ISBN 978-981-14-1684-2
Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering
(WCSE 2019 SUMMER)
Hong Kong, 15-17 June, 2019, pp. 697-703
doi: 10.18178/wcse.2019.06.103

Improving SQL Query Response Time thru Client Side Processing in Client-Server Environment

[1] Ruben A. Parazo[+], Dr. James A. Esquivel[2]

[1] Tarlac Agricultural University, Camiling, Tarlac, Philippines
[2] Angeles University Foundation, Angeles City, Pampanga, Philippines

Abstract. Queries executed by a client in the database server were profiled, managed and reused to respond to future subdued queries to be executed in the client. This technique not only reduced the number of queries to be processed by the database and the server but also contributes in decreasing the utilization of network, database and server resources because future subdued queries to be executed that are subdued from the previous executed query were responded locally. Conditions in the query were removed before sending to the database for evaluation in order to increase its effectiveness for its intended purpose. Profiled queries that are subdued by incoming query were purged including its corresponding result set while the incoming query was profiled in the repository. This was implemented as a technique to avoid storing redundant data in the repository as well as to avoid query capability duplication. Tests showed that utilization of a local repository of previous executed queries to respond to a subdued requested query decreased the latency incurred in fetching data both in small, medium and large number of records as compared to a requested query responded by a database and server where the data still travelled thru the network.

Keywords: SQL Query, Database, Reusing Result set, Condition Elimination, Query logs, serial scanning.

1. Introduction

Query processing in a client-server database system raises the question of where to execute queries to minimize the communication costs and response time of a query, and to load-balance the system [1]. The latency incurred in the processing of a query depend on the server-side and on client-side query processing capabilities [2]. SQL query statement sent by the client to be responded by the server performs some steps before a user may be able to view the requested information. 1. The user formulates query using the application software. 2. The application software connects to the database and submits the query. 3. The database retrieves data and returns these to the user. 4. The application software receives the incoming data, and presents them to the user. These four steps will be repeated from time to time for every query that will be requested. This method constantly utilizes the resources of the database, server and the network. To this end in a recent paper of [3], to reduce the number of queries to be sent by the client to the server, previous queries requested by the client should profiled and reused to respond to a future subdued queries to be requested. Management of the query includes removing the conditions (if there is any) attached to the query before sending to the database for evaluation so as not to affect the number of records to be returned and to enhanced its usefulness to respond to a future subdued queries to be requested. Profiling a query includes the storing of the text of the query and its result set [4]. Every future queries executed by the client subdued to previously executed query will be responded using the repository to speed up the execution time [5] because the request was served locally as compared to sourcing out the data to the original source and travels thru the network [6]. This processed shifted the workload of the database in the server-side to the client-side. Queries that are unable be responded by the latter were fetched its data to the database server which will then be subjected to profiling in the repository.

[+] Corresponding author. Tel.: 00639774623431; fax: + (None).
E-mail address: raparazo@yahoo.com.

2. Objectives

The objective of the study is to design an algorithm that will reuse previously executed query to answer future subdued query to be requested. Specifically it aims to;

1. Enhance the ability of the query that are subjected to be profiled.
2. Utilization of the profiled query to answer future subdued query to be requested.
3. Evaluate the performance of the algorithm in responding to a requested subdued query.

3. Related Literature

3.1. Session-Based browsing for more effective query reuse

Session Based Browsing allows users to access past queries stored in query logs which are the starting point of the users in formulating or writing a query. SQB provides keyword search over a query log. Instead of simply listing all matching queries, it presents the results as a set of query sessions. It allows users to view the result of the query in order to visualize and understand how the query works which eventually serve as the users' guide in formulating a new query [7].

3.2. Super rack: reusing the results of queries in MapReduce systems

One of the options to speed up the execution of workflows in MapReduce system is to save the results of the query and reuse them in future if needed. Each workflow can store query result in Super Rack, so incoming query can simply reused them rather than obtaining the data in the original source and employ parsing the query to produced results [8].

3.3. Pass-Thru architecture via Hash techniques to remove duplicate query results

First hash index was computed for the first query result from the data source and it passed on to the user. For the second query results, a second hash index will be computed and compared to the first hash index. It will be checked if there is a collision between the first and second index, if the first and second indexes match, then the first data Source is queried for data corresponding to the Second query result. And if the first data Source contains the data, then the Second query result is considered a duplicate and is discarded [9].

3.4. Method for presenting database query result sets using polymorpic output format

Database queries are submitted with an indication of a selected output format. To process the query, data records are retrieved and formatted according to the selected output format, as well as formatted for additional output formats supported by a given a query application. Once returned, query results may be presented in the selected format. A user may switch the presentation of the query result from the selected format to others, without having to re-execute the database query [10].

