International Journal of Scientific & Engineering Research, Volume 4, Issue 9, September 2013
ISSN 2229-5518

A Framework for Predicting Query Response Time at Application Development Stage

Rekha Singhal

Abstract— In a typical database application environment, emphasis has been given on promising Service level Agreements (SLAs) for perceived query elapsed response time; the SQL queries are tested on the small size of database at application developmentstage, which may be a fraction of the production database. As time progresses the database grows and the earlier optimized queries may not hold SLA anymore. Once the application is deployed, it becomes difficult to modify the application or alter the production system. In this paper, we have discussed a framework for predicting the SQL query response time with growth of the database while being transparent to the production hardware, storage subsystem and DB Server. We have discussed various factors which can impact query response time and are also affected by the increase in the data size. We have presented a theoretical model for predicting the elapsed response time of SQL queries and also discussed a case study of Oracle 10g for implementing the proposed framework.

Index Terms— SQL, Query, Response Time, Parsing, Data Access, Large size database, Execution, Fetching.

1 Introduction

The advent of inexpensive computing and storage system has led to increase in autonomic computing and hence generation and storage of large sized data. The data intensive applications once launched in the production environment cannot be changed without a downtime; however data keeps growing as time progresses. The growth in data size may impact the application performance and violate the Service Level Agreement (SLAs) promised during the application development.

In a typical database application development, the testing of the application is done on a size of database which may not be realistic – especially for the banking and finance sector, as time progresses, customer base increases and hence the data grows in size. The SLA satisfied by the earlier application in terms of promised query response time etc, may not hold true for the same application with increased data size and workload. This is because of the testing methodology which was done on a subset of the projected data size. Sometimes, it is difficult to arrange all the resources for doing testing in the real environment. Large storage may be required to store trillions of records and conduct testing on that. However, an organization may not be willing to purchase this entire infrastructure at the testing stage. Moreover, even if resources are available, it may take days to load large database for a single query.

This raises the need to have a tool which can predict the performance of a database application on the same database (or DB), grown in size, a number of years later. It is required to have a tool to estimate the performance of the database application (or SQL queries) with increase in size of the database. With this one can take appropriate decisions to optimize/modify the queries, or application logic, or change DB server settings, or replace the hardware to avoid degradation in application performance over a time period. We propose a framework for predicting the response time of SQL queries with increase in size of the database such that it is independent of the hardware platform and database server on which the application is developed and tested.

The Section 2 discusses the related work in this direction. Section 3 illustrates the proposed framework. Section 4 derives the theoretical analysis for the performance metrics using the framework. Section 5 discusses a case study for use of the proposed framework. Finally, we conclude in Section 6.

2 Related Work

In database application domain, query response time has been a serious concern. In [1, 2, 3, 8 and 10], authors have discussed ways of optimizing queries to improve query response time. Broder [1] uses dynamic programming techniques with two levels for query evaluations - first level does approximate evaluations of query's terms and then at second level query is fully evaluated. With this they could eliminate full evaluations and hence improve query performance. In [10], authors consider progressive optimization of query instead of optimizing it using estimated cardinality of the tables in the query. They continuously check the estimated cardinality with actual cardinality perceived by query and stops to do re-optimize if the deviation is too large. Tolga [8] discussed creating a query execution plan on the fly using knowledge about load, communication delay on the system and the query optimizer's execution plan.

Huaiming [4] proposes a prediction model to forecast query predicates assuming short time events which have similarity in predicates and then to choose them for speculative execution to optimize the query response time. However, all of them optimize the queries for the current system and the existing database size; these optimizations alone may not give satisfactory results as database size increases over a period of time. One needs a tool to understand the performance of the optimized query for large data size which is generally the case for data intensive application.

