International Journal of Enterprise Computing and Business Systems
ISSN (Online) : 2230-8849
Volume 4 Issue 2 July 2014
International Manuscript ID : ISSN22308849-V4I2M15-072014

AN EFFECTIVE IMPLEMENTAION OF QUERY AND CACHE BASED OPTIMIZATION IN DISTRIBUTED SYSTEMS

Hitu Kalra
M.Tech. Research Scholar
Shree Siddhivinayak Group of Institutions
Shahpur-Bilaspur, Distt. Yamuna Nagar, Haryana, India

Prof. Ajay Kumar
Shree Siddhivinayak Group of Institutions
Shahpur-Bilaspur, Distt. Yamuna Nagar, Haryana, India

ABSTRACT

A distributed system is a piece of software that ensures that a collection of independent computers that are interconnected by a computer network, appears to its users as a single coherent system and that cooperate in performing certain assigned tasks. The optimization of queries in distributed systems is a complex activity that depends on many factors. In a certain percent it is performed by the DBMS, but there are situations when the user applications must contain algorithms for the query optimization. I studied many related work and found out that very few work addressed the problem of considering run-time conditions in query optimization. By analyzing both theoretically and experimentally, I have presented the need to take run-time conditions, including CPU utilities in the data sources and network environment, into account in optimization process. This research work studies two existing approaches, namely Sequential Processing and Parallel Processing. However, after analyzing their pros and cons, we found that both of them are not sufficient for optimization of distributed queries as they do not consider run time conditions in the optimization process. In this work, I have presented a framework of proposed system. The system can perform the join query effectively through the interaction of the system components: join query recognizer, optimizer, and executor and metadata database. I also have proposed a join query optimization algorithm which predicts the best execution plan for a join query which retrieves data from two remote computers. My algorithm estimates and compares the total response time of all possible sequential plans and parallel plans. The total response time of each execution plan is impacted by the size of transmission data and speed of transmission. The size of transmission is determined by the type of join query. In the data

transmission test, I discovered that the total response time of data transmission in my system comprises of the time for querying, transferring and inserting. The querying time is short compared to the time for transferring and inserting, but it is important to ensure that the size of querying data does not exceed the system memory limitation. The time for transferring data is significant when the transmission speed is slow and the size of transmission data is large. Moreover, the test results of data transmission shows that the transmission speed is linear and has a strong correlation with respect to the amount of data transmitted.

In the test of the join query performance, I proved that the neither parallel query processing nor the sequential query processing are the best plan in all cases. The parallel query processing is better than sequential query processing when the size of the join query result is equal to or larger than the maximum of the sizes of two source tables. The sequential query processing is faster than parallel query processing when size of the join query result is relatively small compared to the source tables. In addition, the transmission speed is a factor which the system must consider to predict the best plan for a join query.

Finally, I have implemented the prototype system with the proposed architecture and optimization algorithm. The experimental results showed the capabilities and efficiency of join query optimization algorithm and gave the target environment where the algorithm performs better than other related approaches and predict the best plan for a join query.

Keywords – Distributed Databases, Query Optimization, Performance of Distributed Systems

Proposed join query optimization system comprises of seven basic components: query recognizer, query optimizer, query executor, speed calculator, and metadata database.

The proposed framework has following modules:

User

A user is an agent, either a human agent (end-user) or software agent, who uses a computer or network service. A user often has a user account and is identified by a username (also user name). Other terms for username include login name, screen name, users are also widely characterized as the class of people that use a system without complete technical expertise required to understand the system fully. In projects in which the actor of the system is another system or a software agent, it is quite possible that there is no end-user for the system. In this case, the end-users for the system would be indirect end-users

Global Query Interface

Global Query Interface is used to obtain a pointer to another interface, given a Graphical User Interface Database (GUID) that uniquely identifies that interface.

Query Recognizer

Query recognizer checks if the user input query is a valid join query. This recognizer firstly detects if the input query has any syntax errors and if the columns, tables and database in user query exist in the distributed database. If the recognizer detects any error in user input query, recognizer will terminate the query execution and returns an error message. The error detection prevents the network and computation resources from being executing an invalid query. Query recognizer also checks if the user query is a valid join query or not. If the query is not a valid join query, this query will not be executed using the join query optimization algorithm because it is more efficient to execute non-join query directly.

