Scalable Query Profiling Employing Purging and Elimination Technique

Ruben A. Parazo
Tarlac Agricultural University
Paniqui Tarlac Philippines
+639774623431
raparazo@yahoo.com

Dr. James A. Esquivel
Angeles University Foundation
Capas Tarlac Philippines
+639054407933
esquivel.james@auf.edu.ph

ABSTRACT

Reusing Queries contributes in speeding up the performance of database in responding to future queries as it can reduce the number of database queries to be processed and sent back to the user. Profiling a query in a machine who requested a query in database server improved the response time when the query is reused. It also avoids the utilization of database and network resources. This is because the data will be served locally as compared to obtaining the data from the original source that is still travelling in the network which entails cost not only on the database server but also in the network infrastructure. A condition in the query that limits the result set of the query will be removed before it will be sent to the database for evaluation. This is to enhance the ability and usefulness of the result set of the query in answering future subdued queries to be requested. To prevent query capability duplication as well as to efficiently manage the space utilizes in the profiling of queries, profiled queries that are subdued by an incoming query will be purged while queries that are subdued to profiled query will not be accepted in the query logs and its result set will not be exported.

CCS Concepts

Profiling Query → Query logs • Purging query → avoids query capability duplication • Elimination → enhanced **the ability of the result set of query in answering future subdued query to be executed.** The dependency of the client to the database server in terms of responding to queries will be 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. It can also decrease the response time for requested queries that are subdued to previously executed query.

Keywords

Queries, Database, Subdued Queries, Reusing Result Set, Purging, Query Identification Number, Index;

1. INTRODUCTION

Reusing Queries contributes in speeding up the performance of database in responding to future queries as it can reduce the number of database queries to be processed and sent back to the user. Moreover it can also decrease the utilization of database resources as well as the infrastructure cost according to West [1]. Information in databases is typically accessed using SQL query. The select statement is the responsible statements in order obtain results from a database. A typical request of information in the database performs some steps before a user may 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 made. This entails cost because the resources of the database will be frequently utilized. If it will be applies in a server-client setup, the bulk of the workloads will lies on the database server as well as the regular utilization of infrastructure is needed for the processing of every requested query. The text of the query and its result set will be profiled in the client side and it will be used to respond to future subdued query to be executed.

2. OBJECTIVES

The objective of the study is to develop a model that will enhance the capability of SQL Queries by employing purging and elimination technique. Specifically it aims to;

1. Eliminate condition in the query before sending to the database for evaluation.
2. Profile a query.
3. Purge queries that are subdued by incoming query.
4. Reuse past query in responding to requested query.

3. METHODOLOGY

3.1 Elimination

The purpose on the presence of a condition in the query is designed to extract only those information from the database that meets the criteria. This scenario will limits the number of rows to be produced, thereby it also limits is ability to respond to future subdued queries to be executed because its result set when reused is only capable to answer subdued queries that are joined with similar condition. In order to avoid this, the requested query will undergo checking process to identify the existence of a condition. If a condition is detected, next step is removing it from the query before it will be sent to the database for evaluation. Conditions in the query will be distinguished by the existence of a "where" keyword in the query.

3.2 Query Profiling

Query Profiling will be applied after the query undergone elimination process. A folder that serves as the repository of unique requested query will be created to store the text of the query and its result set. A file will be created and it will contain the text of requested queries which referred to as the query logs [2]. The result set of the query will be exported as text file [3] and it will be stored in a row and column format. The uniqueness of the query is determined by its source where the information will come from and the included field. Before the text of the query will be registered in the query logs, it will be attached with QUERY INDENTIFICATION NUMBER, which is an auto number generated by the algorithm [4]. The generation of QIN number will start to one (1) and progresses

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 will used purposely to respond to queries that are subdued from the past queries. Requested Queries that are unable to be responded by profiled queries will be directed to source-out its data to the database and it will be deposited in the repository.

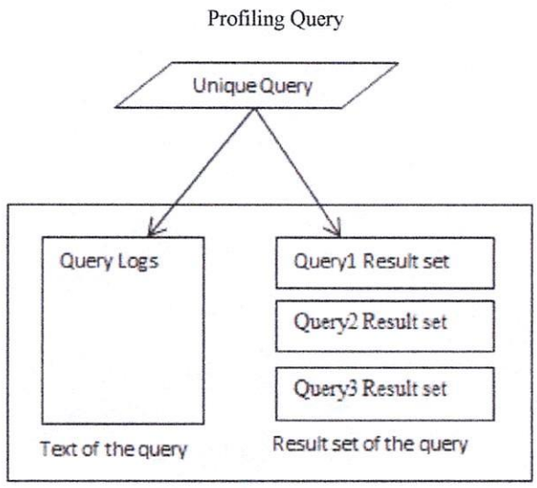

Figure 1. The text of the select statement will be stored in the query logs while its result set will be exported in the same repository.

