© 2017 IJSRST | Volume 3 | Issue 7 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology

Query Processing in the Crowdsourcing Environment

Cincy. W. C[1], J. R. Jeba[2]
[1]Research Scholar, Department of Computer Applications, Noorul Islam Centre for Higher Education, Kumara Coil, Tamilnadu, India
[2]Associate Professor & HOD, Department of Computer Applications, Noorul Islam Centre for Higher Education, Kumara coil, Tamilnadu, India

ABSTRACT

Optimization of the query is the biggest problem now days for crowdsourcing system. Crowdsourcing is source for the experts to solve the problem and freely sharing the answer with everyone also hiding the complexities and to relief the user from burden of dealing with the crowd. The user has to submit an SQL query and the system takes the responsible for compiling the query, generating the execution plan and evaluating in the crowdsourcing market. Query Processing is the scientific art of obtaining the desired information from a database system in a predictable and reliable fashion. Database systems must be able to respond to requests for information from the user i.e. process queries. In large database systems, which may be running on unpredictable and volatile environments, it is difficult to produce efficient database query plans based on information available solely at compile time. Getting the database results in a timely manner deals with the technique of Query Optimization. Efficient processing of queries is an important requirement in many interactive environments that involve massive amounts of data. Efficient query processing in domains such as the Web, multimedia search, and distributed systems has shown a great impact on performance. This paper will introduce the basic concepts of query processing and query optimization in the relational database. We also describe and difference query processing techniques in relational databases.

Keywords : Query Processing, Query Optimization, Database, Crowd Sourcing, Data Mining

I. INTRODUCTION

The field of crowdsourcing, which deals with solving hard problems by combining the power of machine and human computation, has gained a lot of traction in the last few years. Crowdsourcing techniques are used for solving problems that are hard for machines, such as comprehending and analyzing abstract concepts as well as media such as images, video and text.

Query optimization is an operation of frequent relational database management systems. The query optimizer workout to regulate the most active way to evaluate a given query by considering the possible query plans. Practically, the query optimizer cannot be getting directly by users once queries are submitted to database server, and parsed by the parser; they are then moved to the query optimizer where optimization occurs. However, some database engines grant guiding the query optimizer with hints. Queries results are produced by accessing relative database data and evaluating it in a way that return the requested information. By the reason of data-base structures are complicated, in most cases, and especially for not-very-simple queries, the needed data for a query can be gathered from a database by bring it in different ways, through different data-structures, and in different orders. Each different way commonly requires different processing time. Processing times of the same query may have high variance, a second to minutes, hours, depending on the way selected. The plan of query optimization, which is an automated process, is to find the way to process a user query in small amount of time. Hence there must be system that helps to analyze the query, optimize it, find query execution plans and finally predict potential query plan for execution over crowd sourced data.

The fundamental part of any DBMS is query processing and optimization. The results of queries must be available in the time frame needed by the submitting user. Query processing techniques based on multiple design dimensions can be classified as.

1. Query model: Processing techniques are classified according to the query model they assume. Some techniques assume a selection query model, where scores are attached directly to base tuples. Other techniques assume a join query model, where scores are computed over join results. A third category assumes an aggregate query model, where we are interested in ranking groups of tuples.

2. Data access methods:
Processing techniques are classified according to the data access methods they assume to be available in the underlying data sources. For example, some techniques assume the availability of random access, while others are restricted to only sort access.

3. Implementation level:
Processing techniques are classified according to their level of integration with database systems. For example, some techniques are implemented in an application layer on top of the database system, while others are implemented as query operators.

4. Data and query uncertainty:
Processing techniques are classified based on the uncertainty involved in their data and query models. Some techniques produce exact answers, while others allow for approximate answers, or deal with uncertain data.

5. Ranking function:
Processing techniques are classified based on the restrictions they impose on the underlying Ranking (scoring) function. Most proposed techniques assume monotone scoring functions.

II. QUERY PROCESSING

Query processing refers to the range of activities involved in extracting data from a database. The activities include translation of queries in high-level database languages into expressions that can be used at the physical level of the file system, a variety of query-optimizing transformations, and actual evaluation of queries. A database query is the vehicle for instructing a DBMS to update or retrieve specific data to/from the physically stored medium. The actual updating and retrieval of data is performed through various "low-level" operations [1]. Examples of such operations for a relational DBMS can be relational algebra operations such as project, join, select, Cartesian product, etc. [1].While the DBMS is designed to process these low - level operations efficiently, it can be quite the burden to a user to submit requests to the DBMS in these formats. There are three phases [1] that a query passes through during the DBMS' processing of that query:

