Query Optimization Based on Time Scheduling Approach

Wajeb Gharibi[1] and Ayman Mousa[2]

College of Computer Science and Information Systems, Jazan University, Saudi Arabia
[1]gharibi@jazanu.edu.sa
[2]ayman_mosa2004@yahoo.com

Abstract

Distributed database systems suffer from many difficulties. The most common difficulty is the environment where such systems mostly are running on unpredictable and volatile environments. So it is difficult to produce efficient database query optimization based on information available at compilation time. The performance of re-optimization techniques which are used by many systems suffers from problems. This paper attempts to reduce query re-optimization by executing a set of simple predefined queries in predefined time scheduling approach to collect estimates of statistics at runtime. For this purpose, a timer object with adaptive query processing systems is proposed. Experimental result is given to demonstrate the performance of real queries. This result shows significant performance improvements is obtained compared with other traditional approaches when the proposed object timer is used.

Index Terms— Query optimization, adaptive query processing, query reoptimization, distributed database

1. Introduction

Research in distributed database systems has popularized the mediator/wrapper architecture. The mediator provides a uniform interface to query heterogeneous data sources while wrappers map the uniform interface into the data source interfaces [11]. In this context, processing a query consists in sending sub-queries to data source wrappers, and then integrating the sub-query results at the mediator level to produce the final response. One of the key reasons for the success of relational database technology is the use of declarative languages and query optimization. The user can just specify what data needs to be retrieved and the database takes over the task of finding the most efficient method of retrieving that data. It is the job of the query optimizer to evaluate alternative methods of executing a query, and selecting the best alternative [2].

Several techniques have been proposed to improve traditional query optimization. These techniques include better statistics [12], new algorithms for optimization, and adaptive architectures for execution [3]. A very promising technique in this direction is reoptimization, where the optimization and the execution stages of processing a query are interleaved, possibly multiple times, over the running time of the query [6][7][14]. Current re-optimizers take a reactive approach to re-optimization. In this case use a traditional optimizer is used to generate a plan, then track statistics and respond to estimation errors and resulting sub-optimality detected in the plan during execution. Reactive reoptimization is limited by its use of an optimizer that does not incorporate issues affecting reoptimization, and suffers from several shortcomings. Also proactive reoptimization approach [5] is used during optimization to generate robust and switch able plans and it is used random-sample processing for every query execution, consequently it is suffered from some negative points affecting from reoptimization process that mention in section 3.

This paper presents query optimization based on time scheduling approach to reduce query reoptimization. For this purpose, we added a timer object with adaptive query processing systems to handle the problems with effect reoptimization. A timer object built to execute a set of simple predefined queries in predefined time scheduling to collect estimates of statistics about the database quickly, accurately, and efficiently at runtime. When the timer object fires (multiple times) executes a set of simple queries that were specify, and the amount of time specify between executions of the timer object perform execution of the real queries. Finally we present simulation experimental that demonstrated our anew approach results in significant improvements on the performance of the real queries.

The remainder of this paper is organized as follows. Section 2 discusses related work to query reoptimization techniques. Section 3 focuses on the problems with reactive reoptimization and the problems with proactive reoptimization. Section 4

978-1-4799-2096-9/13/$31.00 ©2013 IEEE

shows time scheduling approach. Section 5 a typical example with simulation results. Finally section 6 presents conclusions.

2. Related Work

In the most cases the distributed database environment is running on unpredictable and volatile environments. So it is difficult to produce efficient database query optimization based on information available at compilation time. A solution to this problem is to exploit information that becomes available at query runtime and adapt the query plan to changing environmental conditions during execution. This section presents adaptive query plan reoptimization techniques which were used to modify query plan dynamically at runtime.

Mid-query Re-optimization [6], dealt with mid-estimated intermediate result sizes in a lazy-evaluation way, by extending a traditional execution system so that it could re-invoke the optimizer after completion of any pipelined segment of the initial plan. Their system inserts statistical monitors where they incur low overhead but provide information about when re-optimization is useful.

Query Scrambling [17], was developed specifically to cope with unexpected delays that arise when processing distributed queries in a wide-area network. With Query Scrambling, a query is initially executed according to a plan generated by query optimizer. If, a significant performance problem is detected during the execution, the query plan is modified on the fly. Query Scrambling uses two basic techniques to cope with unexpected delays rescheduling and operator synthesis.