Furthermore, the join query recognizer prepares the information related to a join query after it detects input query is join query. First, it determines which type of join query the input query is. The type of join query is used to estimate the quantity of the result. Also the recognizer divides the query into sub-queries and sends it to query optimizer. Finally, the recognizer retrieves the IP addresses of the machines, the database names and the types of database platforms where the source tables reside. The IP addresses and database names help the system locate the web services and data.

Query Optimizer

Local optimizer receives the sub-query from global optimizer, and executes the sub query in its local database management system to determine the number of rows, number of columns, and the total quantity of data. Local optimizer also performs the query which requires the distribution information of sub-query and sends the distribution information to global optimizer. The global Optimizer controls the entire of join query optimization and chooses the best execution plan.[19] The global optimizer interacts with the local optimizers which reside on the remote sites to retrieve the information from the two sub-query result. Then, the global optimizer estimates the size of the join query result based on the type of join query and the distribution information of the sub-query, and calculates the estimate of the total response time for the three possible execution plans based upon the network speed, process speed and the time for initialization from metadata database. Finally, the global optimizer selects the plan that has the minimum response time as the best execution plan.

Query Executor

Query executor performs the join query using the execution plan chosen by the join query optimizer and records the time spent on data transmission and local query execution. There are three sub-components in join query executor: the data sender, the data receiver and the global query executor.

The data sender and data receiver lies on the remote sites. The data sender works on the participating remote sites and data receiver works on the joining site. The data sender executes the sub-query and save the information includes the number of rows, the number of columns, the column names, and the column data types and transfers the resulting data to the data receiver.

The data receiver receives the data and inserts it into the database. The data transmission between the data sender and the data receiver is divided into several smaller transmissions when the amount of resulting data is too large. If the size of object that holds the data exceeds the capacity of the machine's memory, the system will respond very slowly or crash. Thus, the system must divide the query into a set of sub-queries such that the quantity of data from the sub-query fits to memory size. Furthermore, it is more efficient to run a set of small queries than one large query. Even if the large size of the query result does not exceed the memory capacity, it still consumes a significant amount of memory. If the machine's free memory gets too low, the system will perform very slowly.

The global query executor interacts with the data sender and data receiver to perform the execution plan. The global query executor calls the data senders on the remote sites to prepare the result, and then instructs the data receiver on the joining site to retrieve the data and store it. After the data transmission completes, the global query executor interacts with the data sender of the site where all participating tables now reside. The data sender executes the join query and returns the result to global query executor. In addition, global query executor records the response time for data transmission join query execution, and the quantity of data. For each transmission, the system records the source site, destination site, quantity of transmission data, and response time. The data are saved into the metadata and used to update the transmission speed and processing speed.

Speed Calculator

Speed calculator calculates the data transmission speed between sites, the processing speed of each machine, and the initial time for data transmission and query execution. The speed calculator retrieves the history execution data from metadata database, applies the *Least Squares Regression Algorithm* to fit the data, and then inserts new records or updates the existing records in the metadata database.

Metadata Database

A metadata database is a centralized database that stores the metadata of the entire distributed database.[20] The metadata database stores the information of all database systems within the distributed database, such as the IP address of each database, the database platforms, the database names, the database file directories, the table names, the Column names, the data types, and so on. The database metadata helps the system to locate data, access data, and test for errors. It enhances the efficiency and effectiveness of database collaboration. Moreover, the metadata database records the transmission speed, processing speed and history performance data. The global optimizer on each site uses this data to estimate the response time and choose an execution plan.

Proposed Work

My proposed work goes through two methods - Sequential method and Parallel method.