Perros [6] uses regression model for predicting response time of query on database system. In [5], author proposes an online performance model for database appliances using an experiment-driven statistical modeling approach. They use a Bayesian approach and build novel Gaussian models that take into

International Journal of Scientific & Engineering Research Volume 4, Issue 9, September 2013
ISSN 2229-5518

account the interaction among concurrently executing queries and predict response times of individual DB queries. However, none of them have addressed the prediction of query response time with increase in database size at the application development stage.

Large size database has been always been an issue for performance especially in data warehouse system for SQL read queries. Kraft [14] attempts to optimize a sequence of SQL queries which are part of a single OLAP query. They exploit the semantics of these related queries in sequence; rewrite them by modifying at the query optimization layer and finally sending for execution.

Some of the researchers [15] have addressed this issue by compromising on query result which is an approximation of actual result. Actually, most of the data warehouse users may prefer to have faster response time than to actual result. However, in OLTP, the accuracy of retrieved data cannot be compromised. A query slow down is implicit with increase in data size, however, one may like to know the extent of degradation. Therefore, it is required to predict query response time for the projected large size database by doing extrapolation of the measures taken at the initial database size. Reference [12] has put up a framework using measurements to observe the query response time for grid architecture with variation of various parameters including the doubling of grid system which includes size of database as well as workload. However, this does not validate the query response time variation with increased database size while addressing various DBMS features. Their entire focus has been on the grid system as whole. However, we propose a framework to estimate a query performance on a large size database at application development stage while concentrating on DB behavior while processing a query.

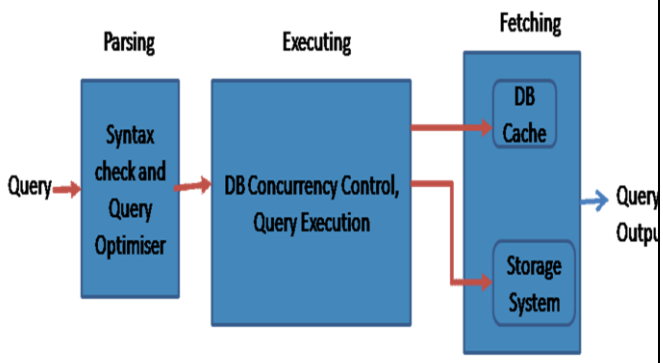

Fig. 1. Stages of Query Processing

3 Framework

We consider a two tier architecture where the application is hosted on the database server to avoid the time delays which may be introduced in the query result due to the query processing at the web server.

The response time for a query on database system depends on followings.

- Design of the query: Any operational problem may be formulated in form of SQL query; there may be many SQL constructions possible for the same problem. For example, use of hints, use of joins instead of sub query and many others techniques as discussed in [16] which may improve the query response time.
- Database schema: Use of indexes, table partitioning and de-normalization may affect the read query response time.
- Database server: Concurrency control techniques and query optimizer choices may affect the query response time. Several system level settings such as size of database cache, library cache , database operation cache and shared pool can directly impact the query performance.
- Workload on the server- number of concurrent queries, size as well as type of transactions may imapact the performance of a query. Most of application developer are interested in query performance in isolation, where workload could be stated as 1, and this may not have significant impact on query performance.
- Disk Subsystem: Data access time from disk subsystem can affect the query response time. This in turn may depend on how data is accessed, storage cache, storage hierarchy and storage hardware.
- Hardware platform: Speed of CPU, number of cores/processor, size of memory guides the resources which could be available for DB server and hence may impact the query performance.

Please, note that the affect of the hardware platform and disk subsystem on query performance do not vary with size of the database.

As time progresses, both database size as well as transaction workload may increase on the system which can affect the query response time. Both problems are orthogonal to each other. The variation of query response with increase in the workload on the system can be modeled for a fixed size of database. The concurrent workload model can be plugged in our framework while doing the prediction. Similary, one may model query performance with growth of database size keeping workload fixed (say query in isolation) and which may get plugged in the final framework along with concurrency model.