3.3 Purging

As a way of maintaining the repository as well as to efficiently manage the space it utilizes, purging technique will be employed. The text of the profiled queries in the query logs will be compared to the text of the incoming query. If the text of the profiled queries stored in query logs matched or subdued by the text of the incoming query then the text of the profiled query in the query logs will be purged from the list along its result set [5]. Text of profiled queries are considered to be subdued if it meets the criteria; 1. It's source of data is similar to the source of the incoming query and 2. Its field/s are all existed in the incoming query. 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 method will not only reduce the number of queries stored in the repository but also to free some used space [6].

3.4 Method of Reusing Query

The text of the requested query will be 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. A profiled query in the query logs will become responsive to the requirements of the requested query if they have similar source and the field/s in the text of the requested query are all existing in the text of the profiled query. After determination, the QIN attached to the text of the profiled query will be used to pinpoint its result set followed by extraction and population for the purpose of reuse [7]. An index/es will be generated for the text of requested query by way of matching it to the fields of the text of subdued profiled query which eventually served as referenced field in the populated result set in the format of rows and column followed by iteration until the last data will be obtained.

Assume that this query requested;

and let assume that the query below is stored in the query logs,

Select
user_id,username,first_name,last_name,gender,password,status **from** user details

The index/es would be;

2 3 4 6

Reusing the query logs and its result set

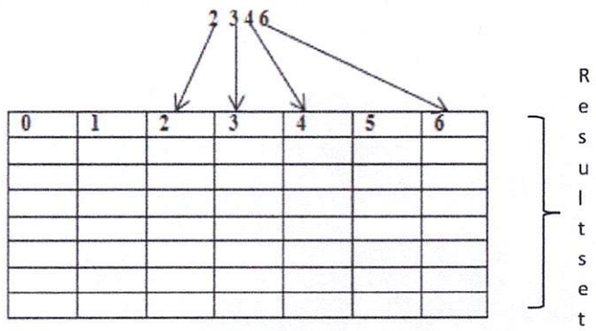

Figure 2. Utilization of the generated indexes of the query as bases in referencing to the populated result set of the query logs.

4 PERFORMANCE TEST

4.1 Testing

The model was tested on a server-client setup. In the server side, MySQL were used to create the database and the table and uploaded a dataset containing ten thousand (10,000) rows with seven (7) fields [8] [9]. The sample dataset was downloaded at https://www.sample-videos.com/download-sample-sql.php. The uploading process was done by utilizing the interface of phymyadmin. Seven (7) different queries were formulated and executed using the designed model installed in the client side. The queries are shown below.

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

Every query was executed twice because in the first execution, the model obtains its data from the database server and profiled it in the client side. In the second execution, the query is responded by utilizing the repository because it is now subdued to the previous query which means the query will be served locally [10]. Instead of obtaining the data to the database server, the request will be serve locally. This method avoided the utilization of the database server and network resources because the data will not be obtained from the original source. Two scenarios were used in executing the formulated queries which is the ascending and descending order. In ascending order of execution, there was a chance that the query will be

query will be purged because it is subdued by the incoming query. After executing the seven queries, the last 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 of the queries, the last query which is the **"Select user_id,username,first_name,last_name,gender,password,status from user_details"** was executed first. The next six (6) queries were not admitted to the repository because they are all subdued to the first query.

Table 1. Latency incurred in the execution of the query.

Queries	Latency for the First Execution (in seconds)	Latency in Second Execution (in seconds)
Select user_id from user_details	0.26764297485 4	0.077657938 0035
Select user_id,username from user_details	0.31285214424 1	0.123073101 044
Select user_id,username,first_name from user_details	0.36400580406 2	0.156177997 589
Select user_id,username,first_name ,last_name from user_details	0.35187792778	0.175606012 344
Select user_id,username,first_name ,last_name,gender from user_details	0.31780791282 7	0.204800844 193
Select user_id,username,first_name ,last_name,gender,password from user_details	0.63918900489 8	0.251147985 458
Select user_id,username,first_name ,last_name,gender,password, status from user_details	0.43551301956 2	0.264527797 699

The result shows that the latency in the second execution was decreased by fifty (50) percent as compared to the first execution across to all the executed query. One of the contributory factors for this reason is that in the second execution, the data were not travelled in the network instead the query was served locally.

