Development of Intelligent Tools for Detecting Resource-intensive Database Queries

Salah M.M. Alghazali[1], Konstantin Polshchykov[2], Ahmad M. Hailan[3], Lyudmila Svoykina[4]
Department of Applied Informatics and Information Technology, Belgorod State University, Belgorod, Russia[1]
Institute of Engineering and Digital Technologies, Belgorod State University, Belgorod, Russia[2]
College of Computer Science and Mathematics, Thi-Qar University, Thi-Qar, Iraq[3]
Institute of Intercultural Communication and International Relations, Belgorod State University, Belgorod, Russia[4]

***Abstract*—The detection of resource-intensive queries which consume an excessive amount of time, processor, disk, and memory resources is one of the most popular vulnerabilities of Database Management Systems (DBMS). The tools for monitoring and optimizing queries typically used in modern DBMS were analyzed, and their shortcomings were identified. Subsequently, the relevance of new intelligent tools' development for timely and reliable detection of resource-intensive queries to databases was distinctly justified. The study concluded a set of analysis of an extended statistical parameter which indicated to be of interest for identifying resource-intensive queries. The initial set of queries' parameters reduced by two consecutive methods. Firstly, normalizing the set of indicators using a sigmoid function. Secondly, selecting a finite number of principal components based on the Cattell test. Whereas the clustering of a set of queries performed using self-organizing Kohonen maps. Suggestions for further studies in the classification algorithm context were indicated in lights of the study's conclusions.**

***Keywords*—Resource-intensive queries; database; detecting; self-organizing Kohonen maps; statistical parameters**

I. Introduction

A resource-intensive request to the database will be considered as such when a request uses an excessive amount of time, processor, disk, and memory resources to process that request. Search and conversion of resource-intensive queries is carried out by Data Base Administrators (DBAs) based on information from clients. The gradual decline in the performance of distributed clients' systems requires proactive optimization measures.

Queries to databases are programmed based on special tools which are part of the Structured Query Language (SQL) [1]. The existing methods currently used in system utilities for searching and identifying problematic queries do not always identify resource-intensive SQL statements. Correspondingly, there are instances where those queries would wrongfully not be classified as resource-intensive. This is due to the fact that utilities only exploit a limited number of performance parameters or incorrect analysis's algorithms based on simple ranking. Furthermore, the functional redundancy and high cost of such utilities are yet other drawbacks for the effects of those inefficient queries.

It is perceived that manual searches could provide correct detection of problematic SQL queries. Notwithstanding, the complexity and time-consuming processes of analyzing large amounts of information does not allow a person to process data at a reasonable speed. Manual searches involve continued time-wasting and require experience in Database Management Systems (DBMS). Especially the knowledge on the architecture, features of storing system information, the language of structured queries, data structures and models of application programs.

The identification of resource-intensive SQL queries entails the purchase of expensive tools. Likewise, it compels the involvement of highly qualified specialists with relevant knowledge and experience in writing unique scripts for the necessary data acquisition.

This article presents the results of research obtained by the authors on the analysis and development of intelligent tools designed for timely and reliable automatic detection of resource-intensive database queries.

II. Literature Review

There are many approaches is use today to either prevent or minimize the impact of inter-query interactions on a shared cluster. Despite these measures, performance issues due to concurrent executions of mixed workloads still prevail causing undue waiting times for queries [2]. The foundation of modern database query optimization is the collection of statistics describing the data to be processed, but when a database or Big Data computation is partially obscured by user-defined functions, good statistics are often unavailable [3]. H.Bodepudi [4] mentioned how the Relational DBMS reporting scripts performance can be improved by incorporating the Spark framework without changing the existing queries in the RDBMS. In paper [5] presents Intermittent Slow Query Anomaly Diagnoser, a framework that can diagnose the root causes of Intermittent Slow Queries with a loose requirement for human intervention. Intermittent Slow Queries are slow queries which might be more hazardous to database users than other slow queries.

Z.Miao [6] proposes a system designed to help users understand SQL query evaluation and debug SQL queries. The system lets users interactively "trace" the evaluation of complex SQL queries, including those with correlated subqueries.

B.G.Lekshmi [7] focuses on the hardware-conscious the query optimization in a relational DBMS using extended rules and cost models as well as on refining the optimization

strategies for changes in the hardware state of execution.

Cost-based optimizer studied in paper [8] is adopted in almost all current database systems. A cost-based optimizer introduces a plan enumeration algorithm, and then uses a cost model to obtain the cost of that plan, and selects the plan with the lowest cost. In the cost model, cardinality, the number of tuples through an operator, plays a crucial role. The research [9] aims to provide approximate query processing as a middleware solution using query optimization for heterogeneous databases.

