Research on Automated Database Health Check and Slow SQL Optimization

Xiaomin Tan
Intermediate Engineer, China Telecom Corporation Limited Research Institution
China
tanxm@chinatelecom.cn

Yuzhong Zhang
Senior Engineer, China Telecom Corporation Limited Research Institution
China
zhangyz22@chinatelecom.cn

Abstract

Databases are the foundation of application services. Low-quality SQL queries leads to performance bottlenecks of the database, which in turn may affect other services. The research focuses on developing a comprehensive framework that enables self-awareness, self-optimization, self-evaluation for databases basing on anomaly detection and experience. Algorithms are used to automatically detect anomalies of the database resources. Slow SQL statements are extracted from logs for trend analysis and large models are used in analyzing changes in SQL execution plans to find out the SQLs to be optimized. Finally, an Experiential rule library is decided for SQL Optimization. The research is able to find out poor-performing SQL queries and provide optimization recommendations, which helps to improve working efficiency and ensure the stability of database services.

CCS Concepts

• **Information system** → Data management systems; Database management system engines.

Keywords

SQL diagnosis, SQL optimization, Trend detection, Large Language Model

ACM Reference Format:
Xiaomin Tan and Yuzhong Zhang. 2024. Research on Automated Database Health Check and Slow SQL Optimization. In *2024 International Conference on Cloud Computing and Big Data (ICCBD 2024), July 26–28, 2024, Dali, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3695080.3695132

1 Introduction

With the explosive growth of internet businesses, databases, as the core components for data storage and processing, directly impact the response speed of applications and user experiences. Database operations are gradually transitioning from passive handling to proactive prevention, and from manual to automated processes. Intelligent maintenance technologies are continuously being explored and developed, aiming to achieve a cycle of problem detection, automatic repair, and optimization evaluation. Machine Learning is the most common way to achieve self-monitoring [1,2,3]. A "self-driving" database management system is designed for autonomous operation using forecast models by monitoring the executed queries, which will automatically improve the latency of the queries [2]. Both supervised and unsupervised machine learning methods are used to automated tune database configurations based on the past experience [3].

SQL queries, as the primary interface for database operations, have become a crucial aspect of database management and application development. However, traditional SQL optimization methods often rely on human experience and trial and error, making it difficult to adapt to dynamically changing business requirements and data characteristics. Therefore, exploring a method that can automatically diagnose root causes and intelligently optimize SQL queries and evaluate the effectiveness of SQL optimization has become an important issue in the database field [4]. A framework that can diagnose the root causes of slow SQLs with a loose requirement for human intervention [5].

Index and view optimization are popular methods for SQL self-optimization [6,7]. Index optimization usually achieves the purpose of SQL optimization by adjusting or adding indexes, which can often produce very significant effects in a short period of time. Further, faster queries are needed for better performance [8,9]. However, past experiences are indeed an important aspect to pay more attention to rewrite the poor SQL queries, which is the most challenging work.

2 Challenges

Firstly, most of database performance bottlenecks stem from inefficient SQL statements. In a high-concurrency environment, poorly written SQL can severely impact or even cripple the performance of the entire database system However, developers vary in their ability to write SQL, leading to significant disparities in performance.

Secondly, even though we are aware that poorly performing SQL queries can significantly impact database performance, finding out these poor queries efficiently and accurately remains a challenge. For example, dealing with a small table containing just a few dozen rows, queries tend to execute swiftly regardless of the type of query. However, as the volume of concurrent requests escalates and the scale of data increases significantly, SQL queries can become noticeably sluggish.