A query with a specific design on a specific DB server with a particular schema may perceive best response time, however as database size grows that query design may no longer yields best results. Moreover, schema may require changes in terms of index creation etc. to promise the same performance for the

International Journal of Scientific & Engineering Research Volume 4, Issue 9, September 2013
ISSN 2229-5518

query on the new large sized database. Various types of caches at DB server may need to be upgraded to improve the response time of query on database of increased size. The prediction tool may be used to estimate the performance of a query in future. If it is not acceptable, the user has an option to modify the query design or change DB server settings to get desired performance. We shall look at query processing at fine granularity level to formulate a framework which shows the parameters which change with increase in database size and affect the query response time as well.

A SQL query initiated through an application goes through three main stages during its processing – parsing, execution and fetching as shown in Fig 1.

3.1 Parsing

In this phase a SQL query is parsed to check its syntax and run through the query optimizer to decide its path of execution. Time taken to parse a query depends on how it is structured, e.g. use of bind variables in Oracle reduces the number of hard parse and hence the elapsed time. The elapsed time at this stage is not dependent on the size of the database; however the path chosen by the query optimizer may depend on size of the tables involved in the SQL query. The structure of a query plays a very important role here in deciding the path of execution and hence the type and number of operations, which in turn may affect the query response time. For example, use of hints may direct the query optimizer to use indexes or hash join which could speed up the query. Also, an absence of index may force the DB to do full scan of the database to answer the query which may lead to increase in elapsed time with increase in size of database.

3.2 Execution

Once a query is parsed it is ready for execution. During this phase, the query is executed which may involve a sequence of computations and fetching operations, from storage subsystem, overlapped with each other. However, contributions to query response time from both execution and fetching are disjoint. The execution phase primarily contributes towards the computations in the query. Therefore, we model them as two separate phases in query processing as shown in Fig 1.

In execution phase, DB concurrency control unit and hardware platform have critical role to play in deciding the query response time. The path chosen by query optimizer as discussed in the above phase is executed in this phase. The type and number of operations to be executed contribute to the query elapsed time. The cost of these operations is dependent on the size of database. For example, the size of index depends on the size of database; the hash join operation will be dependent on the size of tables involved in the join etc.

First, DB concurrency control mechanism decides whether the query can be scheduled for execution based on its conflict with other queries executing in parallel. This leads to waiting time for the query before it is scheduled for execution, which gets added up to the elapsed time. Probabilistically, larger the size of database, lesser is the chances for queries to conflict. In other words waiting time due to conflict may get reduced with increase in size of the database. However, if a query happens to access large data sets, then its high execution time may increase the waiting time for the conflicting queries.

Once the query is scheduled for execution, it is up to the OS to execute it either in parallel with other queries each on different processor (inter query parallelism), or it may execute single query on multiple processors (intra query parallelism), or pipe lining or serial execution in case of single processor. Each of these modes of executions will have affect on query response time, however the choice of mode of execution is independent of the size of the database, and this will not change with growth of database.

3.3 Fetching

An execution of a query will definitely leads to retrieving and may be modifications of data, (in case of DML queries), from the database. A record may be returned from the database server cache or it may be required to get fetched from the storage subsystem where the database is stored. The query elapsed time is small in the former case as compare to the later. This choice depends on the caching policy of the DB server as well as the size of system cache. This is independent of the size of database. However, probability of finding a requested record in cache is more for a small size database.

In other case, if the record is not in the cache, the record is fetched from the disk subsystem. A storage subsystem could be a single hard disk, SAN or JBOD. The time elapsed in fetching a record will depend on the performance of the disk subsystem which will contribute to the total query response time. Accessing (reading or modifications) a record from the disk subsystem is independent of the size of the database- i.e. it does not change with increase in the database growth.

3.4 Extrapolation

Based on the phases of query processing as discussed above, we can model each of the phase as function of size of database and use them for estimating a query response time. The framework for predicting query response time with grwoth in database is shown in Fig 2.