Tukwila system [8], addressing adaptiveness in data integration environment. Adaptiveness is introduced at two levels: In the first level, Adaptiveness is deployed by annotating initial query plans by ECA rules (event-condition-action). These rules check some conditions when certain events occur and subsequently trigger the execution of some actions. For the second level of adaptiveness, two operators are used: dynamic collector operator dynamically chooses relevant sources when a union involves data from possibly overlapping or redundant sources, and the double pipelined hash join operator is a symmetric and incremental join.

Eddies algorithm [3], an eddy encapsulates the ordering of operators by dynamically routing tuples through them. The idea is that there are times during the processing of a binary operator (e.g., join, union) when it is possible to modify the order of the inputs without modifying any state in the operator.

Convergent Query Processing [18], present a logical query optimization framework and a set of adaptive techniques, In high level, it is similar to several previous methods, during execution do monitor the costs of operations and size of intermediate results, if the plan is poor, must replace it with one that is expected to perform better.

Progressive query optimization (POP) [9], provides a plan "insurance policy" by lazily triggering re-optimization in the midst of query execution whenever cardinality estimation errors indicate that the QEP might be sub-optimal. It does this by adding one or more checkpoint operators (CHECK), which compare the optimizer's cardinality estimates with the actual number of rows processed thus far, and trigger reoptimization if a pre-determined threshold on the error is exceeded. Note, the merger POP technique and use a Timer Object will be give high performance in the execution of the queries.

Proactive re-optimization approach [5] incorporates three techniques:

- Bounding boxes are computed around estimates of statistics to represent the uncertainty in these estimates.
- The bounding boxes are used during optimization to generate robust and switchable plans that minimize the need for reoptimization and the loss of pipelined work.
- Random-sample processing is merged with regular query execution to collect statistics at run-time.

Note, not use a Timer Object with this approach because the system will be poor performance in the execution of the queries. Fig.1 presents related a Timer Object with adaptive query plan reoptimization techniques.

3. Problems with Reactive and Proactive Reoptimization

The query optimizers use a plan-first execute-next approach; the optimizer enumerates plans, computes the cost of each plan, and picks the plan with lowest cost [16]. Current re-optimizers take a reactive approach to reoptimization, they use a traditional optimizer to generate a plan, and then track statistics and respond to estimation errors and resulting sub-optimality detected in the plan during execution. Reactive reoptimization is limited by its use of an optimizer that does not incorporate issues affecting re-optimization, and suffers from at least three shortcomings:

- The optimizer may pick plans whose performance depends heavily on uncertain statistics, making re-optimization very likely;

– The partial work done in a pipelined plan is lost when reoptimization is triggered and the plan is changed;

– The ability to collect estimates statistics quickly and accurately during query execution is limited. Consequently, when reoptimization is triggered, the optimizer may make new mistakes, leading potentially to thrashing.

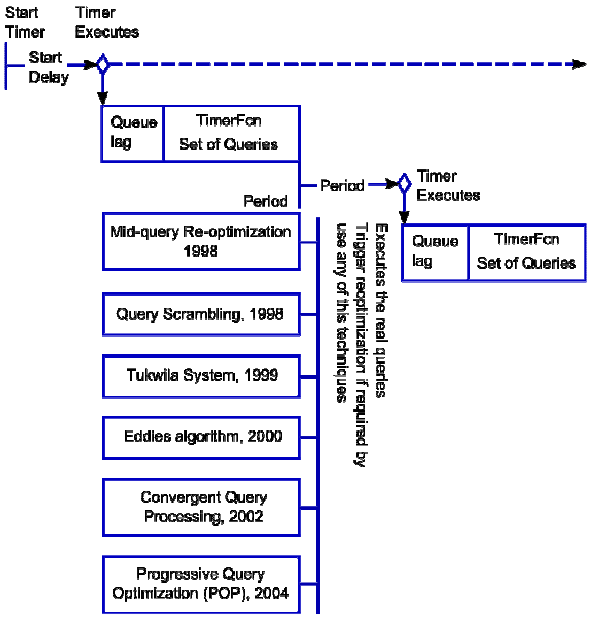

Figure1. Timer Object with query re-optimization techniques

The proactive reoptimization approach [5] used during optimization to generate robust and switch-able plans and it is used random-sample processing for every query execution, consequently it is suffer from some negative points affecting from reoptimization process:

– It defines a robust plan as one that is “near optimal" but gives no formulae for comparing a robust plan with an optimal one. How close is good enough?

– The idea of robust plans is very appealing, but this approach gives little evidence. Switchable plans appear to be more promising, although they do not always allow the reuse of work.