Thirdly, complex business logic frequently demands solutions beyond straightforward index tuning to address the performance

ICCBD 2024, July 26–28, 2024, Dali, China

ACM ISBN 979-8-4007-1022-3/24/07
https://doi.org/10.1145/3695080.3695132

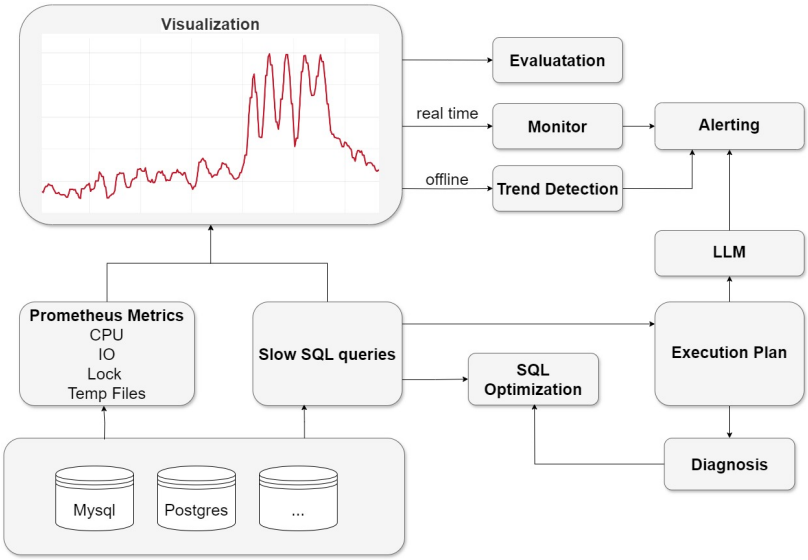

Figure 1: Structure of automated database health check system

of SQL queries, presenting a formidable challenge for automated diagnostics.

Currently, monitoring technology has reached a mature stage, and when the overall performance of the database encounters a bottleneck, the system will automatically trigger an alert. However, at this point, it is usually necessary for a database administrator (DBA) to conduct an in-depth analysis to determine the specific reasons for the performance degradation. Although automated diagnosis of performance issues is already quite challenging, achieving self-healing capabilities for databases is even more difficult. Therefore, employing a comprehensive set of strategies to identify and optimize these performance bottlenecks is essential for maintaining the stability and responsiveness of the database.

3 Methodology

This paper proposed an automated database health check system, the core of which lies in its real-time monitoring capabilities. This capability can instantly capture and deeply analyze key performance indicators of the database. Such real-time monitoring not only focuses on the current performance status but also leverages trend analysis of historical data to further uncover anomalies, providing data support for long-term performance optimization and effectiveness evaluation.

Another key feature of the system is the continuous optimization of SQL queries with an understanding of business requirements. Additionally, the trend analysis of slow SQL queries combined with in-depth analysis of execution plans, integrated with a large language model (LLM) to analyze execution plans, enables the system to predict the SQL operations that may lead to performance degradation. This advanced analytical capability helps us understand the root causes of performance bottlenecks and can even provide optimization suggestions.

Overall, a real-time monitoring, historical trend analysis, SQL query optimization, and intelligent alert mechanisms into a comprehensive solution for database performance monitoring and optimization is shown in Figure 1. It not only ensures the efficiency of database operations but also provides database administrators with powerful decision-making tools.

3.1 Slow Query Extraction

Slow SQL queries are extracted from logs, recording the execution user and duration time for each SQL. The data will be deeply analyzed to regularly calculate the quantiles of SQL execution duration on a weekly and daily basis, as well as the cumulative total duration.

Let R denote the total number of executions on the previous day, Sa represent the median number of executions from the previous day, Rratio indicate the ratio of the median execution time on the previous day to the median execution time of the previous week. Poor queries and the slowest SQL statements are preliminary screened according to the performance standards set,

$$R > S_a \tag{1}$$

and

$$S_{ratio} > 1 \tag{2}$$

3.2 Performance Monitor

Database performance are monitored in real time. Metrics include but not limited to CPU utilization, I/Os, the status of lock files and the total counts of connection. Alerts will be triggered once current value exceeds the threshold.

On one hand, if the CPU usage rate shows regular spikes or sustained high loads, this may indicate that the database is processing a large number of requests or executing complex queries. Such conditions may result in increased response times for the database, which in turn could impact the stability of both the database and the business operations. On the other hand, increasing execution time

of queries also leads to performance degradation or even failure. Therefore, anomaly detection algorithms and trend prediction techniques are used to detect upward trends and fluctuations, which provides early warnings before a disaster occurs.