A large scale of resource requirements can be reduced by minimizing query execution time that maximizes resource utilization [10]. Recently, parallel DBMSs have significantly improved query processing performance [11].

Thus with the growing volume of data being analyzed, more and more advanced monitoring, preventive analysis, and proactive optimization tools are required to identify problematic SQL queries. The above reasons determine the relevance of developing new intelligent tools for timely and reliable resource-intensive database queries' detection.

III. Materials and Methods

During processing of the request, a plan for executing a SQL statement should be formed to get the information requested from the database more quickly and cost-effectively. Modern tools for monitoring and optimizing queries in Oracle DBMS, MS SQL Server, and DB2 sort and evaluate SQL statements based on the values of one, less often two, or three indicators that characterize the speed of query processing. Further, the use of these indicators could be used to assess the resources used for the processing of queries. These methods were found to be causing an erroneous diagnostics of the resource-intensive queries and beg the need for an optimized method.

Advanced data collection technologies for resource-intensive queries searching are implemented in the automatic workload storage of the Oracle DBMS [12]. These include the Advanced Workload Repository (AWR). This infrastructure provides services to Oracle DBMS components for collecting, maintaining, and using statistics for problem detection and self-tuning [13]. AWR makes it possible to collect various information about requests and the operation of the system. The statistics on executed SQL statements contained in the DBA_HIST_SQLSTAT table are best suited for searching and subsequent optimization of resource-intensive queries.

The main source of information useful for identifying problematic SQL statements is statistical data on the query execution period. Most often, a resource-intensive SQL query detection is performed using the following parameters:

1) for a large number of disk reads;

2) by the number of logical reads or buffer gets operations;

3) by the number of parse calls;

4) by the number of executions.

However, studies had shown that the extended set of parameters contained in Table I is of interest for identifying resource-intensive queries. The initial statistical analysis of the initial data on various test sets showed that almost all parameters have sharply asymmetric right-hand distributions with long thin tails. The asymmetry coefficients take values from 10 to 15, and the excesses take values from 150 to 400.

TABLE I. Extended Set of SQL Query Parameters

Parameter	***Name***
Executions	number of request executions
sharable_mem	the size of the sharable memory occupied by the child cursor (bytes)
loaded_versions	the number of child cursors for which memory is allocated
version_count	total number of child cursors
physical_read_bytes	the number of bytes physically read from the disk
physical_write_bytes	number of bytes written to disk
physical_read_requests	number of disk reads
physical_write_bytes	number of bytes written to disk
physical_read_requests	number of disk reads
physical_write_requests	number of operations to write information to disk
rows_processed	number of rows processed and returned
parse_calls	number of calls to the parsing procedure
px_servers_execs	the number of runs of auxiliary server processes for parallel execution of the request
end_of_fetch_count	the number of times the query returned a full set of data on successful completion
Fetches	the number of calls to the procedure for extracting rows with data from the cursor
buffer_gets	number of gets from the buffer cache (logical read)
Invalidations	the number of times the child cursor became invalid
Loads	number of memory downloads / reloads
Sorts	number of sorting operations
direct_writes	number of direct write operations to disk
cpu_time	processor time for parsing, executing, and fetching data for a given cursor (μs)
elapsed_time	total request duration time (μs)
javexec_time_delta	time elapsed in the execution of Java programs (μs)
plsexec_time_delta	PL/SQL code execution time (μs)
iowait	time spent waiting for user input / output (μs)
apwait	time spent waiting for the app (μs)
ccwait	time spent waiting for shared data and resources (μs)
clwait	time spent waiting for concurrency events to complete (μs)
io_interconnect_bytes	the number of bytes sent between the DBMS and the Exadata storage system
io_offload_elig_bytes	the number of bytes filtered for processing at the Exadata data store level
io_offload_return_bytes	number of bytes returned from the Exadata data store layer
cell_uncompressed_bytes	number of bytes received during data decompression on Exadata data storage nodes
optimized_physical_reads	number of optimized reads from the Exadata data store

