machine learning models for automatic index optimization can adopt a variety of methods. For example, decision tree model, random forest model, deep learning model. Machine learning algorithms can predict which indexes are likely to have a positive

impact on specific query performance by analyzing query execution plans, query optimization history, and database statistics. Query plan performance allows the system to select execution plans to minimize query response time. Automatically identify potential query patterns and data access patterns by analyzing database workloads. Based on these patterns, the system can automatically create appropriate indexes and can provide real-time index optimization suggestions, allowing the database system to flexibly adapt to changing data and query requirements.

3.3 Adapt to large scale and high complexity

Taking into account the characteristics of the data set, machine learning model learning, real-time performance monitoring and automatic index adjustment, the principle of automatic indexing is designed to adapt to large-scale, high-complexity data environments. This design enables the system to flexibly and intelligently select and optimize indexes to meet the needs of complex query patterns when facing large and dynamically changing data sets.

4 Research design and experiments

In order to gain a deeper understanding of the performance and effect of Automatic Indexing in a big data environment, we conducted a series of experiments, focusing on the experimental parameters, steps and performance evaluation. The research design of this article and the specific details of the experiment are described in detail below.

4.1 Software and hardware configuration

Oracle version 21c was selected for the database, the operating system was a Linux virtual machine, and a separate table space was configured.

4.2 Data selection and feature engineering

The experiment selected representative large-scale data sets of customers, products, orders, and order details, including multi-table associations, complex queries, and large-scale data, and loaded approximately 900,000 random records. Such a data set can better simulate the database workload in a real big data environment.

4.3 Experimental steps

1) Start SQL optimization tracking and collect detailed information about the query execution plan.

2) Use concurrent queries to simulate workloads, multi-table queries, and associate tables with 900,000 pieces of data.

3) Use WHERE conditions to simulate filtering queries on different columns.

4) Turn off SQL optimization tracking, analyze the execution plan, and query the cost estimate and access path of each step.

5) Machine learning model training and prediction, use historical query execution plans and performance data to train the model, and learn which index configurations are most effective

for different types of queries. For example, use the decision tree algorithm for training, use the scikit-learn library, partition the data set, model training, model evaluation, and model prediction to determine the index configuration most likely to improve performance.

6) Automatic index selection, configure automatic indexing at the solution level in the Oracle database, turn on the configuration parameter AUTO_INDEX_MODE, and automatically select or recommend indexes for queries based on the prediction results of the machine learning model.

7) Generate a report, view the automatic indexing report, and select several important parts from the report to display. For example, Table 1 shows the index list created using automatic indexing, and Table 2 shows the comparison of historical query performance indicators and automatic indexing indicators. Table 3 shows the comparison between historical execution plan indicators and automatic index indicators.

Table 1: Index creation status

Owner	Table	Index	Key	Type	Properties
HR	Test1	*SYS_AI_03vgaf9tybk	Id2	BTree	-
HR	Test1	SYS_AI_dmu539wl39n7s	section	BTree	-

Table 2 Query statistics

	Original Plan	Auto Index Plan
Elapsed Times(s）	393880521	5234120
CPU Time （s）	161139246	1715183
Buffer Gets：	1433221	31712
Optimizer Cost：	86828	32751
Disk Reads：	1433960	0
Direct Writers：	0	0
Rows Processed：	3	1
Executions：	7	1

Table 3 Execution plan

	Original Plan	Auto Index Plan
Plan Hush Value	520385367	3101189324
Operation	Full Table Scan	Index Range Scan
Rows	45237068	14419670
Cost（%CPU）	86828	32751
Time	00:00:04	00:00:02

5 Effect analysis

5.1 Performance analysis

From the experimental results shown in the above three tables, performance evaluation is carried out in terms of response time, resource utilization, and indexing effect. Evaluate the

actual improvement in query performance made by automatic indexing by measuring the execution time of each query. Monitoring the usage of system resources, including CPU utilization, memory usage, etc., it can be seen that after using the machine learning algorithm, the number of data rows accessed is reduced, and the execution time is shortened by half. After using the machine learning model for training and prediction, the index strategy is intelligently selected based on the actual query mode and data distribution, which can well adapt to complex query modes and optimize real-time performance monitoring indicators, thereby improving the query performance of the database.

5.2 Analysis of the working principle of automatically creating and adjusting indexes

Based on analysis of data set characteristics and dynamic query patterns, Auto Indexing automatically creates new indexes or adjusts the properties of existing indexes. This process is automated and does not require manual intervention by the database administrator. By automatically creating and adjusting indexes, the system can more flexibly adapt to changes in query patterns in big data environments, ensuring real-time optimization of indexes.