There are several steps in prediction as shown in ALGORITHM 1. Rolling windows or time-series cross-validation methods can be chosen to split samples in step 1. Different types of metrics like Mean Squared Error (MSE) and R-squared (R^2) can be chosen to evaluate the model's accuracy in step 3. Similarly, regression models, such as Ridge or Lasso Regression, can be consider to improve model performance.

Algorithm 1 Trend Prediction Algorithm

```
# Assume 'df' is a DataFrame with time-series data and 'duration' is the feature of interest.
# 1. Time-Series Specific Splitting
tscv = TimeSeriesSplit(n_splits=5)
X_train, X_test, y_train, y_test = next(tscv.split(df))
# 2. Data Preprocessing
scaler = StandardScaler()
feature_scaled = scaler.fit_transform(df [ ['duration']])
# 3. Model Evaluation
model = LinearRegression()
model.fit(feature_scaled, df ['duration'])
# Predict and evaluate the model
y_pred = model.predict(feature_scaled)
mse = mean_squared_error(df ['duration'], y_pred)
r2 = r2_score(df ['duration'], y_pred)
# 4. Model Interpretability
slope = model.coef_ [0] intercept = model.intercept_
# 5. Multivariate Linear Regression (Include more features in the model)
# Determine the trend based on the slope
if slope > 0:
    print (f"The time series has an upward trend with a slope of {slope:.2f}")
```

3.3 Execution Plan Analysis

As SQL execution durations increase day by day, monitoring for changes in execution plans becomes critical. Such changes are able to identify SQL statements with declining performance. Manually identifying differences in execution plans quickly is a daunting task. Thus, use of a Large Language Model(LLM) to analyze execution plans over time is crucial for drawing conclusions on SQL performance. The detection of phrases such as "increased cost," "higher memory consumption," or "longer execution times" in the conclusions will activate a warning system, ensuring that database administrators and business teams are immediately notified.

Prompt is as follows:

Algorithm 2 Database Execution Plan Comparison

```
# You are a database administrator and performance optimization specialist with extensive experience in enhancing database efficiency and ensuring optimal system performance.
Input Requirements:
1. Specify the type of database you are using (e.g., MySQL, PostgreSQL, Oracle, SQL Server, etc.).
2. Please provide two segments of SQL execution plans.
Steps to Use:
1. Paste your first execution plan into the text box below.
2. Paste your second execution plan into another text box.
3. Select or type in your database type.
Example:
Database Type: [Your database type here]
Execution Plan 1:
[Your first execution plan text here]
Execution Plan 2:
[Your second execution plan text here]
Output:
The tool will analyze both execution plans and highlight the differences between them.
It will provide insights into the performance impact, such as expected changes in query speed, resource usage, etc
It will give a conclusion whether the plan 2 becomes worse comparing with plan 1.
It will Identify the performance issues with executing the plan 2.
```

3.4 Rule Library Establish

Due to the complex business operations, the occurrence of numerous subqueries is a common phenomenon. To address the persistent issue of SQL performance degradation arising from repetitive business patterns, a rule library grounded in business logic has been meticulously assembled, drawing from early optimization endeavors. This library acts as a strategic tool for SQL optimization, ensuring that potential issues are preemptively identified and resolved before queries are deployed. For instance, when querying for child projects nested under a parent project, it is essential to embed the parent project's ID within the subquery to enhance performance and accuracy.

3.5 Evaluation

The SQL database performance assessment method integrates performance trend analysis, slow log analysis, table space monitoring and forecasting, application of visualization tools and reporting. This comprehensive approach provides database administrators with a set of tools to monitor key performance indicators in real time, promptly identify and resolve potential issues, optimize database performance, and ensure the stability and efficiency of the system.

4 Results

4.1 Slow Query Analysis

An overview of the number of slow SQL queries executed and the total time consumed by each user is shown in Figure 2. Consequently, these accounts are brought to the forefront as the primary candidates for optimization initiatives.