1. Parsing and translation
2. Optimization
3. Evaluation

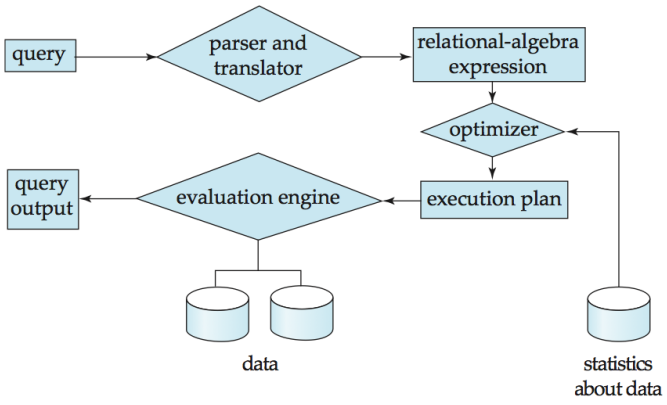

Figure 1. Steps in Query Process

The first step in processing a query submitted to a DBMS is to convert the query into a form usable by the query processing engine. High-level query languages such as SQL represent a query as a string, or sequence, of characters. Certain sequences of characters represent various types of tokens such as keywords, operators, operands, literal strings, etc. Like all languages, there are rules (syntax and grammar) that govern how the tokens can be combined into Understandable (i.e. valid) statements.

The primary job of the parser is to extract the tokens from the raw string of characters and translate them into the corresponding internal data elements (i.e. relational algebra operations and operands) and structures (i.e. query tree, query graph). The last job of the parser is to verify the validity and syntax of the original query string. In second stage, the query processor applies rules to the internal data structures of the query to transform these structures into equivalent, but more efficient representations. The rules can be based upon mathematical models of the relational algebra expression

and tree (heuristics), upon cost estimates of different algorithms applied to operations or upon the semantics within the query and the relations it involves. Selecting the proper rules to apply, when to apply them and how they are applied is the function of the query optimization engine.

The final step in processing a query is the evaluation phase. The best evaluation plan candidate generated by the optimization engine is selected and then executed. Note that there can exist multiple methods of executing a query. Besides processing a query in a simple sequential manner, some of a query's individual operations can be processed in parallel either as independent processes or as interdependent pipelines of processes or threads. Regardless of the method chosen, the actual results should be same.

Consider for example: **select** *balance* **from** *account* **where** *balance* < 2500. This can be translated into either of the following relational algebra expressions:

σbalance<2500 (Πbalance(*account*))
Πbalance(σbalance<2500(*account*))

Which can also be represented as either of the following query trees:
balance<2500
balance | | *balance*
balance<2500 | | *account account*

III. QUERY ALGORITHMS

Queries are ultimately reduced to a number of file scan operations on the underlying physical file structures [2]. For each relational operation, there can exist several different access paths to the particular records needed. The query execution engine can have a multitude of specialized algorithms designed to process particular relational operation and access path combinations.

A. Selection Algorithms The *Select* operation must search through the data files for records meeting the selection criteria. Following are some examples of simple (one attribute) selection algorithms [2]:

Linear search: Every record from the file is read and compared to the selection criteria. The execution cost for searching on a non-key attribute is *br*, where *br* is the number of blocks in the file representing relation *r*. On a key attribute, the average cost is *br*/2, with a worst case of *br*.

2. Binary search: A binary search, on equality, performed on a primary key attribute has a worst-case cost of log (*br*). This can be considerably more efficient than the linear search, for a large number of records.

3. Search using a primary index on equality: With a B+-tree index, an equality comparison on a key attribute will have a worst -case cost of the height of the tree plus one to retrieve the record from the data file. An equality comparison on a non-key attribute will be the same except that multiple records may meet the condition, in which case, we add the number of blocks containing the records to the cost.

4. Search using a primary index on comparison: When the comparison operators (<, >) are used to retrieve multiple records from a file sorted by the search attribute, the first record satisfying the condition is located and the total blocks before (<) or after (>) is added to the cost of locating the first record.

5. Search using a secondary index on equality: Retrieve one record with an equality comparison on a key attribute; or retrieve a set of records on a non-key attribute [2]. For a single record, the cost will be equal to the cost of locating the search key in the index file plus one for retrieving the data record. For multiple records, the cost will be equal to the cost of locating the search key in the index file plus one block access for each data record retrieval, since the data file is not ordered on the search attribute.

B. Join Algorithms The join algorithm can be implemented in a different ways. In terms of disk accesses, the join operations can be very expensive, so implementing and utilizing efficient join algorithms is important in minimizing a query's execution time [3]. The following are 4 well - known types of join algorithms:

1. Nested-Loop Join: It consists of an inner for loop nested within an outer for loop [3].

2. Index Nested-Loop Join: This algorithm is the same as the Nested-Loop Join, except an index file on the inner relation's join attribute is used versus a data-file scan on each index lookup in the inner loop is essentially an equality selection for utilizing one of the selection algorithms Let *c* be the cost for the lookup, then the worst -case cost for joining rand sis *br* + *nr* * *c*.