3.5. Efficient Server Side Data Retrieval for Execution of Client Side Applications

A system, method and article of manufacture are provided for efficiently retrieving data. A total amount of data required for an application executed by a client is determined. In a single call, the total amount of data from a server is requested over a network. All of the data is bundled into a data structure by the server in response to the single call. The bundled data structure is sent to the client over the network and the data of the data structure is cached on the client. The cache data of the data structure is used as needed during execution of the application on the client [11].

3.6. Prefetching Query Results and its Impact on Search Engines

The study proposes offline and online strategies to select and prioritize queries that will potentially benefit from prefetching. The offline strategy relies on the observation that the queries tend to repeat on a daily basis and applies query log mining to identify queries whose results are to be prefetched. The online strategy relies on a machine learning model that predicts the next occurrence time of issued queries. Prefetching operations are then prioritized based on these expected times [12].

4. Methodology

4.1 Query Condition Elimination

Before a query sent to the database for evaluation, it must have to undergo evaluation to determine the presence of a condition. This condition limits the number of result set to be produced because every record to be returned must passed to the criteria set. Serious drawback for this is, if the result set of the query will be utilized to respond to future subdued queries to be requested because it is capable only respond to subdued queries joined with similar condition. The presence of a where keyword in the query signifies that it contains condition. Performing serial scanning on the text of the query to be profiled was implemented [13]. Serial scanning works by directly searching the presence of a "where" keyword in the query to be profiled followed by elimination including its attributes before sending to the database for evaluation.

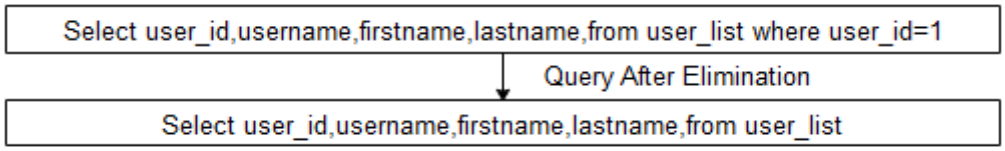

Fig. 1: Shows the state of the query after undergone condition elimination process

4.2. Query Profiling

Query Profiling was applied after the query undergone elimination process. A folder that serves as the repository of unique requested query was created to store the text of the query and its result set. A file was created and it will contain the text of requested queries which referred to as the query logs [14]. The result set of the query was exported as text file [15] [16] and it was stored in a row and column patterned on the format replied by the database over the requested query. The uniqueness of the query is determined by comparing the text of incoming query to the text of queries stored in the query logs in terms of their source and the included field. Before the text of the query was registered in the query logs, it was attached with QUERY INDENTIFICATION NUMBER, which is an auto number generated by the algorithm [17]. The generation of QIN number will start to one (1) and progresses sequentially. The same QIN number was used as a filename of the exported result set of the query. The QIN number is used to: 1. Identify the text of the query in the query logs; 2. Identify the result set of the query; 3. Established relationship between the text of the query and its result set and 4. Key to pinpoint who among the profiled queries are capable to respond to the requested query. The query logs was used purposely to respond to queries that are subdued from the past queries. Requested Queries that are unable to be responded by profiled queries was directed to source-out its data to the database and it was deposited in the repository which will become one of the queries to be used as referenced to respond to future queries that will be requested.

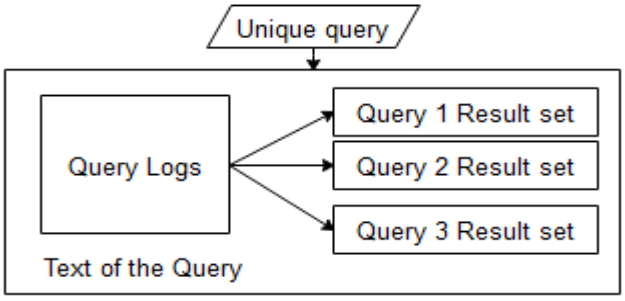

Fig. 2: The text of the select statement of the unique query was stored in the query logs while its result set was exported in the same repository which the QIN establishes relationship between the Query logs and the result set of the query.