– Every execution query, executes random-sample tuples which is merged with regular query execution to collect statistics, hence incur high overhead. And before this during optimization generate robust and switchable plans to select execution plan this making delay.

– This approach would have been much easier to understand if they had gone ahead and drawn detailed bounding boxes for each of their examples. And it was difficult to keep track of them without drawing a box in the margins and marking it.

–

4. Time Scheduling Approach

Query optimization based on time scheduling approach is aimed to reduce query reoptimization. This approach executes a set of simple predefined queries in predefined time scheduling to collect estimates statistics about the database quickly, accurately, and efficiently at runtime. For this purpose, a Timer Object [10] incorporate with adaptive query processing systems to handle the problems with reactive and proactive reoptimization approaches that were as mentioned in section III. When the amount of time specified by the timer object elapses (much time elapses before the timer fires) and the timer object fires (multiple times to execute a timer object) executes a set of simple queries that were specify, and the amount of time specify (the period time) between executions of the timer object perform execution of the real queries. Fig.2 illustrated architecture a Timer Object with adaptive query processing systems (AQP) [4].

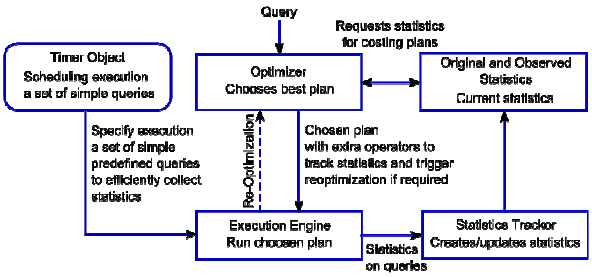

Figure 2. Architecture a Timer Object with AQP

Every query in a set of simple queries is select primary key field (numeric field, because this field is very small size take littleness execution time) from all tables in database to efficiently collect statistics on the execution environment such as CPU workload, network traffic, available memory and resource availability. The different query execution plans share the same system statistics, predictions, and on the estimation of query characteristics (e.g. the data selectivities of operators are computed on the fly and the number of the tuples in every table).

Database administrator invokes these simple queries by running a timer objects multiple times every day on period specify to collect new estimates statistics and to bring existing ones up-to-date statistics continuous at runtime. The following general configuration the simple predefined queries (e.g. SQL language):

Q1: SELECT Fieldname (primary key or numeric field)

FROM Tablename1
Q2: SELECT Fieldname (primary key or numeric field)
FROM Tablename2
⋮
Qn: SELECT Fieldname (primary key or numeric field)
FROM Tablenamen

A. Use a Timer Object

- Create a timer object;
- Specify the set of the queries will be executed when the timer object fires, and control other aspects of the timer object behavior;
- Start and stop the timer object;
- Delete the timer object when done the work.

B. Timer Object Execution Modes

The timer object supports several execution modes that determine how it schedules the timer object function (TimerFcn, it is the some of commands or file to execute when the timer fires) for execution. Specify the execution mode a timer object function Once or Multiple Times [10].

1) Executing a Timer Object Function Once

To execute a timer object function once, set the execution mode property to 'Single Shot'. This is the default execution mode. In this mode, the timer object starts the timer, after the time period specified (much time elapses before the timer fires), adds the timer (TimerFcn) to the execution queue. When the timer function finishes, the timer stops. Fig.3 illustrates the parts of the timer object execution for a single shot execution mode.

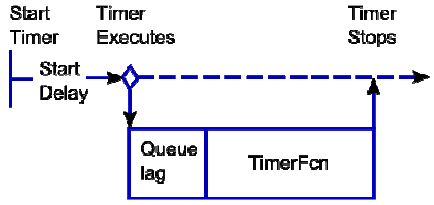

Figure 3. Timer Object execution (Single Shot Execution Mode)

The shaded area is shown in the Fig.3 where the label queue lag, represents the indeterminate amount of time between when the timer adds a timer function to the execution queue and when the function starts executing. The duration of this lag is dependent on what other processing happens to be doing at the time.

2) Executing a Timer Object Function Multiple Times

The timer object supports three multiple execution modes [9]:

- Fixed Rate;
- Fixed Delay;
- Fixed Spacing.

In many ways, these execution modes operate the same:

- The Tasks to Execute property specifies the number of times executions of the timer object function (TimerFcn).
- The Period property specifies the amount of time between executions of the timer object function.
- The Busy Mode property specifies how the timer object handles queuing of the timer object function when the previous execution of the function has not completed.