4.2 Trend Prediction Accuracy

Time series segmentation, random splitting, and K-fold cross-validation are all used for sample division. It has been proven

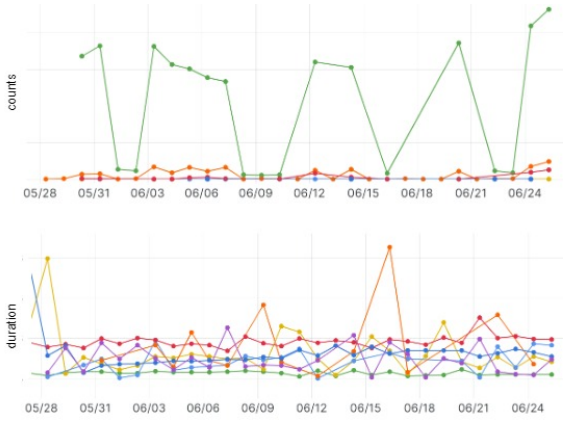

Figure 2: Overview of counts and duration of slow SQL queries group by users

Table 1: Accuracy of trend prediction using 3 sample split method.

Split Method	Accuracy
Time series segmentation	100%
Random splitting	40%
K-fold cross-validation	40%

that for time series, time series segmentation has higher accuracy, as shown in Table 1, which can accurately identify the execution model with an upward trend or slow SQL.

4.3 SQL Diagnosis

Using the template from ALGORITHM 2, following LLMs are compared to evaluated the execution plan. Outputs of the comparison are shown in Table 2. Plan 1: execution time: 1328.553 ms, Buckets: 1048576 (originally 1048576), Batches: 64 (originally 16), Memory Usage: 32192KB.

Plan 2: execution time: 13849.413 ms. Buckets: 524288 (originally 524288), Batches: 32768 (originally 32), Memory Usage: 24928kB.

LLMs provide accurate and insightful analysis when examining SQL execution plans, which is essential for pinpointing and addressing the causes of SQL performance slowdowns. For instance, the yi-large-preview model can identify when certain indexes contribute minimally to query performance, thereby assisting database administrators (DBAs) in allocating resources more effectively. Meanwhile, the glm-4-0520 model can diagnose potential reasons for a decrease in query performance through the increasing number of batches, which may indicate a significant growth in table data. CodeFree is the AI large model of R&D Cloud of China Telecom, and it indeed offers correct conclusions and identify the issue of hash.

4.4 Optimization Effect

Several rules are shown in Table 3. SQL is a partial query that has undergone desensitization processing. Based on the library, SQL query will be check whether it matches the rules. According to the rules, execution time increase.

Monitor metrics are also applied to evaluate the effect of SQL optimization, such as temp files in Figure 3.

5 Conclusion

This paper presents an innovative approach to automated database monitoring and SQL optimization, capable of promptly identifying SQL queries that are gradually slowing down. It not only offers comprehensive diagnostics and optimization suggestions to assist DBAs in conducting in-depth analyses, but early alerts before database failures occur. Furthermore, a set of optimization rules based on specific business is distilled. It continuously tracks the effectiveness after SQL optimization, ensuring that the optimization measures lead to tangible performance improvements. By employing this method, we can manage database performance more intelligently, ensuring the continuity and stability of business operations.

References

[1] Marcus, R., & Papaemmanouil, O. (2019). Plan-Structured Deep Neural Network Models for Query Performance Prediction.
[2] Pavlo, A., Angulo, G., Arulraj, J., Lin, H., Lin, J., Ma, L., ... & Zhang, T. (2017). Self-Driving Database Management Systems. In CIDR (Vol. 4, p. 1, 2017, January).
[3] Van Aken, D., Pavlo, A., Gordon, G. J., & Zhang, B. (2017). Automatic database management system tuning through large-scale machine learning. In Proceedings of the 2017 ACM international conference on management of data (pp. 1009-1024, 2017, May).
[4] Remil, Y., Bendimerad, A., Mathonat, R., Chaleat, P., & Kaytoue, M. (2021). " What makes my queries slow?": Subgroup Discovery for SQL Workload Analysis ysis. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 642-652, 2021, November). IEEE.
[5] Ma, M., Yin, Z., Zhang, S., Wang, S., & Pei, D. (2020). Diagnosing root causes of intermittent slow queries in cloud databases. Proceedings of the VLDB Endowment, 13(8), 1176-1189.
[6] Lan, H., Bao, Z., & Peng, Y. (2020). An index advisor using deep reinforcement learning. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management (pp. 2105-2108, 2020, October).
[7] Zhou, X., Liu, L., Li, W., Jin, L., Li, S., Wang, T., & Feng, J. (2022). Autoindex: An incremental index management system for dynamic workloads. In 2022 IEEE 38th International Conference on Data Engineering (ICDE) (pp. 2196-2208, 2022, May). IEEE.
[8] Habimana, J. (2015). Query optimization techniques-tips for writing efficient and faster SQL queries. International Journal of Scientific & Technology Research, 4(10), 22-26.
[9] Myalapalli, V. K., & Savarapu, P. R. (2014). High performance SQL. In 2014 Annual IEEE India Conference (INDICON) (pp. 1-6, 2014, December). IEEE.