4.3. Query Purging

Queries that are profiled were subjected to purging if it is subdued by an incoming query. Profiled queries were considered as subdued by an incoming query if they have the same table source and the fields of profiled query are existed in the incoming query. In this scenario the profiled query considered as duplication of the incoming query, so the profiled query/ies were purged while the incoming query was profiled [18]. Purging is a process of deleting the text of profiled query in the query logs including its exported corresponding result set. This technique was implemented in order to avoid query capability

duplication because the incoming query contains or has the capability to respond to future queries that can also be served by queries that are already profiled. This way it avoids the redundant storing of query text and its corresponding result set in the repository. So, retained query in the query logs are considered unique and with different capabilities.

4.4. Identifying the qualified profiled query

The text of the requested query was compared to the text of profiled queries in the query logs in order to determine who among the profiled queries are responsive to the requirement of the requested query. The responsiveness of profiled query in the query logs is determined by having a similar source to that of the requested query and the field/s in the text of the requested query are existed in the text of the profiled query [19].

4.5. Establishing a local and temporary database.

MySQL database were installed in the client side for the purpose of creating table with memory type which were populated with the result set of the profiled query and eventually became the local source of the subdued requested query.

4.6. Creating table

After identifying the qualified profiled query to respond to the subdued requested query, a create table statement with memory type was created using the text of the profiled query as bases**.** Then a table name in the create table statement was statically assigned, in the study we named the table as "***qtbl***". The number of fields to be created in the create table statement were depend on the number of fields existed in the identified profiled query and these were renamed as "f1, f2, f3…" which the "f1" signifies as the first field, "f2" as the second field, "f3" as the third field and progresses sequentially are as long as there is/are fields in the text of the profiled query. The fields in the text of the profiled query are found between the "SELECT and FROM statement". Memory type of table was utilized for this purpose to ensure a high degree of performance. After executing the create table statement it results to the creation of a memory type of table in the database then followed by loading the corresponding result set of the profiled query using the attached QIN to the profiled query that identifies the result set of the profiled query [20] [21].

Assume that the profiled query below was identified and used to respond to subdued requested query

select user_id,username,first_name,middle_name,password status from users

The create table statement would be;

create table qtbl (f1 varchar(30), f2 varchar(30), f3 varchar(30), f4 varchar(30), f5 varchar(30), f6 varchar(30));

4.7. Executing the subdued incoming query

Before a requested query was executed, its field/s and table source were restructured using the text of the identified profiled query as bases. Restructuring is the process of replacing the field/s and table source of the incoming query in order to become identical to the created table. The replacement of its fields was by means of comparing and finding the position or index of the fields of the incoming query in the fields of the profiled query. The newly field/s of the incoming query becomes "f1,f2,f3…" while the table source was the name of the created table in the create table statement.

Assume that the profiled query below was identified and used to respond to subdued requested query.

select user_id,username,first_name,middle_name,password status from users

Assume that the query below was requested and subdued to profiled query

select user_id, first_name,middle_name from users

The restructured query which would be;

Select f1,f3,f4 from qtbl

The field "user_id" in the requested query becomes "***f1***" because it is the first field in the profiled query, the field "first_name" in the requested query becomes "***f3***" because is the third field in the profile query and the field "middle_name" in the requested query as "***f4***" because it is the fourth field in the profiled query.

The same method of replacement were applied for the succeeding fields (if there is any) in the requested query. The keyword after the "from" which is "users" were also replaced by "qtbl" because it is the name of the table used in the create table statement. After restructuring or reconstructing the subdued requested query, execution follows and by this time, the subdued query were directed to source out its data locally. The created table were deleted after the result set of the query has been displayed. This was done in order to prevent and free the memory of the client from storing large amount of information.

5. Results and Discussion

5.1. Testing query

Seven (7) different queries were formulated and executed in a client-server environment. Three types of sample data consisting of 10,000, 100,000 and 500,000 rows categorized as small, medium and large number record respectively were used in the testing [22][23]. The sample data was downloaded at https://www.sample-videos.com/download-sample-sql.php. The server was installed with MySql and uploaded the downloaded sample data utilizing the interface of phymyadmin. In the client side, each of the queries below were executed eleven times in each category of data using the designed algorithm and the standard way of fetching data in the database server. In the eleven times of execution, the response time of the first execution was excluded because it took longer time to render output due to caching works. The computed averaged response time of the next ten executions (2-11) were taken and served as the response time of each of the query [24] both in the designed algorithm and the standard way of fetching data in the database.