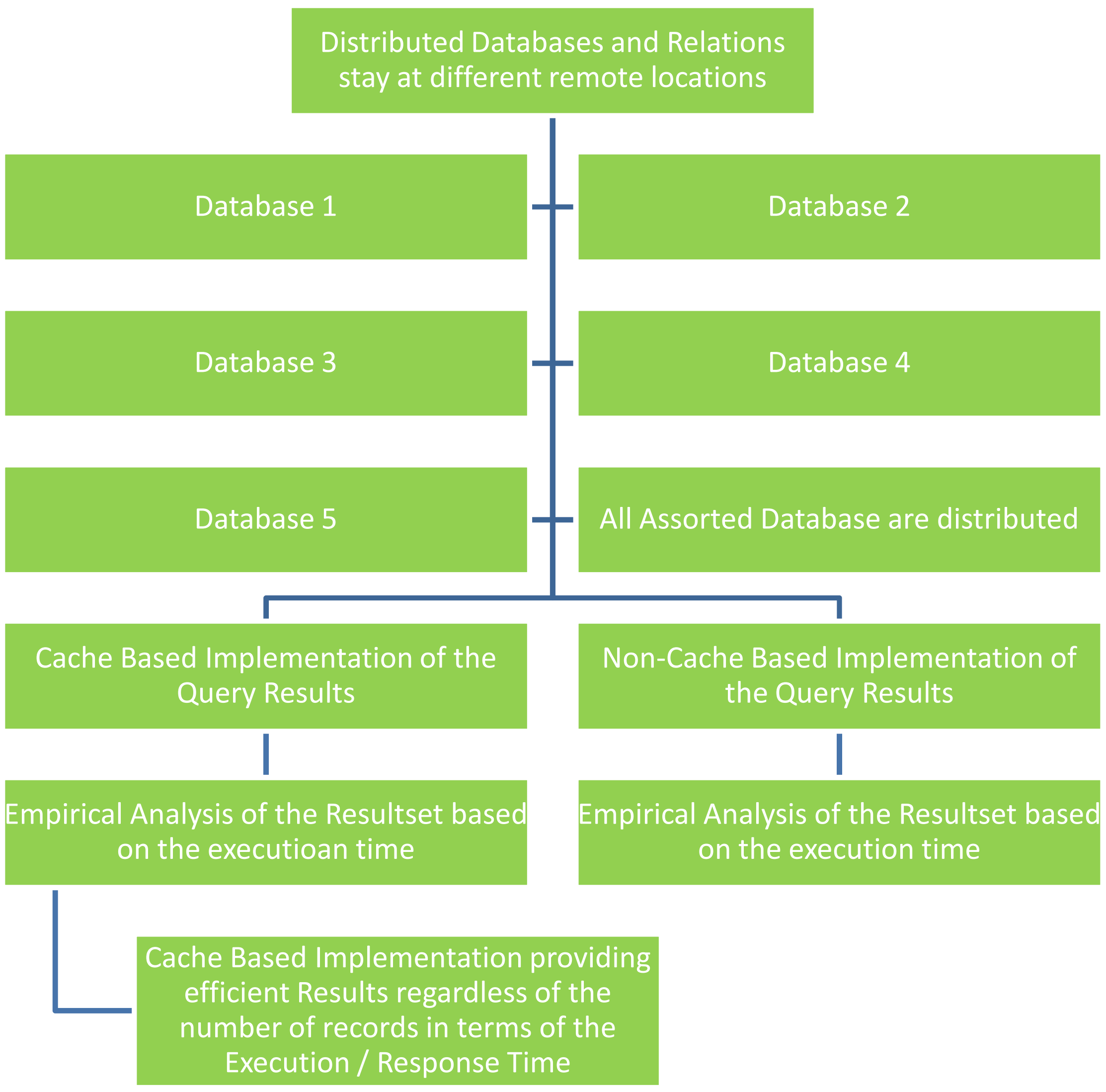
First method implement this join query by sending all the join participating tables to the each site where data is distributed and perform the join at that site and selects the site which perform the join query at minimum response time. Time for this method will equal to the addition of time to fetch the data from participating sites, time to transfer the data from participating site to the joining site, time to insert the data into joining site and join query processing time.