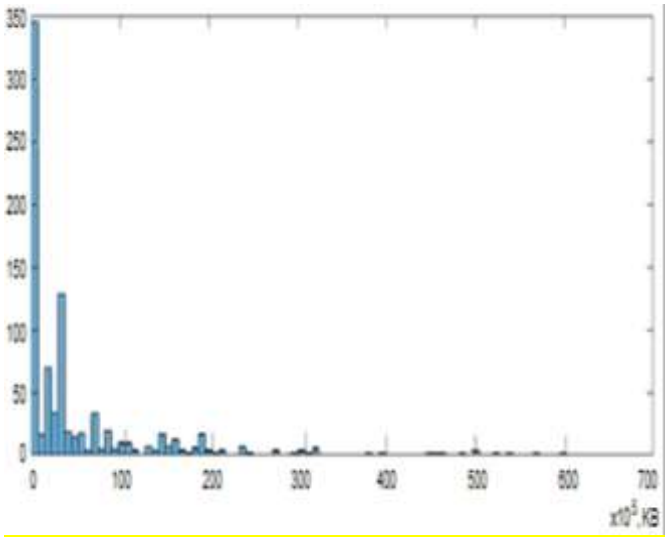

Fig. 1. Distributions of the Sharable_Mem Parameter.

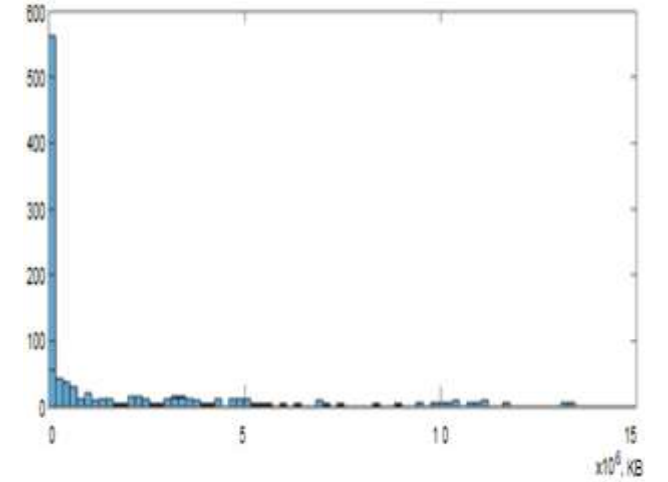

Fig. 2. Distributions of the Physical_Read_Bytes Parameter.

Fig. 1 depicts typical distributions of the sharable_mem parameter. Fig. 2 illustrate a typical distribution of the values of the physical_read_bytes parameter.

Visual histogram analysis outlines most of the SQL query parameter values which were found to be concentrated in the range of 1-2% of the entire diapason of possible values. The parameter values that are in the tails of the distributions indicate the presence of resource-intensive queries in the analyzed sets.

The extended set of parameters presented in Table I is redundant. To account for all these indicators in order to detect resource-intensive queries, algorithms with high computational complexity will be required, which is undesirable in the process of performing practical tasks. It is necessary to significantly reduce the number of analyzed parameters and leave the most informative ones.

The ranges of changes in statistical parameters differ by several orders of magnitude, so to reduce the number of these parameters, a preliminary normalization of the indicators set was carried out using a sigmoid function:

$$\tilde{x} = \text{sigmoid}\left(\frac{x-\bar{x}}{\sigma}\right) \quad (1)$$

where $\tilde{x}$ is the normalized value of the parameter; x is the initial value of the parameter; $\bar{x}$ is the average value of the parameter.

The value of the sigmoid function of the value a is calculated using the expression:

$$\text{sigmoid}\ (a) = \frac{1}{1+e^{-ca}}. \quad (2)$$

where c is the extension coefficient of the sigmoid. For most parameters, the best value is c=8.

Further, in order to reduce the number of parameters that characterize SQL queries, the Principal Component Analysis (PCA) method was used to get the better analyze and make best decision based upon detect the relevant parameters [14].

The PCA implementation allowed us to construct a new space of parameters to reduce dimensions. In this case, the variance of the initial parameter space was first calculated. Then, eigenvectors were obtained that determine the directions of the principal components and the value of the variance associated with them. In order to reduce dimensionality, the variance value associated with each principal component was divided by the sum of the variances for all components. As a result, the proportion of variance associated with each component is obtained. At the final step, so many components were discarded so that the proportion of the variance of the remaining components reached 80%.

The results of applying the PCA method are presented in Table II. The second column shows the variances of the selected components. The third column shows the percentage of the total variance for each component. The first component explains 21% of the total variance, the second component explains 11.5% of the total variance, and so on. The fourth column contains the values of the accumulated variance. Each value shown in the fourth column is the sum of those values from the third column whose component numbers do not exceed the number of the calculated value of the accumulated variance. For example, the value of the accumulated variance in the fourth row of Table II was obtained by summing the values of the third column contained in rows 1, 2, 3, and 4.