The execution modes differ only in where they start measuring the time period between executions. Figure 4 illustrates the difference between these modes. Note that the amount of time between executions (specified by the Period property) remains the same. Only the point at which execution begins is different [10].

3) Handling Function Queuing Conflicts

At busy times, in multiple execution scenarios, the timer may need to add the timer object function (TimerFcn) to the execution queue before the previously queued execution of the function has completed. The timer object handles this scenario by using the Busy Mode property. If the value of the Busy Mode property specifies 'drop', the timer object skips the execution of the timer function if the previously scheduled function has not already completed. If specify 'queue', the timer object waits until the currently executing function finishes before queuing the next execution of the timer object function. If the Busy Mode property is set to 'error', the timer object stops and executes the timer object error function (ErrorFcn), if one is specified [10].

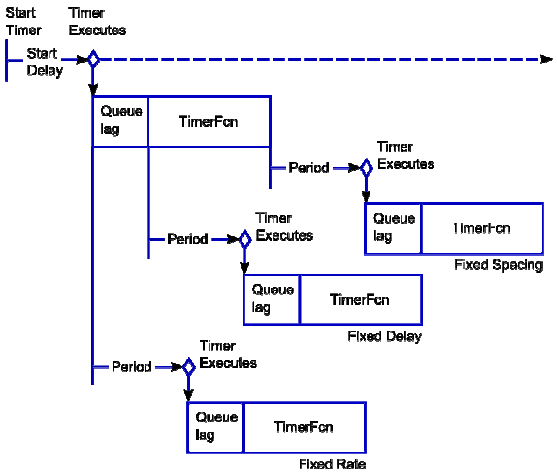

Figure 4. Differences between Execution Modes

5. Simulation Results

Typical case study is studied in this section, the effect of the Timer Object that scheduling the execution of set of the queries that were shown on real queries. All experiments were done on a 2.40 GHz Pentium Processor with 256 K.B cache memory, 256 MB of memory, and one Disk drive 80 GB. The Operating System which was used is Microsoft Windows XP professional version 2002. The simulation results are executed based on database Microsoft Access2000 is "Northwind.mdb", which contains seven tables. The programming tasks was built by MATLAB version 6.5. The following steps are executed to implement the proposed systems:

- Specify a set of the simple queries inside the file, every query select primary key (numeric field) from all tables in database to efficiently collect information about the database quickly, and accurately at runtime.
- A Timer Object is built for using to schedule the execution the file (previous point). When the timer object fires (multiple times) executes this file, the amount of time specify between executions of the timer object perform the execution of the real queries.
- Real queries are built to test as Q1, Q2, Q3, Q4, Q5, Q6, Q7, and Q8. These queries are varying in join the tables.

A. Simulation Experimental Results.

Step1: The first steps, the real queries Q1, Q2, Q3, Q4, Q5, Q6, Q7, and Q8 executed before execution the timer object t and calculated the values execution times of the real queries in variable vector as "Before Execution". The values of the execution times of these queries before execution the timer object t is shown in Fig.5.

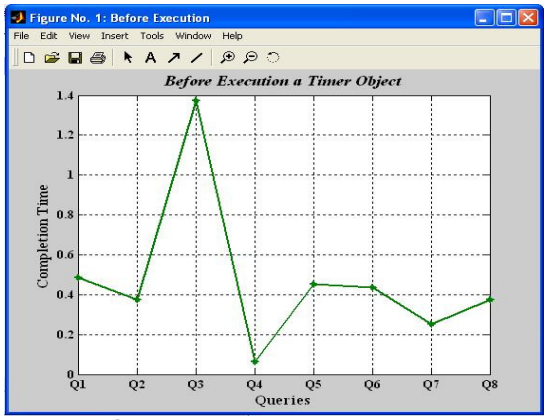

Figure 5. Completion times of the real queries before execution the Timer Object t

Step2: The Timer Object was built with the set values of the timer object properties that were specify. It was executed the file which contains a set of simple predefined queries in predefined time scheduling to collect estimates of statistics at runtime. When the amount of the time specified by the timer object t elapses (such as 5 secs) and the timer object fires (such as 5 times) executes this file, The amount of time specify (such as 60 secs) between executions of the timer object t perform the execution of the real queries.

In terminal the work of the Timer Object t (when done the work) has a list of properties that were specified previously whenever a Timer Object t built. A list of all properties of the Timer Object t after the running off is viewed in Fig.69.