3. Sort –Merge Join: This algorithm can be used to perform natural joins and equi-joins and requires that each relation be sorted by the common attributes between them [4].

4. Hash Join: The hash join algorithm can be used to perform natural joins and equi-joins. The hash join utilizes two hash table file structures (one for each relation) to partition each relation's records into sets containing identical hash values on the join attributes. Each relation is scanned and its corresponding hash table on the join attribute values is built.

C. Indexes Role The execution time of various operations such as select and join can be reduced by using indexes [5]. Let us review some of the types of index file structures and the roles they play in reducing execution time and overhead:

1. Dense Index: Data-file is ordered by the search key and every search key value has a separate index record. This structure requires only a single seek to find the first occurrence of a set of contiguous record s with the desired search value [6].

2. Sparse Index: Data-file is ordered by the index search key and only some of the search key values have corresponding index records. Each index record's data-file pointer points to the first data-file record with the search key value. While this structure can be less efficient than a dense index to find the desired records, it requires less storage space and less overhead during insertion and deletion operations.

3. Primary Index: The data file is ordered by the attribute that is also the search key in the index file. Primary indices can be dense or sparse. This is also referred to as an Index-Sequential File [7].

4. Secondary Index: The data file is ordered by an attribute that is different from the search key in the index file. Secondary indices must be dense.

5. Multi-Level Index: An index structure consisting of 2 or more tier s of records where an upper tier's records point to associated index records of the tier below. The bottom tier's index records contain the pointers to the data-file records. Multi-level indices can be used, for instance, to reduce the number of disk block reads needed during a binary search.

6. Clustering Index: A two-level index structure where the records in the first level contain the clustering field value in one field and a second field pointing to a block [of 2nd level records] in the second level. The records in the second level have one field that points to an actual data file record or to another 2nd level block [8].

7. B+-tree Index: Multi- level index with a balanced-tree structure. Finding a search key value in a B+-tree is proportional to the height of the tree maximum number of seeks required is log (*height*). While this, on average, is more than a single -level, dense index that requires only one seek, the B+-tree structure has a distinct advantage in that it does not require reorganization, and it is self-optimizing because the tree is kept balanced during insertions and deletions.

6. Choice of Evaluation Plans

The query optimization engine is used to generate a set of candidate evaluation plans. Some will use the heuristic theory which produces a faster and more efficient execution. Then the others may produced by the prior historical results which are more efficient than the theoretical models, this can be used very well in case of queries dependent on the semantic nature of the data to be processed. Still others can be more efficient due to "outside agencies" such as network congestion, competing applications on the same CPU, etc. [9]

IV. Conclusion

The most important functional requirements of a database system are to process queries in a timely manner. This is particularly true in case of very large and mission critical applications such as weather

forecasting, banking systems and aeronautical applications where they have millions and even trillions of records containing data and it becomes hard to store and to retrieve data from them. The need for faster and faster, "immediate" results never ceases. Some of the basic techniques and principles with examples of query processing and optimization is mentioned here in this paper.

V. REFERENCES

[1]. Introduction to Query Processing and Optimization by Michael L. Rupley, Jr.

[2]. K. Kiran Kumar[1], T.M. Santhi Sri [2], Voruganti Vamshi priya [3], "Introduction to Techniques of Query Processing and Optimization", International Journal of Innovative Research in Advanced Engineering ., Volume 2,Issue 3, March 2015.

[3]. Jagjit Bhatia," Analytical Evaluation of Different Query Optimization Techniques", THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE., Vol 3, Issue 3, March, 2015.

[4]. E.J.Thomson Fredrick1 and G.Radhamani2," INFORMATION RETRIEVAL USING XQUERY PROCESSING TECHNIQUES", International Journal of Database Management Systems. Vol.3, No.1, February 2011.

[5]. G. R. Bamnote, S. S. Agrawal," Introduction to Query Processing and Optimization", International Journal of Advanced Research in Computer Science and Software Engineering. Vol 3, Issue 7, July 2013.

[6]. Daodu, S. S., Akhatuamen, S., Folorunso, O. and Ukaoha, K.C.", Query Processing and Optimization in a Distributed Database System", Computing, Information Systems, Development Informatics & Allied Research Journal. Vol. 7 No. 1. March, 2016.

[7]. Kind of El Marry, Christoph Lofi and Wolf-Tilo Balke," Crowdsourcing for Query

[8]. Processing on Web Data: A Case Study on the Skyline Operator", Journal of Computing and Information Technology. October, 2014.

[9]. Avi Silbershatz, Hank Korth and Sudarshan. Database System Concepts, 4th Edition. McGraw-Hill, 2002.

[10]. Vandana Jindal 1, A.K.Verma 2, Seema Bawa 2," Survey on Query Processing & Optimization Techniques in WSN", International Journal of Computer Science and Information Security. Vol. 14, No. 2, February 2016.