Second method will show the results by performing the join query on the client site. The join query in this method will directly take the data from server sides. Time for this method will depend upon the time to fetch the data from server sides, transferring the data from sever sides to client side, time to insert the data into client database and join query execution time.

Proposed algorithm compares both the methods based upon their performance and execute the best one. Therefore, the join query will be optimized in distributed database.

The Proposed Research Work is based on the following methods -

- Sequential method and Parallel method execution of the Queries using Join Operations from multiple fragmented databases as well as relations
- Cache Based Optimization of the Query Results that is providing better and efficient results as compared to non-cache implementation
- The pragmatic investigation of cache based implementation to fetch the tuples from assorted databases distributed at multiple locations.
- Graphical analysis of the query execution time so that the investigation can be done with prior implementation.
- The web based scripting as well as live databases for the execution and testing based on the execution time or response time by which the proposed work can be analyzed as well as justified.
- The proposed implementation is justified with an empirical algorithmic approach that is deployed on a web based platform.
- The web based platform as well as query optimization engine shall keep track of each query and administration panel for analysis.


Distributed Databases and Relations stay at different remote locations
Database 1
Database 2
Database 3
Database 4
Database 5
All Assorted Database are distributed
Cache Based Implementation of the Query Results
Non-Cache Based Implementation of the Query Results
Empirical Analysis of the Resultset based on the executioan time
Empirical Analysis of the Resultset based on the execution time
Cache Based Implementation providing efficient Results regardless of the number of records in terms of the Execution / Response Time

The experiment and simulation is performed on the distributed databases.

A SQL join statement joins together records from two or more tables in a database. It makes a set that could be spared as a table or utilized as it seems to be. A Join is a methods for joining fields from two tables by utilizing values normal to each. ANSI-standard SQL indicates five sorts of Join: Inner, Left Outer, Right Outer, Full Outer and Cross. As an exceptional case, a table (base table, see, or joined table) can Join to itself in a self-join.

A programmer composes a Join proclamation to distinguish the records for joining. Assuming that the assessed predicate is accurate, the joined together record is then transformed in the normal configuration, a record set or an impermanent table.

Social databases are frequently standardized to dispense with duplication of data when articles might have one-to-numerous relationships. Case in point, a Department may be connected with numerous diverse Employees. Joining two tables viably makes an alternate table which consolidates data from both tables. This is at some upkeep regarding the time it takes to process the join. While it is likewise conceivable to essentially administer a denormalized table if speed is imperative, copy data might consume additional room, and include the overhead and intricacy of upholding information uprightness if information which is copied later changes.

An 'inward join' is a generally utilized join operation utilized within provisions. It can just be securely utilized as a part of a database that implements referential uprightness or where the join fields are ensured not to be Null. Numerous transaction transforming social databases depend on Atomicity, Consistency, Isolation, Durability (Acid) benchmarks to guarantee information trustworthiness, making inward joins a more dependable decision. Numerous reporting social database and information warehouses use high volume Extract Transform, Load (Etl) group overhauls which make referential trustworthiness troublesome or difficult to implement, bringing about possibly Null join fields that a Sql question creator can't adjust and which cause internal joins to overlook information. The decision to utilize an internal join relies on upon the database outline and information attributes. A left external join can generally be substituted for an internal join when the join field in one table may hold Null qualities.

Internal join makes another come about table by consolidating segment values of two tables (A and B) based upon the join-predicate. The inquiry contrasts every column of A and every line of B to uncover all matches of columns which fulfill the join-predicate. The point when the join-predicate is fulfilled, segment values for each one matched match of columns of An and B are joined into a consequence line. The consequence of the join could be characterized as the result of first taking the Cartesian item (or Cross join) of all records in the tables (joining each record in table A with each record in table B) and after that giving back all records which fulfill the join predicate. Genuine Sql executions ordinarily utilize different methodologies, for example, hash joins or sort-consolidate joins, since registering the Cartesian item is extremely wasteful..