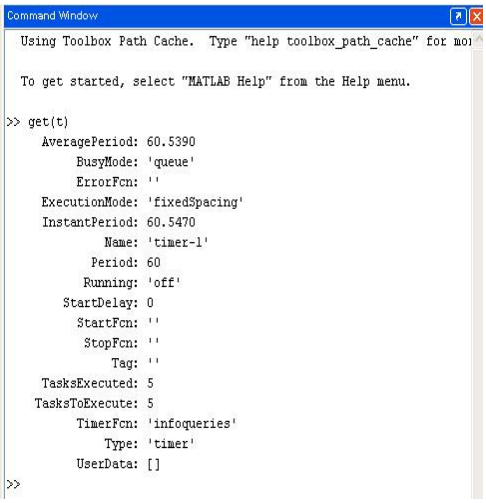

Figure 6. A list of all properties of the Timer Object t

Step3: Repeat execution the same of the real queries Q1, Q2, Q3, Q4, Q5, Q6, Q7,and Q8 there was executed in step1 after fire execution the Timer Object t and calculated the values execution times of the real queries in variable vector as "After Execution". The values of the execution times of these queries after fire execution the Timer Object t is shown in Fig.7.

Step4: We calculated the performance of the Timer Object t on the real queries Q1, Q2, Q3, Q4, Q5, Q6, Q7, and Q8:

1. The difference between completion times for ever real queries before and after fire execution the Timer Object t in the variable vector as "Difference".
2. Calculate the total completion time of the real queries before execution the Timer Object t in the variable vector as "Total Before Execution".

3. Calculate the total completion time of the real queries after fire execution the Timer Object t in the variable vector as “Total After Execution".

4. Calculate the total difference between total completion time of the real queries before and after execution the Timer Object t in the variable vector as "Total Difference".

5. Calculate the improvement in execution real queries in the variable vector as "Percentage Improvement”.

6. Calculate the gain time in the variable vector as "Gain Time" from using a Timer Object t.

7. Calculate the percentage gain time for every victim time in the variable vector as "Percentage Gain Time” from using a Timer Object *t*.

Figure 7. Completion times of the queries after fire execution the Timer Object t

Fig.8 gives the differences between the values of the execution times of the real queries Q1, Q2, Q3, Q4, Q5, Q6, Q7, and Q8 before execution the Timer Object t and the values of the execution times of the same real queries after fire execution the Timer Object t.

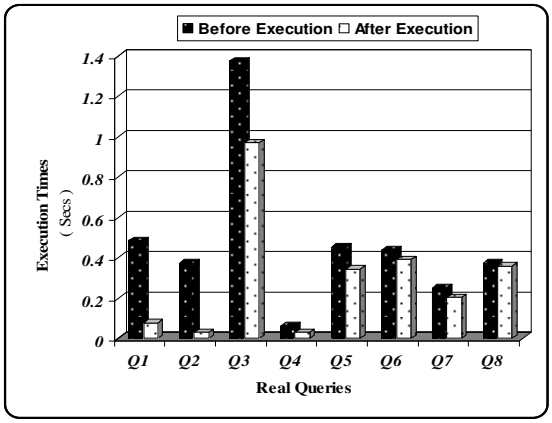

Figure 8. Execution times of the real queries before and after execution the Timer Object t

6. Conclusion

This paper introduces query optimization based on time scheduling approach which is aimed to reduce query re-optimization plan. This approach executes a set of simple predefined queries in predefined time scheduling to efficiently collect estimates statistics quickly and accurately at runtime. For this purpose, a Timer Object incorporates with adaptive query processing systems is proposed to handle the problems with reactive and proactive reoptimization techniques. When the amount of time specified by the timer object elapses , then the timer object fires executing a set of simple queries that were specify, and the amount of time specify (the period time) between executions of the timer object perform execution of the real queries. During this period time, the system is used any previous query reoptimization techniques (Reactive reoptimization approach) doing query execution plan reoptimization if required and this will be very little. Finally, simulation results demonstrated that there are significant improvements on the performance of the real queries due to the time scheduling process.