The prediction tool takes as input the SQL query under evaluation, the database schema, the database/tables size, current DB server settings and the infrastructure details which include hardware details such as memory size, number of cores etc. These inputs could be specified in a language which could be interpreted by the tool. The tool shall examine the query and estimate the given query's response time consulting the various models as shown in Fig 2. Each of these units could be a mathematical model or may be based on experimental results. We are currently working on building these models and shall publish the results later.

International Journal of Scientific & Engineering Research Volume 4, Issue 9, September 2013
ISSN 2229-5518

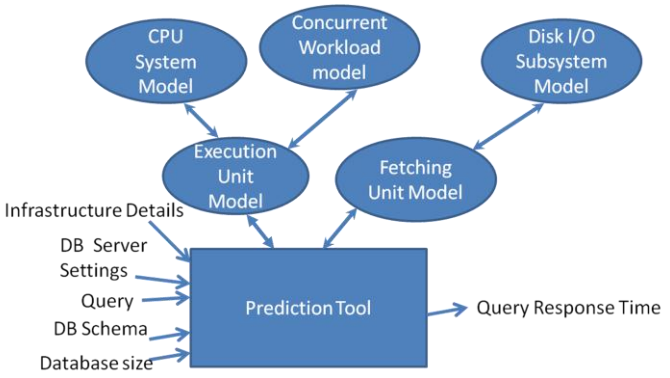

Fig. 2. Framework for Query Elapsed Response Time Prediction

Fig 2 shows a generalized framework, however, this could be curtailed depending on the required application. For example, at application development stage, one may be more interested to know performance of query in isolation to modify the query design or schema design. In such a case, workload is a single query; therefore, framework does not need Concurrent Workload Model. Similary, semantics knowledge of query may help in limiting the focus of the framework. A disk intensive query environment may need sophisticated Disk I/O Subsystem Model and a simple (e.g. a linear a mathematical) Execution Unit model may be sufficient. Further, one may use data access pattern of queries to model the Disk access time. A query may access data as a sequence of data blocks (Full Table Scan) or using an index. Further, index scan could be either using Primary Key Index or Secondary Key Index (or Non Primary Key Index). In these entire scans data acess pattern is different as shown in Fig 2a, which impact the disk access time. Similary, a compute intensive query may require proper modeling of CPU system Model.

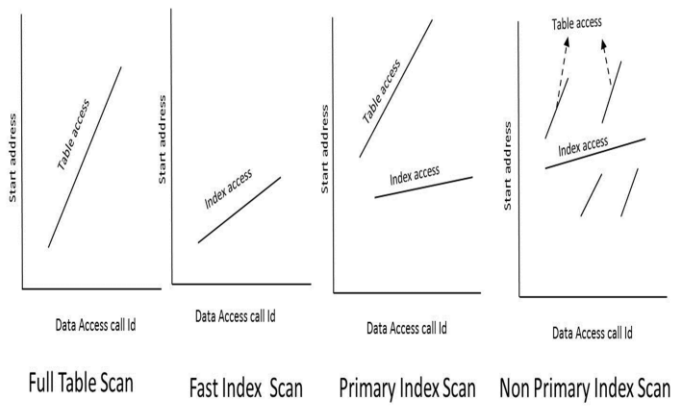

Fig. 2a. Data Access pattern

4 Theoretical Formulation

A query response time quantitatively could be function of the followings.

1. Size of the query result (expected number of rows * expected size of each row) which may be dependent on size of database.
2. Concurrency control mechanism of the DB server.
3. Query execution cost which may depend on the access path chosen by the query optimizer, size of server's various caches.
4. Number of processors in the DB server
5. Size of the Cache at DB server which is dependent on the hardware platform.
6. Disk subsystem performance.

With reference to Fig 1, query processing time (QRT) could be formulated as:

$$QRT = Parsing\ Time + Fetching\ Time + Wait\ Time + Execution\ Time \quad (1)$$

where, *Wait Time* is the extra time contributed in a query execution due to presence of other queries on the system.