IMPLEMENTATION / SIMULATION SCENARIO - 1

Attempt ID	Product ID	Classical Approach	Proposed Approach
1	P001	1.00519800186	0.200445890427
2	P002	1.00735902786	0.37758398056
3	P002	1.00400495529	0.0761110782623

4	P003	1.0077021122	0.00262784957886
5	P003	1.47337388992	0.0465440750122
6	P003	1.01530909538	0.0499310493469
7	P001	1.10068583488	0.0308630466461
8	p004	1.01806807518	0.0550210475922
9	p004	1.08409404755	0.0505809783936

Empirical Comparative Analysis of the Approaches

INTERPRETATION OF THE TABLE

In this simulation scenario, we have taken the set of queries executed on the Live Server that is having number of distributed databases at multiple locations. The implementation has been performed to test and analyse the results from the Live Server Based Deployment. The results very clearly show that the proposed algorithmic system or approach is providing effective as well as efficient results as compared to the classical approach.

The Live web based implementation that is based on MySQL Database Server having assorted databases has been tested and simulated with PHP Scripts and Graphical Implementation. It is found without any specific qualm that the proposed system approach of query optimization is efficient to justify and prove the research work.

In the proposed as well as implemented research work, it is apparent that the query optimization time of the proposed algorithmic approach is rapid and acceptable.

In the upcoming graphical representation and analysis, we will explain the results so that the proposed approach can be proved better than the classical approach.

We have taken the Live Web Server response time and generated the different types of graphs and all graphs are proving that we are getting better results in the proposed approach.

The upcoming graphical demonstrations shall prove the fact that the proposed approach will definitely be better than the traditional database queries. As we have implemented the cache based engine for query optimization, the algorithmic approach is proved to be efficient to get the resultset from the databases distributed at remote locations. The remotely as well as distributed databases are making use of the foreign key and getting the results in very less time.

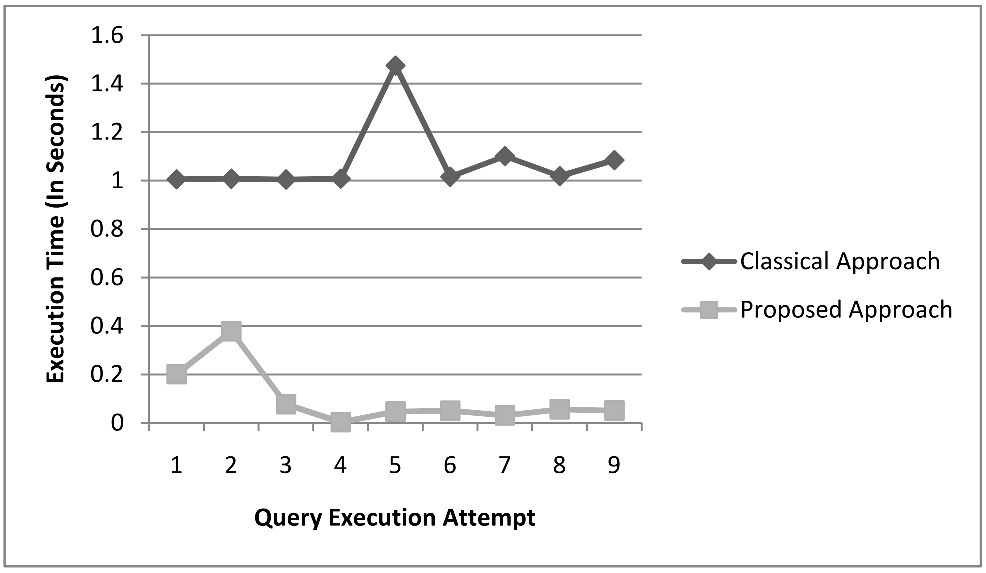


Figure 1 - Execution Time of Classical and Proposed Approach

INTERPRETATION OF THE GRAPH

The above drawn graph has been generated from the data fetched from the database server and our implementation performed on the data set.

It can be seen from the graph that the query execution time in the proposed approach is far lesser than that classical approach. The graphical representation is quite enough to prove the upcoming fact and conclusion that the join query optimization along with the cache based implementation of the engine is better as well as enhanced from the traditional method of records fetching from assorted distributed databases.