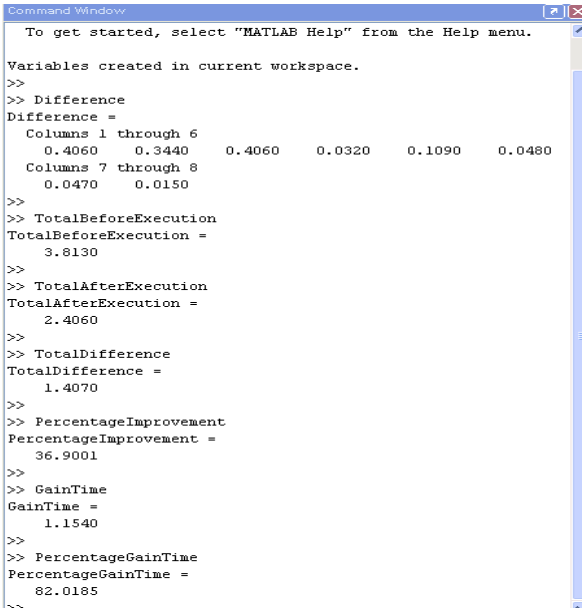

Fig.12. Improvements on the real queries Q1, Q2, Q3, Q4, Q5, Q6, Q7, and Q8 from use a Timer Object *t*

7. Acknowledgement

The authors would like to express their greatest thanks and gratitude to the Deanship of Scientific Research in Jazan University for supporting this research.

8. References

[1] Abdelkader H. and Franck M. "Evolution of Query Optimization Methods" Springer-Verlag Berlin Heidelberg, Transactions on Large-Scale Data & Knowledge Center System I, LNCS 5740, pp. 211–242, 2009.

[2] Alaa A., Emad A., and Mohammed O. " A Survey of Distributed Query Optimization" The International Arab Journal of Information Technology, Vol. 2, No. 1, pp. 48-57, January 2005.

[3] Avnur R. and Hellerstein J. "Eddies: Continuously Adaptive Query Processing". In Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data, pages 261–272, May 2000.

[4] Babu S. and Bizarro P. "Adaptive Query Processing in the Looking Glass ". In Proc. of Second Biennial Conf. On Innovative Data Systems Research (CIDR), Jan. 2005.

[5] Babu S., Bizarro P., DeWitt D. "Proactive Re-Optimization ". SIGMOD 2005, Baltimore, Maryland, USA, June 14-16, 2005

[6] Kabra N. and DeWitt D. "Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans ". In Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data, pages 106–117, June 1998.

[7] Ives Z., Halevy A., and Weld D. "Adapting to source properties in processing data integration queries ". In Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data, pages 395 – 406, June 2004.

[8] Ives Z., Florescu D., Friedman M., Levy A., and Weld D. "An Adaptive Query Execution System for Data Integration ". In Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data, pages 299-310, June 1999.

[9] Markl V., Raman V., Simmen D., Lohman G., and Pirahesh H. "Robust query processing through progressive optimization ". In Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data, pages 659–670, June 2004.

[10] MATLAB Programming Version 7, COPYRIGHT 1984–2006 by The MathWorks, www.mathworks.com.

[11] Ozsu T. and Valduriez P. "Principles of Distributed Database Systems ". 2nd Edition. Prentice Hall, 1999.

[12] Poosala V., Ioannidis E., Haas J., and Shekita J. "Improved Histograms for Selectivity Estimation of Range Predicates ". In Proc. of the 1996 ACM SIGMOD Intl. Conf. on Management of Data, pages 294–305 June 1996.

[13] Pund M., Jadhao S., Thakare P. "A Role of Query Optimization in Relational Database" International Journal of Scientific & Engineering Research, Vol. 2, Issue 1, pp. 24-33, Jan 2011.

[14] Salvarai P., Arputharai K., and Palaniappan K. "An optimizing Query Processor with an Efficient Caching Mechanism for Distributed Databases" The International Arab Journal of Information Technology, Vol. 3, No. 3, pp. 231-236, July 2006.

[15] Saurabh K.,Gaurav K.,Arjun V., and Mukul A. "Cost-Based Query Optimization with Heuristics" International Journal of Scientific & Engineering Research Vol. 2, Issue 9, pp. 1-6, Sept 2011.

[16] Selinger G., Astrahan M., Chamberlin D., Lorie R., and Price T. "Access path selection in a relational database management system ". In Proc. of the 1979 ACM SIGMOD Intl. Conf. on Management of Data, pages 23–34, June 1979.

[17] Urhan T., Franklin M., and Amsaleg L. "Cost Based Query Scrambling for Initial Delays ". In Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data, pages 130–141, June 1998.

[18] Zadorozhny V., Raschid L., Vidal M., Urhan T., and Bright L. "Efficient evaluation of queries in a mediator for web sources ". In Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data, pages 85–96, Aug. 2002.