DISTRIBUTED QUERY OPTIMIZATION USING HILL CLIMBING ALGORITHM FOR COMPLEX CHURCH DATABASES

Esiefarinenh Michael Bukohwo¹, Philemon Uten Emmoh² and Choji Davou Nyab³
¹²Department of Mathematics/Statistics/Computer Science, University of Agriculture, Makurdi, Nigeria.
³Department of Computer Science, University of Jos, Nigeria.

Abstract. The NP-hard join ordering problem is a fundamental issue any optimizer must resolve to produce an optimal execution plan for queries. This problem becomes even more complex when the databases become distributed. With distributed databases comes the problem of query results optimization from multi-execution plan. The issues addressed in this paper includes fast retrieval of queries, high reuse of cached queries and the decomposition of complex queries into smaller queries for faster accessibility by the optimizer. The optimization of the queries was done using the hill climbing algorithm. The data for the queries were collected from a Church Home Cell database. Experiments were performed to compare memory utilization, and speed of execution on optimized and non-optimized queries. The results show that the optimized database was faster and better utilized memory compared to non-optimized queries.

Keywords: query optimization, cached query, query execution, Hill Climbing algorithm.

I. INTRODUCTION

Query optimization in object distributed databases uses query optimization as a caching technique to improve query execution and to enhance fast retrieval and high reuse of cached queries. The query optimizer is widely considered to be the most important part of a database system. The aim of the optimizer is to take a user query and to provide a detailed plan called a Query Execution Plan (QEP) that indicates to the users exactly how the query should be executed [8]. This paper addresses query caching for handling wider queries by using some parts of cached results helpful for retrieving other queries (wider Queries) and combining many cached queries when producing the result. This certainly entails performing multiple experiments on the distributed database to guarantee such productivity.

Data sources are usually centralized in that a single mediator system is placed between a number of data sources and applications. As the number of data sources increases, the centralized mediator architecture becomes an administrative and performance bottleneck. Though their work was complex, they succeeded in developing Query Decomposition for a Distributed Object-Oriented Mediator System using the mediator-wrapper approach [12].

The authors in [8] developed an optimization method to improve query optimization that selects an optimal plan in which the sub-query has the same tables and conditions that are for the outer query (intra query redundancy). However, this degrades the query performance of correlated nested queries. Their heuristic strategies enhances query processing by performing selection operations first in order to limit the number of rows/tuples, then limit the number of columns by performing projection operations, perform the operations with the smaller or simple join first if there are consecutive join in the query and finally, saved the result for the same expression for future use.

Distributed query optimization algorithm for two-step pruning performs a pruning step twice for each sub-query by designing two separate equivalent criteria applicable to each sub-query. This lessens the search work done by the optimizer considerably. Without losing optimality, the search space for finding the optimum is reduced by aggregating partial plans that always incur the same processing time into a single plan and eliminating partial plans that can never be the optimum [4].
Most modern algorithms for basic relational operators use DBMS statistics to estimate their memory requirement which, in turn, determines the algorithms’ performance. The collected statistics can be used to obtain more accurate estimates for the remainder of the query or, if necessary, to create a better QEP for the query [5].

An Ingres algorithm fragment and replicate query processing strategy to achieve high degree of parallelism by partitioning one relation among the processing sites and replicating all other needed relations. Exact optimization of query is not possible as accurate database statistics is not available. Therefore the authors in [1] used different components of query optimization and various algorithms in their work. They also discussed the advantages and disadvantages of query optimization where multiple factors for optimization are involved.

The authors in [10] used the hill climbing algorithm to minimize the sustained throughput of distributed continuous queries; they succeeded to optimize stream queries with respect to a version of throughput measure, the profiled input throughput. This measure is focused on matching the expected behavior of the input streams. To prune the search space they used hill-climbing techniques that proved to be efficient and effective.

The authors in [11] suggested the heuristic approach for selecting the optimal evaluation plan and Semi-join approach for reducing the communication cost. According to them, calculating the cost of each evaluation plan of a query takes lots of computational efforts as well as time. Initially, the heuristic approach is used to find best evaluation plan among various plans of a single query. They further explained that by considering only joins, queries are divided into two: tree and cyclic.