From the analysis of automatically maintained data dictionary query results, automatic indexing first automatically creates invisible indexes, and then tests the impact of the indexes on SQL statements. If the impact is positive and the SQL statements perform better using indexes, then the indexes become visible. If the performance of the SQL statements does not improve due to automatic indexing, then these indexes will be marked as unavailable and the corresponding SQL statements will Be blacklisted. Unused manual indexes are never deleted by the automatic indexing process, but can be deleted automatically if needed. Automatic indexing can be disabled at any time, or set to report read-only mode.

Some SQL statement performance will show improvements while others will show degradation due to automatically created indexes. In this case, the index becomes visible, but automatic indexing creates a SQL plan baseline to prevent performance-degrading SQL statements from using the index. Other SQL statements that do not suffer performance degradation will continue to use the index. The importance of automatic indexing in real-time adaptability. It can dynamically adjust indexes based on changes in actual query patterns to better adapt to dynamic workloads in big data environments.

6 Conclusions

This article first introduces how automatic indexing technology works, including the capture, identification, verification, decision-making and monitoring stages. Then, research methods for applying automatic indexing in big data environments are discussed, including data set feature analysis, dynamic query pattern learning, real-time performance monitoring and feedback, and considerations of adapting to large scale and high complexity. This article also shows the results of evaluating the performance of automatic indexing through experiments, including index creation, query statistics and execution plans. It proves the effectiveness of automatic indexing in improving query performance. The number of accessed data rows is reduced, and execution The time is cut in half. It has better adaptability to complex query models and implements real-time performance monitoring and optimization. .

Although automatic indexing has shown significant advantages in big data environments, it also has some limitations that may affect its performance. In a big data environment, the heterogeneity and complexity of data may cause automatic indexes to perform inconsistently when facing different types and distributions of data. Certain data distribution patterns may be difficult to accurately capture by machine learning models, so when data heterogeneity is high, the performance improvement of automatic indexing may be subject to certain constraints. By continuously expanding and improving the functions of automatic indexing, we can look forward to more intelligent and adaptive index management tools in the future, bringing greater innovation and performance improvements to the database field. Research in these directions will help promote the widespread application of automatic indexing technology in big data environments.

Acknowledgments: This article is one of the phased achievements of the Guangdong University of Science and Technology's school-level scientific research team, "Business Intelligence and Decision Support research team".（GKY-2022CQTD-3）

References

[1] Comer D. The difficulty of optimum index selection [J]. ACM Transactions on Database Systems, 1978, 3(4): 440–445

[2] Chaudhuri S, Datar M, Narasayya V. Index selection for databases: A hardness study and a principled heuristic solution [J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(11): 1313–1323

[3] Agrawal S, Chaudhuri S, Kollar L, et al. Database tuning advisor for Microsoft SQL Server 2005 [C] //Proc of the 30th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 2004: 1110–1121

[4] Valentin G, Zuliani M, Zilio D C, et al. DB2 advisor: An optimizer smart enough to recommend its own indexes [C] //Proc of the 16th Int Conf on Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2000: 101–110

[5] Zilio D C, Rao Jun, Lightstone S, et al. DB2 design advisor: Integrated automatic physical database design [C] //Proc of the 30th Int Conf on Very Large Data bases. San Francisco, CA: Morgan Kaufmann, 2004: 1087–1097

[6] Dageville B, Das D, Dias K, et al. Automatic SQL tuning in Oracle 10g [C] //Proc of the 30th Int Conf on Very Large Data bases. San Francisco, CA: Morgan Kaufmann, 2004: 1098–1109

[7] Li Guoliang, Zhou Xuanhe, Sun Ji, et.al. A survey of machine learning based database techniques [J]. Chinese Journal of Computers, 2020,43(11):2019-2049

[8] Idreos S, Kersten M L, Manegold S. Database cracking [C/OL] //Proc of the 3rd Biennial Conf on Innovative Data Systems Research. 2007[2020-09-01]. https://www.cidrdb.org/cidr2007/

[9] Graefe G, Kuno H. Adaptive indexing for relational keys [C] //Proc of the 26th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2010: 69–74

[10] Graefe G, Idreos S, Kuno H, et al. Benchmarking adaptive indexing [G] // LNPSE 6417: Proc of the 2nd Technology Conf on Performance Evaluation and Benchmarking. Berlin: Springer, 2011: 169–184