Query Optimization- Challenges and Factors Affecting the Overall Cost in Distributed Databases

Authors
Anjali Soni¹, Sangeeta Vaibhav Meena²
¹,²S.S.Jain Subodh P.G.College, Jaipur, Rajasthan, India
Email: anjalisoni81@gmail.com¹, angeeta.yad@gmail.com²

ABSTRACT
Distributed databases provide more flexibility, reliability and ease as compared to the centralized databases. It is the way to maintain data on multiple sites on which data can be stored and retrieved. Data on these sites may be replicated or fragmented. Performance of the distributed query highly depends on the query optimizer. Query optimization is the process of producing a query execution plan which represents an execution strategy for the query. It works to find out the results of a given query in minimum time. As the data is dispersed on various sites, query optimization in distributed databases is complex which involves the transmission of data among different sites. The Select –Project –Join (SPJ) query plays an important role in deciding the best optimization strategy. Current optimizers works on queries where numbers of joins in a query are less than 15 but for more number of joins, new algorithms must be devised to find out the result in minimum time. This paper presents various problems involved in query optimization in large scale distributed databases where number of relations and the number of joins involved in a query increases.

Keyword- Distributed Database, Optimization, Replicated, Fragmented, SPJ Queries, minimum time

INTRODUCTION
The term database means collection of relevant data stored in files or tables. Database Management System (DBMS) is the system that is used to manage these collections of relevant files. There are two ways to manage these data. Centralized database are those that holds all data on a single computer, the database resides physically at one location. In this approach the data is kept on a central repository hence it provides an ease to retrieve data from multiple tables. These database queries are easily convertible into set of relational algebra operation. However, a distributed database is that database in which portions of the database are stored on multiple computers within a network [¹]. The data is distributed and spread over different geographical area but the database is still centrally administered as a corporate resource and also providing local flexibility and customization at each and every node. A user or program at location L¹can be able to access and also updates data at location L²; this is possible only because of the network which allows the users to share their data. The nodes of a distributed database system may be spread over a large geographical area which may include a city or a country, or over a small area of a building. A main objective of distributed databases is easily accessibility of data for users residing at different locations. To attain this, the user or program using data for querying or updating need not to know the location of the data, this is called location transparency. As data is spread over different sites/nodes, it requires more efforts to execute a query. A query in distributed system is defined as accessing, manipulating, and processing of huge numbers of relations concurrently through applying various join strategies. In distributed system query processing involves transmission of data from remote places. During execution of
distributed database queries, query optimization is utmost significance and expensive stage whose aim is to give quick, correct and consistent results to the user through minimum utilization of the system resources.

**Distributed Database Management System**

Distributed Database System is integration of database system and networking technologies. A distributed database is a collection of multiple and logically interrelated databases distributed over a computer network. It adopts client-server architecture which consists of client and server processes. There exists a clean separation of duties between the client processes and the server processes in client-server architecture. Storage of data is at server sites that run server processes that provide access to their local data to client processes whereas client processes are run at client sites that provide an interface between users and the data residing in the system. In a collaborating server system, each process or node in the system can act as both a server and a client just like a peer-to-peer network. Here data is stored at all the nodes and it may be possible for a node to receive a transaction request that involves data which is not stored locally to it. If a node accepts a transaction request which require access to data stored at other sites then the node will cooperate with the appropriate node until the transaction is completed.

A distributed database management system is the software system that permits the management of the distributed databases and makes the distribution transparent to the users. Distributed systems are mainly advantageous for large organizations that have offices and their workforce distributed across the world. As it offers access to shared data from multiple locations, it also provides benefits including improving the availability of data. With distributing data we can avoid the single point of failure situation where data belong to at a single site and that site fails. By replicating data on numerous nodes there would be more chance of having a substitute copy of the required relation always available for us so that we can deal in case of site failure. This would permit the service requesting the relation to continue its work without being disturbed. Replication of data is not only improving availability of data but also improves load distribution. By creating replicas of relations that are in great demand we can reduced the load on owning sites, which may have an impact on decreasing the number of site failures. By duplicating relations to sites where they are most commonly used network accesses can be reduced, which can improve the performance for users at those sites.

**QUERY OPTIMIZATION**

When a user issues a query in a centralized DBMS the optimizer produces a detailed Query Execution Plan (QEP) that can be run at minimum cost and the output is passed to the executer. The executer can then follow this plan for the evaluation of the query and return the results to the user.

SELECT price FROM Cust AS C, Item AS I, Ord AS O WHERE C.ckey1 = O.okey2 AND O.okey2 = I.okey3 AND C.name = 'Joe'

In this query the user is specifying that they wish to receive the price of all items ordered by the customer named 'Joe'. The relations involved in the query are named in the FROM clause as Cust, Item and Ord that store information about customers, items and orders respectively. The output attributes are specified in the SELECT clause and the join conditions are specified in the WHERE clause. Join conditions are predicates that take tuples t from the cartesian product of two input relations and maps them into the boolean set S = {true; false}, where the elements of S indicate whether or not t will be present in the output. For example, if we have a join R1 a=b R2, only tuples from R1 X R2 where R1.a = R2.b will be present in the output. Here, R1.a denotes the attribute named a from relation R1. Here only equi-join operation is considered, that is joining conditions that contain equality between attributes.