The choice of a finite number of principal components was carried out on the basis of the Cattell criterion, which allows us to preserve from 65 to 80% of the information concentrated in the original set of significant features. The selection of the first main components based on the Cattell test is shown in Fig. 3. The number of principal components according to the Cattell criterion was determined by the inflection point on the graph of eigenvalues. In this case, the inflection point is the boundary between the interval of the sharp decline of the analyzed graph and the subsequent interval of the flat curve.

TABLE II. RESULTS OF THE PCA METHOD APPLICATION

Component number	*Component value*	*Explained variance proportion*	*Accumulated variance*
1	6.819	21.31	21.31
2	3.670	11.47	32.78
3	2.673	8.35	41.13
4	2.531	7.91	49.04
5	3.670	11.47	32.78
6	1.838	5.74	54.78
7	1.660	5.19	59.97
8	1.299	4.06	64.03
9	1.112	3.48	67.51
10	1.066	3.33	70.84
11	1.031	3.22	74.06
12	1,005	3.14	77.20
13	0.999	3.12	80.32
14	0.997	3.12	83.44
15	0.949	2.97	86.40
16	0.913	2.85	89.26
17	0.790	2.47	91.73
18	0.716	2.24	93.97
19	0.693	2.17	96.13
20	0.350	1.09	97.22
21	0.262	0.82	98.04
22	0.247	0.77	98.81
23	0.199	0.62	99.44
24	0.101	0.32	99.75
25	0.065	0.20	99.96
26	0.013	0.04	100.00
27	0.001	0.00	100,.0

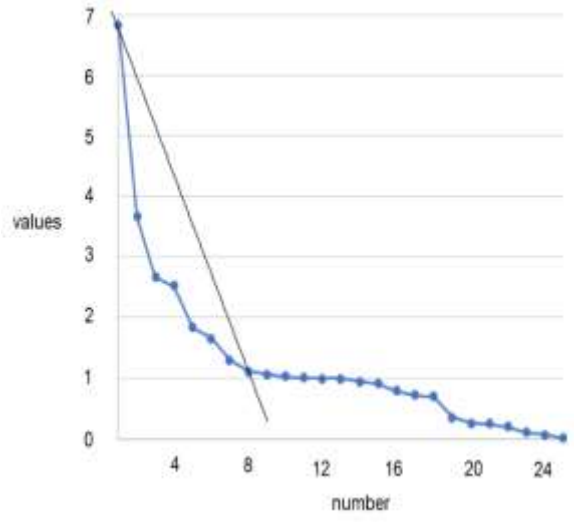

Fig. 3. The Selection of the First Main Components based on the Cattell Test.

The first eight components can be recognized as the main components. For the remaining components, there is a significant slowdown in the values decrease. The selected main eight components contain a fairly high percentage of the explained variance, approaching 70%. The selection of the main components using the PCA method and the Cattell test was performed for different sets of parameter values that characterize SQL queries. As a result, it was found that to detect problematic database queries, it is enough to use from four to nine significant parameters instead of the original thirty-two.

The set of all possible SQL queries must be divided into an unknown number of clusters. For this purpose, the device of Kohonen's Self-Organizing Maps (hereinafter SOM) is used [15, 16, 17]. The SOM training algorithm is developed, based on the use of a rational value of the winning neuron topological neighborhood width, which makes it possible to configure the neural network to prevent its overfitting. The implementation of neural network clustering using more than 200,000 instances of SQL statements processed in the past made it possible to form 4 main subsets of queries, of which two clusters can be classified as resource-intensive queries.

The further process of automating the search for problematic queries can be associated with the construction of a Bayesian classifier [18,19,20], which can be trained on a limited, independent, pre-researched and marked-up subset of SQL queries. The classifier can be trained to assign the studied elements to one of two classes (resource-intensive and non-resource-intensive queries). Further training and improvement of the classifier for subsequent use for operational and proactive optimization can be carried out on newly identified resource-intensive queries. Information about such requests can be received from DBA specialists in an interactive mode.

IV. CONCLUSION

Thus, in the process of detecting problematic SQL statements, it is proposed to use up to nine significant statistical parameters that characterize the speed of query processing, as well as the resources used for this purpose. The original set of thirty-two analyzed parameters contained in the DBA_HIST_SQLSTAT table of the Oracle DBMS workload storage was reduced by first normalizing the set of indicators using a sigmoid function and then selecting a finite number of principal components based on the Cattell criterion. Clustering a set of SQL statements based on SOM made it possible to form four main subsets of queries. At the same time, resource-intensive requests were concentrated in two clusters.

As a result of this study, the authors were able to substantiate the correctness of using a reduced set of statistical parameters to detect resource-intensive queries. This made it possible to successfully perform neural network clustering of the analyzed queries.