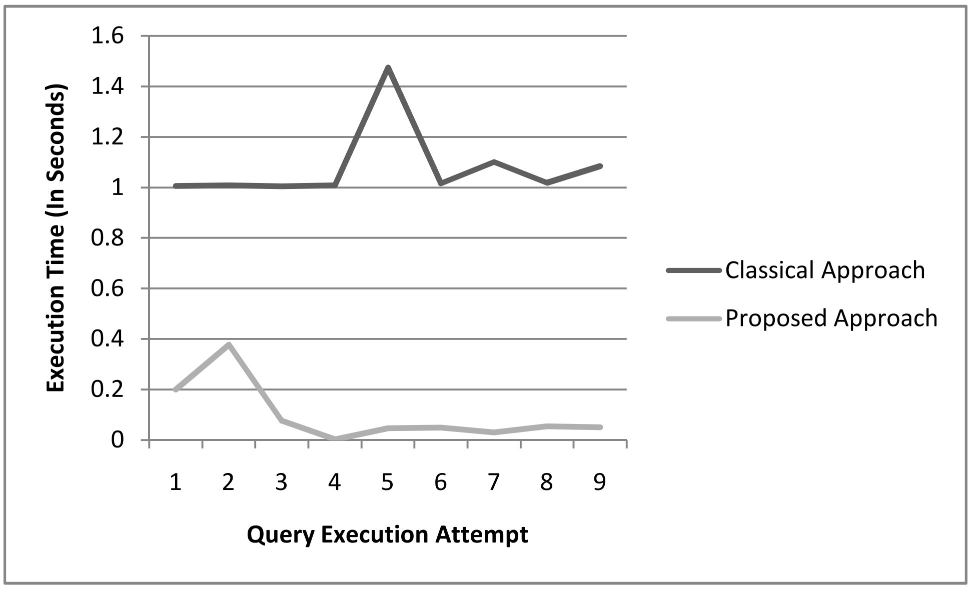


Figure 2 - Execution Time of Classical and Proposed Approach

INTERPRETATION OF THE GRAPH

The graph above mentioned shows that that demarcation line of the classical versus proposed approach is having huge difference.

The graphical representation and analysis of the figure demonstrates the fact as well as conclusion that the cache based implemented is better as compared to the non-caching in the databases from distributed locations.

These distributed locations can be remote and obviously as per the implementation will not affect the results. The results from the proposed approach will provide good results as we can see in the graph.

CONCLUSION

This work implements the prototype system with the proposed architecture and optimization algorithm. The experimental results showed the capabilities and efficiency of join query optimization algorithm and gave the target environment where the algorithm performs better than other related approaches and predict the best plan for a join query. For future work, I plan to extend my study in the following directions:

- The metaheuristic based implementation can be performed that includes ant colony optimization, honey bee algorithm, simulated annealing and many other others. Such algorithmic approach should provide better results when we move towards metaheuristics.
- Join query optimization algorithm assumes that all the relations referenced by a query are not fragmented but distributed in different sites. So a natural extension is to enable the algorithm to process a query where all relations referenced by a query are

fragmented and replicated across multiple sites.

- This dissertation mainly discusses distributed join operation. Certainly we join firstly and then perform other operations on the joined result, but there can be other brilliant way to schedule all operations efficiently in distributed environments.
- We may also plan to extend this algorithm for the processing of heterogeneous database systems.

➢ Another possible extension is to research the methods of inserting data. Inserting data into the database is a significant component of the time to perform the distributed join.

REFERENCES

[1] Swati Gupta, Kuntal Saroba, Bhawna, “Fundamental Research of Distributed Database”, International Journal of Computer Science and Management Studies, pp. 138-146, 2011

[2] Fan Yuanyuan, Mi Xifeng, “Distributed Database System Query Optimization Algorithm Research”, IEEE international conference on Computer Science and Information Technology (ICECT), pp.145-149, 2011

[3] Neera Batra, A. K. Kapil, “Three Tier Cache Based Query Optimization Model in Distributed Database”, IJEST, vol. 2, 2010, pp. 3206-3212

[4] Reza Ghaemi, Amin Milani Fard, Hamid Tabatabee, Mahdi Sadaghizadeh, “Evolutionary Query optimization for Heterogeneous Distributed Database Systems” ,World Academy of Science, Engineering and Technology, pp. 43-49, 2008

[5] S. Upadhyaya and S. Lata, "Task Allocation in Distributed Computing VS Distributed Database Systems: A Comparative Study", IJCNS (International Journal of Computer Science and Network Security),vol. 8:3, pp. 338-346, 2008.