4.2 Simulator

The performance of the query was simulated using the JMeter. [11] [12] It will be the source of response time graph. The response time is the elapsed time from the moment the query is sent to the server until the moment when the last bit of information has returned to the client [13]. Ten (10) users with ten (10) loops counts at one (1) transaction per seconds were set as the parameters for the testing of the query that accessed data from the database server with 10000 rows. It is decided that only the last query will undergo simulation because it is the query that retained in the query logs.

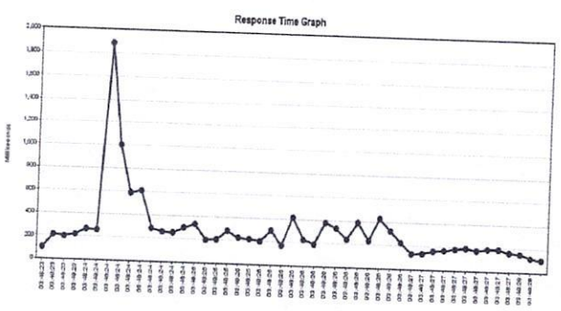

Figure 3. Response time graph when the query Select user_id,username,first_name,last_name,gender,password

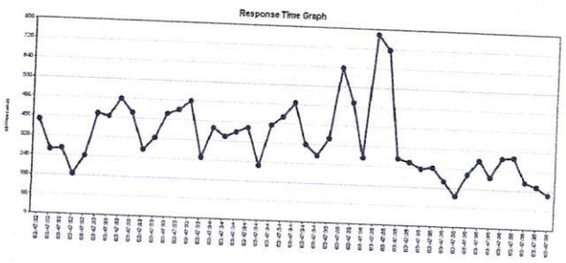

Figure 4. Response time graph when the query Select user_id,username,first_name,last_name,gender,password, status from user_details was executed for the second time.

The response time graph indicates that the figure 1 incurred higher latency as compared to figure 2. The highest response time per request in figure 1 is registered at almost 2000 milliseconds while in the figure 2 is almost 800 milliseconds.

5. CONCLUSION

In a scenario where the same information are to be accessed by a substantial number of users, a single access can be possibly cater the needs of the entire users which substantially reduced the amount of request being sent to the database which lead to decreased utilization of infrastructure.

6. ACKNOWLEDGMENTS

Thanks to my family for always there to support me, also to my adviser for unstinting and invaluable assistance, to CHED,TAU and AUF for giving me a chance to pursue this work.

REFERENCES

[1] West, M. (2013, January 8). *Storing data on the local client with LocalStorage*. Retrieved August 18, 2017, from http://blog.teamtreehouse.com/storing-data-on-the-client-with-localstorage: http://blog.teamtreehouse.com/

[2] Khoussainova, N., Kwon, Y., Liao, W.-T., Balazinska, M., Gatterbauer, W., & Suciu, D. (2011). Session-Based Browsing for More Effective Query Reuse. *International Conference on Scientific and Statistical Database Management*, 583-585.

[3] 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.

[4] 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.

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

[6] Shim, J., Scheuermann, P., & Vingralek, R. (2002). Dynamic caching of query results for decision support systems. *IEEE*.

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

[9] Simanjuntak, H. T., Simanjuntak2, L., Situmorang, G., & Saragih, A. (2015). QUERY RESPONSE TIME COMPARISON NOSQLDB MONGODB WITH SQLDB ORABLE. *JUTI*, 95-105.

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

[11] Flores, A., Ramírez, S., Toasa, R., Vargas, J., Barrionuevo, R. U., & Lavin, J. M. (2018). Performance Evaluation of NoSQL and SQL Queries in Response Time for the E-government. *IEEE.*

[12] FOTACHE, M., & HRUBARU, I. (2017). PERFORMANCE ANALYSIS OF TWO BIG DATA TECHNOLOGIES ON A CLOUD DISTRIBUTED ARCHITECTURE. RESULTS FOR NON NONAGGREGATE. *Scientific Annals of Economics and Business*, 21-50.

[13] Vahlas, N. (2018, August 1). *some-thoughts-on-stress-testing-web-applications-with-jmeter-part-2.* Retrieved from >Some thoughts on stress testing web applications with JMeter (part 2) : https://nico.vahlas.eu/2010/03/30/some-thoughts-on-stress-testing-web-applications-with-jmeter-part-2/