The subject of further research would be the development of a classification algorithm designed to determine the ratio of the analyzed SQL statement to one of the previously allocated clusters, which will automatically detect resource-intensive queries.

REFERENCES

[1] C. Wang, A.Cheungc and R. Bodik,"Synthesizing highly expressive SQL queries from input-output examples," Proceedings of the 38th ACM SIGPLAN Conf. on Programming Language Design and Implementation, pp. 452-466, June 2017.

[2] P. Kalmegh, S. Babu and S. Roy, "Analyzing Query Performance and Attributing Blame for Contentions in a Cluster Computing Framework," arXiv:1708.08435v2, 2018.

[3] S. Sikdar and C. Jermaine, "MONSOON: Multi-Step Optimization and Execution of Queries with Partially Obscured Predicates", Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, pp. 225–240, 2020.

[4] H. Bodepudi, "Faster The Slow Running RDBMS Queries With Spark Framework," International Journal of Scientific and Research Publications, volume 10(11), pp. 287-291, 2020.

[5] M. Ma, Z. Yin and S. Zhang et al. "Diagnosing Root Causes of Intermittent Slow Queries in Cloud Databases," PVLDB, vol. 13(8), pp. 1176-1189, 2020.

[6] Z. Miao, T. Chen, A. Bendeck, K. Day, S. Roy and J. Yang, "I-Rex: an interactive relational query explainer for SQL," Proc. VLDB Endow, vol 13, pp. 2997–3000, 2020.

[7] B.G. Lekshmi and K.Meyer-Wegener, "COPRAO: A Capability Aware Query Optimizer for Reconfigurable Near Data Processors," 2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW), pp. 54-59, 2021.

[8] H. Lan., Z. Bao and Y.A. Peng, "Survey on Advancing the DBMS Query Optimizer: Cardinality Estimation, Cost Model, and Plan Enumeration," Data Sci. Eng., vol 6, pp. 86–101, 2021.

[9] M. Muniswamaiah, T. Agerwala and C.C. Tappert, "Approximate Query Processing for Big Data in Heterogeneous Databases," 2020 IEEE International Conference on Big Data, pp. 5765-5767, 2020.

[10] A. Bachhav, V. Kharat and M. Shelar, "An Efficient Query Optimizer with Materialized Intermediate Views in Distributed and Cloud Environment," Tehnički glasnik, vol 15, pp. 105-111, 2021.

[11] X. Zhou and C. Ordonez, "Matrix Multiplication with SQL Queries for Graph Analytics," 2020 IEEE International Conference on Big Data (Big Data), pp. 5872-5873, 2020.

[12] I. Fernandez, "Beginning Oracle Database 12c Administration. From Novice to Professional," Apress, p. 384, 2015.

[13] S.M.A.Fattah, M.A. Mahmoud and L.A.E. Abd-Elmegid, "An Adaptive Hybrid Controller for DBMS Performance Tuning," (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 5(5), 2014.

[14] S. Belattar, O. Abdoun and H. El khatir, "New Learning Approach for Unsupervised Neural Networks Model with Application to Agriculture Field," (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 11(5), 2020.

[15] S. Sinha, T.N. Singh, V.K. Singh and A.K. Verma, "Epoch determination for neural network by self-organized map (SOM)," Computational Geosciences, vol. 14, pp. 199-206, 2010.

[16] M. Sakkari and M. Zaied, "A convolutional deep self-organizing map feature extraction for machine learning," Multimedia Tools and Applications, vol. 79, pp. 19451-19470, 2020.

[17] L.R.Clovis, C.A.Scapim, R.J.B. Pinto, M.Vivas, J.E.A. Filho and A.T.A. Júnior, "Yield stability analysis of maize hybrids using the self-organizing map of Kohonen," Euphytica, vol. 216, p. 161, 2020.

[18] J. Yu, M. Bai, G. Wang and X. Shi, "Fault diagnosis of Planetary Gearbox with incomplete information using assignment reduction and flexible naive bayesian classifier," Mechanical Science and Technology, vol. 32, pp. 37-47, 2018.

[19] S.R.B. Shree and H.S. Sheshadri, "Diagnosis of alzheimer's disease using naive bayesian classifier," Neural Computing and Applications, vol. 29, pp. 123-132 , 2018.

[20] W. Zhang, Z. Zhang, HC. Chao and FH. Tseng, "Kernel mixture model for probability density estimation in Bayesian classifiers," Data Mining and Knowledge Discovery, 2018, vol. 32, pp. 675-707.