An optimizer can combine selectivity information
from the system catalog with an SQL query in order to produce a join graph representation of the given query. The selectivity $\sigma$ of a join $R_1 \bowtie a = b R_2$ is defined as the ratio of the number of result tuples $nt_{1,2}$ to the cardinality of the cartesian product of $R_1$ and $R_2$ i.e.

$$\sigma(R_1 \bowtie a = b R_2) = \frac{nt_{1,2}}{|R_1 \times R_2|}$$

A join graph consists of a number of vertices, representing relations, and edges, representing join conditions. An edge between vertices $v_1$ (representing relation $R_1$) and $v_2$ (representing $R_2$) indicates that a join condition exists between $R_1$ and $R_2$. Edges are annotated with the join condition and associated selectivity. Four common structures of join graphs are chain, cycle, star and clique. For each user query there can exist a large number of equivalent plans that produce the same result but which have different execution costs. The cost of a plan is estimated using a cost model that is based upon examining the number of times the disk is accessed during execution of the plan. The problem that the optimizer faces is that equivalent plans can have costs that vary by orders of magnitude so choosing the wrong plan can result in a user having to wait days for a result instead of seconds \cite{6}.

Ideally the query optimizer would use an enumeration algorithm to enumerate over the entire search space and find the plan with the lowest cost. However, in practice the expectation of an effective optimizer is often just to avoid very poor plans.

**QUERY OPTIMIZATION CHALLENGES**

As the data is distributed at different nodes it is quite challenging task to compute efficient query plan in distributed environment. There are many problems encountered when designing an optimized for distributed database. The first and foremost problem relates to the size of the search space. In a distributed database the existence of relation copies at several sites and the number of execution sites contribute to making the search space even larger. In a distributed setting the search space will be considerably larger as we have the opportunity to execute joins at any site and use any available copies of relations residing at different sites. The second problem emphasizes on the necessity of identifying inter-operator parallelism opportunities within a given plan in order to be able to determine the minimum possible total execution time of the plan accurately.

The problem of finding the optimal query execution plan in a centralized database system is an NP hard problem. In a distributed setting the search space will be considerably larger as we have the opportunity to execute joins at any site and use any available copies of relations residing at different sites. The worse case consists of all system relations being present at all sites. We proceed by considering three levels on annotations on plans. The first level consists of plans with join and relation nodes without any site annotations. The next level of plans contains site annotations at leaf nodes only and the final level consists of complete plans with all node site annotations.

In a centralized DBMS the cost of a plan is determined by aggregating individual operator I/O estimations. Using the number of I/Os as a cost metric is a good indication of the total execution time or response time of the plan. It is important to have a cost model that closely reflects the actual execution cost of plans otherwise it may be the case that good plans are dismissed erroneously by the optimizer while enumerating through the search space. When considering the cost model in a distributed DBMS it is not enough just to consider the I/O cost. The cost of sending relations between sites via the network has a significant contribution to the overall execution time and should also be included in the cost model. The network transmission rate would therefore have to be available to the optimizer. However the network transmission rate is dependent on the network traffic and therefore can fluctuate over time. It is possible to make an optimistic assumption that the network traffic is low and use the baseline transmission rate of the network. However, in cases where network traffic varies significantly the optimizer would have to have a means of monitoring network traffic in order to determine the network transmission rate at runtime.
In a distributed DBMS there exists the possibility of executing parts of an execution plan in parallel. There are two main sources of parallelism - intra-operator and inter-operator. Intra-operator parallelism refers to the situation where a single operator is executed by multiple processors concurrently. This is only possible in situations where relations are fragmented across sites. Inter-operator parallelism is the term given to the case where two operators are executed concurrently. This includes the data-dependent case where two operators adopt a pipelining producer consumer relationship and also the data-independent case where two operators independently execute concurrently. As well as sources of parallelism there are also deterents of parallelism, which include resource contention and data dependencies.

Resource contention refers to the situation where multiple processes wish to access the same resource. Data dependencies exist when an operator is required to wait for its input operators to complete before it is able to proceed [7]. As mentioned previously, in a centralized setting the response time of a given execution plan can be estimated by considering resource consumption i.e disk usage. However, in a distributed setting a consequence of the existence of parallel execution opportunities is that resource consumption does not give a good indication of the plan response time.

There are other challenges of Query Optimization which can be summarized as follows:-

- The main challenge is to break a query in a distributed database environment into components that are isolated at different sites.
- Determining which node has the potential to give the fewest but qualified records because it affects the communication cost of the network. The minimum the data to transfer over the network, the less communication will be required for the same.
- Another challenge is to transfer this result to another node where additional work is yet to be performed on it.
- If there are more than two nodes, then they require even more complex analyses and more complicated heuristics to guide query processing.

- There is another challenge to find out the effective cost model at each and every site to compute the cost.

**CONCLUSION**

As the data is growing which is dispersed all over the world, on multiple sites. User is transparent about location, fragmentation and any other details of distributed databases. Whenever a query is submitted, role of optimizer is to find out the results in minimum time. But with the increased size of data and number of joins, a new efficient method should be devised to find out the result of a query in minimum time. This paper focuses on challenges appear in distributed environment, which if deal carefully will give better results.

**REFERENCES**