The architectural design of distributed query optimization using UML tools

Esiefarienrhe Michael Bukohwo[1],Philemon Uten Emmoh[2] and Djibo Idrissa[3]

[1,2]Department of Mathematics/Statistics/Computer Science,
University of Agriculture, Makurdi. Benue State, Nigeria.

[3]Department of Computer Science,
Federal Polytechnic, Bauchi, Bauchi State, Nigeria

Abstract

Distributed query processing select data from database located at multiple sites in a network to achieve a single result. Query optimization then find the best execution plan for the given query which represents the execution strategy for the query. The efficiency of processing strategies for queries in a distributed database needs to be considered in system performance. This paper used caching techniques to improve query execution. Hill climbing and Ingres algorithm are used to obtain query optimization and query decomposition respectively. The paper only focused on the architectural design of the system using UML tools namely conceptual and Use case diagram for fast retrieval and high reuse of cached queries to obtain optimal result. This was achieved by using some parts of cached results helpful for retrieving other queries (wider Queries) and combining many cached queries while producing the result. These techniques will provide efficient performance in optimization of query processing in distributed databases environment.

***Keywords**: Query Optimization, caching, query execution, Hill Climbing Algorithm, Ingres Algorithm.*

1. Introduction

Given a query, there are many plans that a database management system (DBMS) can take to process the query and produce result. All these plans are equivalent in terms of their final output but vary in their cost. Which of these plans utilizes the least amount of time and cost is a issue to be determined by the optimizer. Optimization problem is a join ordering problem that has been proven to be NP-complete meaning that no polynomial time algorithm presently exist to find the optimal plan for large sizes of queries within a feasible time [1],[2]. An optimal plan is the execution sequence that produces the least cost in terms of time and memory utilization. The cost difference between two alternatives (plans) can be quite enormous. Distributed query processing is the process of selecting data from database located at multiple sites in a network while distributed processing performs computations on multiple CPUs to achieve a single result. Any statement in a data manipulation language that references tables at sites other than the site the application program is submitted for compilation is called distributed query. The goal of query optimization is to find an execution strategy for the query that is close to optimal. An execution strategy for a distributed query can be described with relational algebraic operations and communication primitives (send/receive operations) for transferring data between sites [3]. Query optimization is very difficult task in a distributed client/server environment as data location becomes a major factor [4]. In order to optimize queries accurately, sufficient information must be available to determine which data access techniques are most effective. Query processing is observed to be more difficult in distributed environment than in centralized environment due to: Large number of parameters that affect the performance of distributed queries, relations involved in distributed query may be fragmented and/or replicated and with many sites to access, query response time may invariably become high. The performance of distributed database system (DDBS) is dependent on the ability of the query optimization algorithm to derive efficient query processing strategies, while distributed database management system (DDBMS) algorithm attempts to reduce the quantity of data transferred. The act of minimizing the quantity of data transferred is a desirable optimization criterion, since more data transported across telecommunication networks requires more time and labor. Query optimization is a major aspect of query processing and it is the process of finding the best execution plan for a given query which represents the execution strategy for the query. This query execution plan (QEP) minimizes the objective cost function. The main objective of the query optimization is to decide the most efficient query execution plan which has minimum execution cost, among many possible plans by determining execution sequential order of relational operators. Query Optimization in distributed databases is a very difficult task due to number of factors such as data allocation, communication channel's speed, memory availability, database size , storage of intermediate result, pipelining and size of data transmission [5],[6]. In a distributed query processing, first of all, initial query is decomposed into fragment queries which operate on fragments rather than on global relation (data localization).Then in the second phase, joins/semi joins are applied to reduce the size of data that has to be sent across the network to different sites. In the final phase all the processed files are sent to assembly site for generation of final output. A Query Optimizer is used to generate QEP which represents an execution strategy of the query with minimum cost [6],[7].The performance of distributed query depends upon the query optimizers ability to obtain efficient strategies of processed query.

A distributed database is more popular because it improves system performance, reliability, availability and modularity in distributed system. The data distribution problem and query processing are the critical issues in distributed database. Database system performance effectiveness depends on join operator. The allocation of operations or sub queries involved in a particular query to the various sites of a network is one of the important components of distributed query processing and query optimization. The query is broken into various sub operations like selection, projections join and semi join and these operations performed at many different sites of the network in different sequences. Order sequence problem (OSP) and Operation allocation problem (OAP) are the two components of query optimization. OSP requires the optimal sequence of operations for example Join order sequence while OAP requires optimal placement of these operations to different site [8].

This paper therefore seeks to provide the design of a robust query optimization system using UML tools that can easily serve as input to developing the optimization software.

The rest of the work is organized as follows: Section two gives a brief background of the research related to this work while section three gives the methods and models used. Section four shows the implementation using the conceptual diagram and the Use case diagram while section five concludes the paper and gives an insight into future work in this research.