Let us assume that we will ignore all those time constants which are invariant to the size of database so that we can have only those components which will contribute to the elapsed time with growth of database size.

Consider a database of size *'N"*. Parsing time is independent of the database size. So we will ignore this component. Wait Time may differ for different database servers due to their different concurrency control algorithm and policies. It primarily depends on the number of the conflicting queries , *Q*, and the type of queries. For heterogeneous mix of queries, it may be function of database size as well. Therefore, this part of the query execution time shall be provided by the database concurrency control model as shown in Fig 2, and represented as *W(Q,N)*.

Execution time depends on two parameters – system architecture (number of processors, memory size etc.) and execution path given by the parsing unit. The former is independent of the size of the database, however later depends highly on the database size. The expected number of rows to be returned by the DB Server increases with database size- the rate of increase will depend on the model of query optimizer. Let's assume,

$$Execution\ time = K(N) + Constants \quad (2)$$

where K(N) models the number of operations with their costs and the cardinality set of the data processed by the query.

Fetching time depends on the DB cache size and the disk subsystem. Let's assume ,

$$Fetching\ Time = F(N) + H(N) \times T_{Cache} + \big(1 - H(N)\big) \times T_{Storagesubsystem} \quad (3)$$

where *F(N)* returns number of fetches which depends on the path executed at the execution phase and the size of database.(Please note we assume that each fetch returns same number of bytes and all fetches together corresponds to the cardinality set of the query result.). *H(N)* is the hit ratio for DB cache- this decreases with increase in the number of concurrent transactions as well as with size of database for uniform data access, T_{Cache} is the average time taken to retrieve a record from cache, $T_{Storagesubsystem}$ is the average time taken to access a record from the storage subsystem.

Therefore, using equations 1, 2 and 3, we get *QRT* as

$$QRT = F(N) + W(Q,N) + K(N) + H(N) * T_{Cache} + \big(1 - H(N)\big) * T_{Storagesubsystem} \quad (4)$$

Once we feed in these various models we can get approximate query response time and can observe its behaviour with

International Journal of Scientific & Engineering Research Volume 4, Issue 9, September 2013
ISSN 2229-5518

increase in the size of the database.

5 Case Study

We consider a specific DB server, Oracle 10g [16], as a case study to show the applicability of the framework. Oracle, in dedicated mode, has many monitoring processes and 'oracleorcl' for processing the queries. The Orcl process will do all the job of parsing, executing and fetcing. Writing, in case of update and insert operation, is done by a separate process called DBWR.

To estimate a query response time on Oracle 10g, according to the proposed framework, we need model for functions *K(N), Q, F(N), H(N),* T_{Cache} and $\boldsymbol{T_{Storagesubsystem}}$.

T_{Cache} and $\boldsymbol{T_{Storagesubsystem}}$. can be obtained from the inputted infrastructure details, the CPU system and Disk I/O subsystem models. These are in terms of time per fetch. The value of *Q* is dependent on the concurrency control policy of the DB server and may be obtained from the DB server model. A DB server using strong serialization approach may have high value for *Q*, however DB server using semantics of applications may work with weak serialization and decrease the number conflicting transactions, hence low value for *Q*. This can be obtained from Concurrent workload model.

F(N) is dependent on size of the query result. It returns the number of fetches required to produce the query result. It takes certain inputs, such as DB schema, as shown in Fig 3. In case of oracle, DB schema can be imported in the model system using dump facility. A SQL read may only read from the storage subsystem, while a SQL update may perform both read and write on the storage subsystem. For simplicity, we assume only SQL read so that in later case the time taken will be just twice of the former one. The challenge here is to estimate the number of rows returned by the query which will be independent of the access path chosen by the execution unit, but dependent on the size of database. Oracle has a tool called Explain plan[16], which gives an estimate of number of rows returned by using information about DB schema and size of various tables in the database. Another tool, TkProf, gives an exact number of fetches performed by the query during its execution. Since we do not have mechanism of executing query on the future database, we can do measurements by executing a test workload and build a model using TkProf. Both these tools can be used together, as shown in Fig 3, to build *F(N)* which can return size of query result on inputting future size of database.