1. Select user_id from user_details
2. Select user_id,username from user_details
3. Select user_id,username,firt_name from user_details
4. Select user_id,username,first_name,last_name from user_details
5. Select user_id,username,first_name,last_name,gender from user_details
6. Select user_id,username,first_name,last_name,gender,password from user_details
7. Select user_id,username,first_name,last_name,gender,password,status from user_details

5.2. Order of query execution

Two scenarios were used in the execution of the formulated queries, these are the ascending and descending order of execution. In ascending order of execution, each of the queries were profiled and accepted in the repository because the algorithm treated each of them as different from each other but when the next query comes-in, the previous query was purge while the incoming query was profiled. This is because the profiled query is subdued by incoming query which means the queries that can be responded by the profiled query can also be responded by the incoming query. In this case, only the seventh query which is the "Select user_id, username, first_name, last_name, gender,password,status from user details" retained in the repository. In descending order of execution, the seventh query was executed first, then the next six queries were not accepted in the repository because they are all subdued to the previous query.

Table 1: Average response time (in milliseconds) of execution in ascending order

Queries	Standard fetching			Fetching using the Model		
	Small	Medium	Large	Small	Medium	Large
1	0.103	5.403	12.521	0.085	0.783	8.0576
2	0.091	5.771	17.701	0.090	1.257	12.0593
3	0.081	6.023	13.615	0.078	1.467	13.8807
4	0.095	6.307	24.287	0.069	1.734	16.5249
5	0.078	3.778	22.711	0.070	1.738	13.7572
6	0.076	7.670	43.877	0.069	4.491	31.0488
7	0.081	4.468	40.402	0.054	3.941	30.1295

Table 1 shows the average response time of ten executions using the standard fetching and fetching using the algorithm in ascending order of execution of the queries. In general it is noted that the fetching using the

algorithm performs faster compared to standard fetching in all of the categories of data. The average response time incurred by fetching using the algorithm in executing the queries for small number of records is lower by .09 milliseconds as compared to standard fetching which translates to an equivalent of four ten percent (14%) improvement. Furthermore, in medium and large number of records, fetching using the model recorded an average difference of twenty four (24) milliseconds and fifty (50) milliseconds respectively as compared to the standard fetching. This translates an equivalent improvement of sixty percent (60%) for medium number of records and twenty eight percent (28%) for large number of records.

Table 2: Average response time of execution in descending order

Queries	Standard fetching			Fetching using the Model		
	Small	Medium	Large	Small	Medium	Large
1	0.0767	8.5508	34.2958	0.0630	4.0252	30.4180
2	0.0747	7.2726	33.0053	0.0572	4.0436	31.0439
3	0.0760	1.9033	14.4684	0.0678	1.7616	12.2233
4	0.0830	1.7654	15.2745	0.0678	1.7665	15.2570
5	0.0850	1.6573	12.3372	0.0710	1.5170	11.2573
6	0.0939	1.2940	10.3029	0.0943	1.0974	9.9910
7	0.1072	0.9288	7.60139	0.0875	0.4311	7.22380

Table 2 shows the average response time of ten executions using the standard fetching and fetching using the algorithm in descending order of execution of the queries. In general it is noted that the fetching using the algorithm performs faster compared to standard fetching in all of the categories of data. The average response time incurred by fetching using algorithm in executing the queries for small number of records is lower by .09 milliseconds as compared to standard fetching which translates to an equivalent of four ten percent (14%) improvement. In medium and large number of records, fetching using the model recorded an average difference of nine point eleven (9.11) milliseconds and nine point eighty seven (9.87) milliseconds respectively as compared to the standard fetching. This translates an equivalent improvement of thirty nine percent (39%) for medium number of records and eight percent (8%) for large number of records. Furthermore, it is noted that the standard fetching demonstrated a better performance in terms of its response time for the medium and large number of records as compared in the ascending order of execution, however, still did not exceeded the performance of the algorithm.

5.3. Failure transparency test

After executing all of the seven queries, the database in the server side were turned off to signify that the database and the server encountered problems which causes the database and database server of being unable to respond to a request. In the client side, all of the seven queries specified in the testing query were executed. As expected, the standard fetching encountered error due to unavailability of the source while the algorithm were still able to respond to the entire request. This is because all of the queries executed are subdued to a previously executed query that is already profiled in the client side, so there is no need to transfer the request to the database via the network, instead it was served locally. This is one of the reasons why the performance of the model outperformed the standard fetching data in all of the categories of data.

6. Conclusions

Based on the result of performance testing, it is concluded that the utilization of previously executed query to answer a subdued requested query decreased the response time and cost of processing on the database, database server and network because the data was serve locally as compared to the data that still requesting from server and travels thru the network The dependency of the client to the database server in terms of responding to queries was decreased as the number of query in the query logs increases. The shifting of some of the workloads of the database server to the client prevents the constant utilization of infrastructure such as the database server and network resources by properly utilizing the previously requested information..