2. Background

A cost model was developed by [9] which consist of the following components:

1. Secondary storage cost - This is the cost of searching for reading and writing data blocks on secondary storage).
2. Memory storage cost: This is the cost pertaining to the number of memory buffers needed during query execution.
3. Computation cost - This the cost of performing in memory operations on the data buffers during query optimization.
4. Communication cost - This is the cost of transmitting the query and its results from the database site to the site or terminal where the query originated.

In distributed database systems there are three processes by which data is distributed among various sites, these are: fragmentation, allocation, and replication. Fragmentation process requires empirical knowledge of data access and query frequencies. Authors in [10] had proposed a horizontal fragmentation technique that is capable of taking proper fragmentation decision at the initial stage by using the knowledge gathered during requirement analysis phase without the help of empirical data about query execution. It allocates the fragments properly among the sites of DDBMS.

Authors in [11] worked on analysis of joins and semi joins in a distributed database query. Their focus was on computing and analyzing the performance of joins and semi joins in distributed database system. They explained various metrics that should be considered when analyzing performance of join and semi join in distributed database system namely Query Cost, Memory used, CPU Cost, Input Output Cost, Sort Operations, Data Transmission, Total Time and Response Time.

From their study, they concluded that data transmission in a distributed query using semi-join is always lesser than the data transmitted in distributed query using joins operation however data accessed using semi join may be larger than join.

Gregory in [7] developed a Genetic algorithm (GA) for optimizing queries and its performance and compared it with other alternative random optimization techniques like random search, multistart etc. All their tables were fully reduced in a tree query by optimizing a semijoin using this GA. For this problem, evaluation of the fitness function is a costly task. The GA proposed for this uses a tree-structured data model with customized crossover and mutation operators that avoid the need for full reevaluation of the fitness function for new solutions. To meet the real time nature of query optimization task, the GA prosed here uses a local search phase to provide the required real-time performance. Their GA is robust, performs well at the beginning of a search, overcomes the problem of premature convergence and makes persistent progress to better solutions.

3. Methods

This work combines the Ingres and Hill climbing algorithms to develop a Distributed Query Decomposition and Optimization algorithm. The Ingres algorithm will be used for distributed query decomposition while the Hill Climbing algorithm will be used for query optimization.

4. Implementation

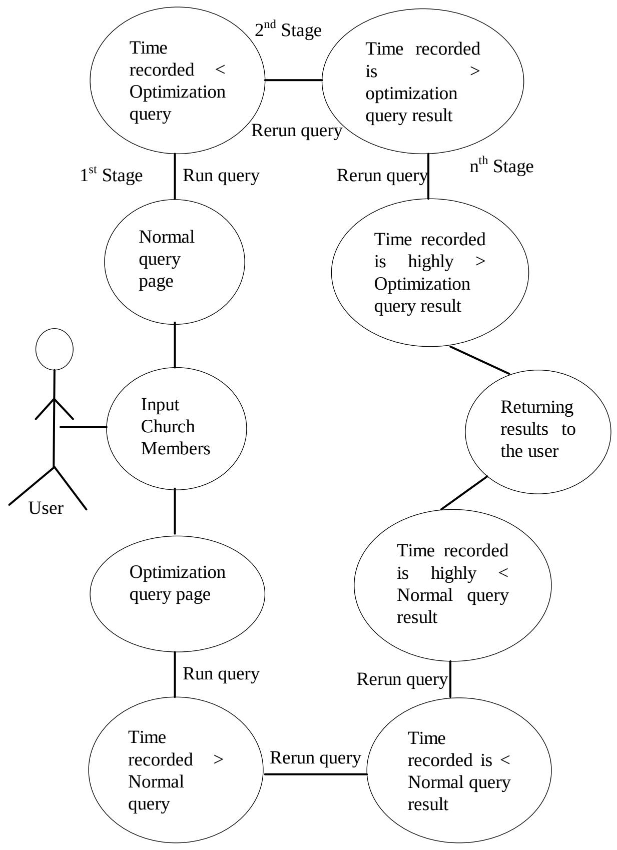
Time recorded < Optimization query
2nd Stage
Time recorded is > optimization query result
Rerun query
1st Stage
Run query
Rerun query
nth Stage
Normal query page
Time recorded is highly > Optimization query result
Input Church Members
Returning results to the user
User
Optimization query page
Time recorded is highly < Normal query result
Run query
Rerun query
Time recorded > Normal query
Rerun query
Time recorded is < Normal query result

Fig. 1 Use case for system query process.

In Figure 1, the user is meant to input the details of the church members at the distributed church database system. The church application has will have two pages; the normal query result page and optimization query result page where the query will be carried out. At the first stage, the same number of members is entered in both the normal result query page and the optimization result query page i.e. when 35 members were entered in both query result pages, a query is run in both the normal and the optimization result page while the time taken by each page will be recorded. At this stage, the time recorded in normal result query page at when members are 35 is < (less than) the time recorded in optimization query result page at 35 members. At the second stage, we rerun the query, the time recorded in normal result page still at when members are 35 is > (greater than) the optimization query result pages. We continued to rerun the query until infinity (nth stage), we will therefore discovered that the time recorded in the normal result is highly > (greater than) the time recorded in the optimization query result. The intermediate result proved that the optimization result is faster and better than the normal result and the result will be returned to the user issuing the query command as shown in figure 3.2 above. If more members were entered, and the same process is taken, the normal query result page will always be < (less than) the optimization query result page. As the query is rerun in the second stage, the reverse is the case i.e. the time taken to run the normal query result page will be higher than the time taken to run the optimization query result page. Here, running the optimization query results gives us the best and faster results than the normal query results.