[6] Ţambulea L., Horvat-Petrescu M., “Redistributing Fragments into a Distributed Database, International Journal of Computers Communications & Control”, ISSN 1841-9836, 3(4):384-394, 2008.

[7] Stocker, Kossman, Braumandl, Kemper, “Integrating Semi Join Reducers into state of the art query processors”, ICDE, pp. 143-156 , 2001

[8] J. Callan, “Distributed Information Retrieval ” , W. B. Croft, Ed. Kluwer Academic Publishers, pp. 127-150, 2000

[9] D. Kossman, “The state of the art in distributed query processing” , ACM Computing Surveys, pp. 422-469, 1998

[10] P. Griffiths, Selinger, M. M. Astrahan, D. D. Chamberlin, R.A. Lorie, T. G. Price, “Access path selection in a rational database management system”, Morgan Kauffman series in Data Management Systems, pp. 141-152, 1998

[11] Yannis. E. Loannidis and Youngkyung Cha Kang, “Randomized Algorithms for optimizing large Join Queries”, ACM Computing Surveys, pp. 47-53, 1990

[12] Chin-Wan Chung, "An Optimization of Queries in Distributed Database Systems", Journal of Parallel and Distributed Computing 3, pp. 137-157, 1986

[13] Philip A Bernstein and Nathan Goodman, Engene Wong, Christopher L. Reeve and James B. Rothnie, "Query Processing in a System for Distributed Databases(SDD – 1), ACM Transactions on Database Systems, vol. 6, no. 4, 1981, pp. 602-625

[14] Li, Vector O. K. , "Query Processing in distributed databases" , MIT. Lab. For Information and Decision Systems, pp. 1107, 1981

[15] Huang, Kuan – Tsae, Davenport, Wilbur B., "Query Processing in Distributed Heterogeneous Systems", MIT Laboratory for information and Decision Systems, pp. 45-49 , 1981

[16] B.M. Monjurul Alom, Frans Henskens and Michael Hannaford, "Query Processing and Optimization in Distributed Database Systems", IJCSNS International Journal of Computer Science and Network Security, vol.9 no.9, 2009, pp. 143-152

[17] Syam Menon, "Allocating fragments in distributed Database", IEEE Transactions on Parallel and Distributed Systems, pp. 577-585, 2005

[18] D. Kossmann, K. Stocker, "Iterative Dynamic Programming: A New Class of Query optimization Algorithms", ACM Computing Surveys, pp. 422-469, 2000

[19] Lee Chiang, Chi-sheng shin, Yaw-huei chen, "Optimizing large join queries using a graph based approach", IEEE Transactions on Knowledge and Data Engineering, pp. 441-450, 2006

[20] Tsai, P.S.M, Chen A.L.P, "Optimizing queries with foreign function in a distributed environment", IEEE Transactions on Knowledge and Data Engineering, pp.809-824, 2002

[21] Sukheja Deepak Singh Umesh Kumar (July 2011), "A Novel Approach of Query Optimization for Distributed Database Systems", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1.

[22] Pawandeep Kaur, Jaspreet Kaur Sahiwal, "Join Query Optimization in Distributed Database", International Journal of Scientific and Research Publications, Vol. 3, Issue 5, May 2013.

[23] Nicoleta Iacob, "Distributed Query Optimization", PhD Student, University of Piteşti, Issue 4/2010.

[24] Jyoti Mor, Indu Kashyap, R. K. Rathy, "Analysis of Query Optimization Techniques in Databases", International Journal of Computer Applications (0975-888), Vol. 47 - No. 15, June 2012.