7. References

[1] Kossmann, D., & Franklin, M. J. (1998). *A Study of Query Execution Strategies for Client Server Database Systems.* Maryland, USA: UMIACS.

[2] Hartig, O., Lette, I., & Perez, J. (2017). *A Model of Distributed Query Computation in Client-Server Scenarios.* Chile: Millennium Institute for Foundational Research on Data, Chile.

[3] Parazo, R. A., & Esquivel, J. A. (2018, October). Scalable Query Profiling Employing Purging and Elimination Technique. In *Proceedings of the 4th International Conference on Industrial and Business Engineering* (pp. 169-173). ACM

[4] Khoussainova, N., Kwon, Y., Liao, W.-T., Balazinska, M., Gatterbauer, W., & Suciu, D. (2011). *Session-based Browsing for More Effective Query Reuse.* Seattle, WA, USA: University of Washington.

[5] Wang, Z., Xu, T., & Wang, D. (2015). Super Rack: Reusing the Results of Queries in MapReduce Systems. IEEE

[6] Hygerth, H., & Hakansson, M. (2016). *Effcient Web Scraping and Database Synchronization.* Stockholm: Royal Institute of Technology KTH, Stockholm, Sweden.

[7] Khoussainova, N., Kwon, Y., Liao, W.-T., Balazinska, M., Gatterbauer, W., & Suciu, D. (2011). *Session-based Browsing for More Effective Query Reuse.* Seattle, WA, USA: University of Washington.

[8] Wang, Z., Xu, T., & Wang, D. (2015). Super Rack: Reusing the Results of Queries in MapReduce Systems. IEEE

[9] Guha, R. V. (2000). *Patent No. 6081805.* USA.

[10] Dettinger, R. D., & Daniel P. Kolz. (2013). *Patent No. US 8,370,375 B2.* USA

[11] Browman-Amuah, M. K. (2003). *Patent No. US 6615253 B1*. USA.

[12] Jonnassen,S.,Cambazoglu,B.B.,& Silvestri, F. (August 16, 2002). Prefectchig Query Result and its impact on Search Engines.

[13] Umajancy.S, & Thanamani, D. A. (2013). AN ANALYSIS ON TEXT MINING -TEXT. *International Journal of Advanced Research in Computer and Communication Engineering*, 3125-3129.

[14] Khoussainova, N., Kwon, Y., Liao, W.-T., Balazinska, M., Gatterbauer, W., & Suciu, D. (2011). *Session-based Browsing for More Effective Query Reuse.* Seattle, WA, USA: University of Washington.

[15] Dettinger, R. D., & Daniel P. Kolz. (2013). *Patent No. US 8,370,375 B2.* USA.

[16] Beck, T., Hastings, R. K., Gollapudi, S., Free, R. C., & Brookes, A. J. (2014). GWAS Central: a comprehensive resource for the comparison and interrogation of genome-wide association studies. *European Journal of Human Genetics*, 949-952.

[17] Thakare, A., Dhande, P. M., & Bamnote, D. G. (2013). Query Optimization in OODBMS using Query Decomposition & Query Caching. *International Journal on Recent and Innovation Trends in Computing and Communication*, 469-474.

[18] Guha, R. V. (2000). *Patent No. 6081805.* USA.

[19] Parazo, R. A., & Esquivel, J. A. (2018, October). Scalable Query Profiling Employing Purging and Elimination Technique. In *Proceedings of the 4th International Conference on Industrial and Business Engineering* (pp. 169-173). ACM

[20] Dahlberg, M. (2002). *USA Patent No. US 6463439 B1.*

[21] Bramer, M. (2019, February 19). *PHP_in_Action_Converting_Data_between_Text_Files_and_Database_Tables.* Retrieved from PHP_in_Action_Converting_Data_between_Text_Files_and_Database_Tables.html: https://www.researchgate.net/publication/302495295_PHP_in_Action_Converting_Data_between_Text_Files_and_Database_Tables

[22] Grandinetti, L. (2008). *High Performance Computing and Grids in Action* Amsterdam; Berlin; Oxford; Tokyo; Washington, DC: IOS Press.

[23] Asiminidis, C., Kokkonis, G., & Kontogiannis, S. (2018). DATABASE SYSTEMS PERFORMANCE EVALUATION FOR IoT APPLICATIONS Vol.10, No.6,. *International Journal of Database Management Systems (IJDMS)*.

[24] Heinrich, M. E., & Kambayatughar, J. (2017). *Advanced Database Project Report.* (ULB)Universit'e Libre De Bruxelles.