Table 2: Outputs of comparison of SQL execution plan.

Model	Outputs
qwen2-72b-instruct	Plan 2 indeed performs worse compared to Plan 1 Reduced efficiency in join operations leading to higher processing times
yi-large-preview	Plan 2 becomes worse compared to Plan 1 in terms of execution speed and resource usage Plan 2 seems to have more parallel operations, but these operations took much longer to complete. This could be due to increased parallelism not being effectively utilized, possibly due to resource contention or less efficient parallel execution plans The time taken for index scans in Plan 2 is significantly higher, suggesting that the index usage might be less efficient or that the indexes themselves might need to be optimized
glm-4-0520	Plan 2 is worse than Plan 1 in terms of execution time The number of batches in the Parallel Hash increased dramatically, which could imply more memory and CPU time being used in the operation, possibly due to a higher number of rows being processed or a less efficient hash table setup The increased number of batches in the Parallel Hash could be a sign of excessive memory usage or spillage to disk, which can severely degrade performance
CodeFree	Execution Plan 2 exhibits worse performance than Execution Plan 1 The increased time in the Gather and Parallel Hash Left Join stages suggests potential bottlenecks or inefficiencies in data distribution and join operations

Table 3: SQL rules based on business logic.

Rule	SQL before optimization	SQL after optimization	Execution time before	Execution time after
Change the OR clause for retrieving sub-projects under a parent project to union	SELECT t1.id AS m_003, t2.id AS m_002 FROM t_001 t1 JOIN t_002 t2 ON t2.parent_id = t1.id OR t2.id = t1.id	select dsp2.display_name, select t1.id AS m_001, t2.id AS m_002 from t1 join t2 on t2.parent_id = t1.id UNION select t1.id AS m_001, t2.id AS m_002 from t1 join t2 on t2.id = t1.id	2-10s	< 1s
Set dataset parameters by passing in the project ID in sub query	Select id from (select * from t1) as m_001 where t1.project_id = 1	Select id from (select * from t1 .project_id = 1) as m_001	2-10s	< 1s

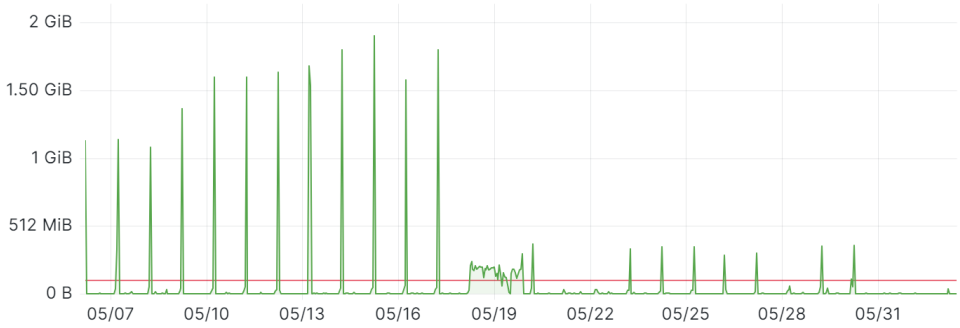

Figure 3: Temp file size of database