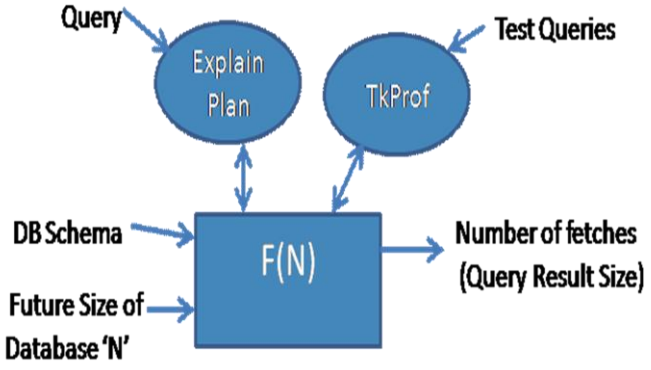

Fig. 3 Model for Fetching Unit, F(N).

K(N) is the expected execution cost of the query which depends on the DB server system parameters such as cache etc, query size result and the cost of the access path which includes operations performed for executing the query. Please note that the cost of the operations performed after fetching such as 'DISTINCT' etc. is counted in *K(N)* only. The query size result can be provided by F(N). As discussed before, in Oracle 10g, Explain Plans can give expected access path followed by the query for execution and TkProf can provide actual execution cost of the query. The breakup of the execution cost in terms of cache reads and physical reads can be obtained by running AWR[16] reports in Oracle. Some of the operations' cost such as join etc. depends on the DB server settings (e.g. higher PGA_TARGET_AREA, lower cost of sort operation), which can be obtained from AWR reports. These three tool can be used together, as shown in Fig 4, to model *K(N)*. As discussed above, in Oracle, *H(N)* and *W(Q,N)* can be modeled by doing measurements using AWR[16] on the given system.

Consider TPC-H [17] benchmarks with one of its query Q6.sql, which is a '*select on lineitem table with filter conditions*', and its response time to be predicted for DB size 8GB with no conflicting workload. Assuming, we have Oracle DB model, CPU model and Local disk access for data, the model will work as follows.

W(0,N) = 0, from the AWR reports. *K(8GB)* will be obtained from DB model in terms of number of operations and per operation unit response time, which turn out to be 10 sec. Disk System Model shows that = 0.00000002 sec and = 0.00008 sec. *F(8GB)* will be 877591 logical reads as given by the Fetching model. AWR reports shows *H(8GB)*=0.1, Therefore putting values in equation 4,

QRT = 10 +0+877591*(0.1*0.00000002+0.9*0.00008)

QRT = 73 secs and the actual measurements shows ERT to be 63 secs. There is an error of 15%.

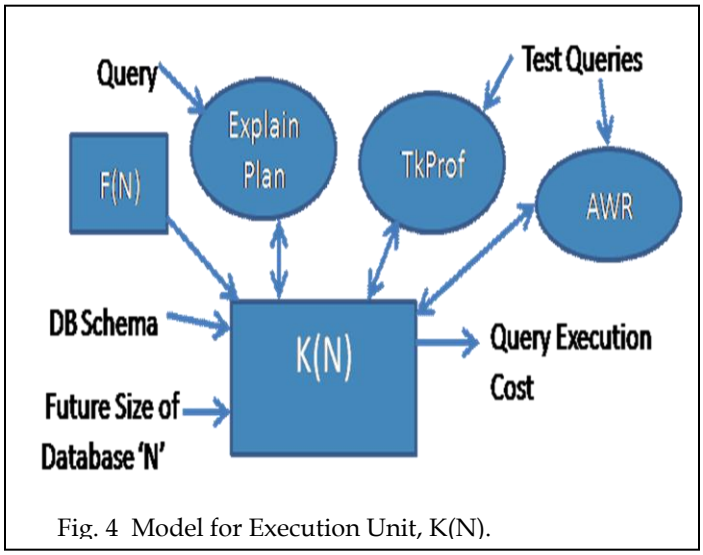

Fig. 4 Model for Execution Unit, K(N).