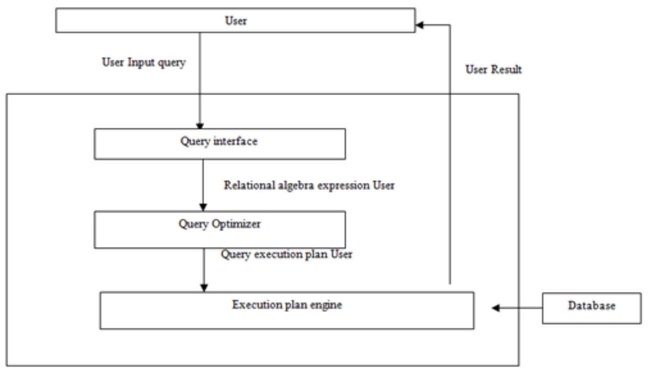

Fig. 2 Conceptual model of the system.

Figure 2 show the user input the query to the query interface which handles both the normal query and the optimization query of the designed system. Relational algebra expression here expresses a language that is being used to explain basic relational operations and its principles. The relations are stored in a database and the results from a database can be obtained by using database queries. The query optimizer then generates one or more query plans for each query, each of which may be a mechanism used to run a query. The query execution plan is being considered here as an ordered set of steps used to access data in the distributed church database. When being considered, the execution plan query engine will do a scan over the primary key index on the distributed church database and a matching seek through the primary key index on the distributed church database to find a matching rows whenever a query command is issued. And the final result of the query is given to the user who input the query and the optimization result is produced as fast as possible as the result while the normal result is ignored. After the query interface, the query optimizer analysis both the normal result and the optimization result while taking the optimization result and passed it to the next stage of the query to give a better result to the user.

5. Conclusions

Most database application systems grow slower in processing over time, this is due to several reasons which includes; macro query composition and data redundancy which is a condition created within a database or data storage technology in which the same piece of data is held in two separate places. This can mean two different fields within. By decomposing and optimizing distributed queries these problems are a single database, or two different spots in multiple software environments or platforms eliminated thereby achieving a better performance.
The optimization results in the distributed church database system are better than the normal results. This is because during the process of the query decomposition and optimization algorithm, the time recorded when a record is queried in optimization result page is always less than the time recorded during the normal result page. Here we therefore believed that the optimization results are better and faster than the normal results. Our results here clearly have shown the effect of optimization on CPU processing time across distributed database systems.

3. Future Work

The design done will be implemented using the ingress and hill climbing algorithm in future. This two algorithms will not only compressed the data for efficiency but will improve multi-source data query performance through optimization technique.

References

[1] W. Scheufele, G. Moerkotte, and S. A. Constructing optimal bushy processing trees for join queries is np-hard (extended abstract). Technical report, Universitat, Mannheim, 1996.
[2] R. Taylor. Query Optimization Database Systems. Master Thesis, Computer Science Department, University of Oxford, 2010. Available at: http://www.cs.ox.ac.uk/people/dan.olteanu/theses/Robert.Taylor.pdf
[3] M. T. Oszu and P. Valduriez. Distributed and Parallel Database Systems in Alaa Aljanaby, Emad Abuelrub and Mohammed Odeh, IAJIT, Vol. 2 No. 1. Pp. 48-7.Jan.2005.
[4] B. M. Alom, F. Henskens and M. Hannaford, Query Processing and Optimization in Distributed Database Systems, International Journal of Computer Science and Network Security, 9(9), 143- 152. (2009).
[5] M.O. Tamer and P. Valduriez, Principles of Distributed Database Systems, Third Edition,Springer,2011.
[6]M. P. Tiwari and V. S. Chande, Query Optimization Strategies in Distributed Databases, International Journal of Advances in Engineering Sciences Vol.3 (3), July, 2013.
[7] M. Gregory, "Genetic Algorithm Optimization of Distributed Database Queries", IEEE, 1998.
[8] A. Aljanaby, E. Abuelrub and O. Mohammed, A Survey of Distributed Query Optimization, The International Arab Journal of Information Technology, Vol. 2, No. 1, January 2005.
[9] R. Elmasri and S.B. Navathe. Fundamentals of Database Systems, Reading, MA, Addison-Wesley, 2000
[10] S. I. Khan and A. S. M. LatifulHoque, A New Technique for Database Fragmentation in Distributed Systems, IJCA Vol. 5. (2010).
[11] G. Singh and V. Rajinder, Analysis of Joins and Semi Joins in a Distributed Database Queries, International Journal of Computer Applications 49(16):14-18, July 2012.