International Journal of Scientific & Engineering Research Volume 4, Issue 9, September 2013
ISSN 2229-5518

6 Conclusion

The paper has discussed a need to predict query response time for large database system without actually running the query at application development stage. We have presented a framework to estimate the query response time, for large database system, which depends on the parameters affected by the growth of the database. The framework consists of DB concurrency model, CPU system model, IO Subsystem Model which in turn gives rise to Fetching and Execution Model. These models will be developed in our future work. We have also given a theoretical formulation of the framework. We have presented a case study of Oracle 10g on using the proposed framework for estimating the query response time on large sized database. Our future work will detail out the model for the executing and fetching units of the proposed framework.

References

[1] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, J. Y. Zien. Efficient Query Evaluation Using a Two-Level Retrieval Process. CIKM 2003.

[2] Bini, T.A.; Lange, A.; Sunye, M.S.; Silva, F.; Stableness in large join query optimization,Computer and Information Sciences, 2009.

[3] Hameurlain, A.; Morvan, F., An overview of parallel query optimization in relational systems, Proceedings of Database and Expert Systems Applications, 2000.

[4] Song Huaiming; Wang Yang; An Mingyuan; Wang Weiping; Sun Ninghui; Query Prediction in Large Scale Data Intensive Event Stream Analysis Systems, Proceedings of Grid and Cooperative Computing, 2008.

[5] Muhammad B. S. , Umar F. M., Omar Z. K., Ashraf A., Pascal P., David J. T, A bayesian approach to online performance modeling for database appliances using gaussian models, Proceedings of the 8th ACM international conference on Autonomic computing, June 2011.

[6] H. G. Perros, A model for predicting the response time of an on-line system for electronic fund transfer, ACM SIGMETRICS Performance Evaluation Review, Volume 12 Issue 1, Winter 1982-1983

[7] Rodney L., Managing Data Growth in SQL Server, 21 January 2010, http://www.simple-talk.com/sql/

[8] Tolga U., Michael J. F. , Laurent A., Cost-based Query Scrambling for Initial Delays, ACM SIGMOD Conference 1998

[9] Minos N.G, Philip B.G, Approximate Query Processing: Taming the TeraBytes, In processdings of SIGMOD 2011

[10] Volker M.,Vijayshankar R., David E. S., Guy M. L.,Hamid P., Robust Query Processing through Progressive Optimization, SIGMOD Conference, 2004.

[11] Deepak S.,Umesh S., Query Optimizer Model for Performance Enhancement of Data Mining Based Query, International Journal of Computer Science & Communication, Vol. 1, No. 1, January-June 2010, pp. 235-237.

[12] L. Field, M. W. Schulz, F. Ehm Scalability and Performance Analysis of the EGEE Information System, International Conference on Computing in High Energy and Nuclear Physics (CHEP'07) IOP Publishing, 2007

[13] Florian W., Cesar Galindo-Legaria, Counting, Enumerating, and Sampling of Execution Plans in a Cost-Based Query Optimizer, ACM SIGMOD 2000.

[14] T.Kraft, H. Scwarz, R.Rantazu and B. Mitschang, Coarse- grained optimization:Techniques for rewriting SQL sequences, 29th conference of VLDB, 2003.

[15] M. Garofalakis, P. Gibbons,Approximate Query Processing:Taming the tetrabytes, Information Science, Research Center, Bell Labs, VLDB, 2001.

[16] R. Niemiec, Oracle Database 10g performance Tuning, Oracle Press, 2007.

[17] TPC-H Benchmark, url:http://www.tpc.org/tpch.