The authors in [2] discussed the basics of query optimization and its key components like search spaces and accurate cost estimation technique. An efficient algorithm was been proposed to generate the best execution plan. According to them, there are many phases when a query is submitted to a database server namely:

- The first phase is called parse tree,
- The intermediate phase is called logical operation tree
- And the final representation is called the operator tree

The authors in [7] developed a model for query decomposition and answer construction in heterogeneous distributed database system that is able to answer some problem of retrieving information from a collection of heterogeneous distributed databases. However, the task of integrating established database systems is complicated not only by the differences between the database systems themselves, but also by the differences in structure and semantics of the information contained within them. The problem is exacerbated when one needs to provide access to such a system for naive end-users. Their work although complex, was able to present a Knowledge-Based Systems approach in solving this problem for clearly bounded situations, in which both the domain and the types of query are constrained. At the user interface, dialogue is conducted in terms of concepts with which the user is familiar, and these are then mapped into appropriate database queries. To achieve this, they developed a model for query decomposition and answer construction. This model is based around the development of an Intentional Structure containing information necessary for the recapture of semantic information lost in the query decomposition process and required in the answer construction process.

The author in [6] addresses the processing of a query in distributed database systems using a sequence of semi joins. The objective is to minimize the inter-site data traffic incurred by a distributed query. The method accurately and efficiently estimates the size of an intermediate result of a query. It provided the basis of the query optimization algorithm. Since the distributed query optimization problem is known to be intractable, a heuristic algorithm was developed to determine a low-cost sequence of semijoins. The cost comparison with an existing algorithm is provided. The complexity of the main features of the algorithm is analytically derived. The scheduling time for sequences of semijoins is measured by using a PASCAL program to implement the algorithm.
This paper is geared towards implementing the HILL CLIMBING algorithms in a distributed database to achieve faster query optimization in a network system using distributed church home cells.

II. METHODS

This work uses hill climbing algorithm [3] to develop a distributed query optimization algorithm. The hill climbing algorithm is used for query optimization. A distributed church database was be used for managing home cells to implement the algorithm.

A. Hill Climbing Algorithm

The hill-climbing algorithm proceeds as follows [3]:

Step 1. Select initial feasible execution strategy ES0 i.e., a global execution schedule that includes all inter site communication
- Determine the candidate result sites, where a relation referenced in the query exist
- Compute the cost of transferring all the other referenced relations to each candidate site
- ES0 = candidate site with minimum cost

Step 2. Split ES0 into two strategies: ES1 followed by ES2
- ES1: send one of the relations involved in the join to the other relation’s site
- ES2: send the join result to the final result site

Step 3. Replace ES0 with the split schedule which gives
\[
\text{cost}(\text{ES1}) + \text{cost(local join)} + \text{cost(ES2)} < \text{cost(ES0)}
\]

Step 4. Recursively apply steps 2 and 3 on ES1 and ES2 until no more benefit can be gained

Step 5. Check for redundant transmissions in the final plan and eliminate them

Using the model of the database structure of Figure 1, the following model database statistics are derived.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Size</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMBERS</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>DISTRICTS</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>HOMECELL</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>CELLMEMBERS</td>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1. Model Database Statistics

Assumptions:
- Size of relations is defined as their cardinality (number of tuples)
- Minimize total cost
- Transmission cost between two sites is 1
Ignore local processing cost
- size(MEMBERS \times HOMECELL) = 8, size(HOMECELL \times CELLMEMBERS) = 2, size(CELMEMBERS \times MEMBERS) = 12

To determine the initial feasible execution strategy, we consider the following alternatives

Alternative 1: Resulting site is site 1
Total cost = cost(HOMECELL → Site1) + cost(CELMEMBERS → Site1) + cost(MEMBERS → Site1)
Total cost = 4 + 12 + 8 = 24

Alternative 2: Resulting site is site 2
Total cost = 1 + 4 + 12 = 17

Alternative 3: Resulting site is site 3
Total cost = 1 + 4 + 8 = 13

Alternative 4: Resulting site is site 4
Total cost = 8 + 12 + 1 = 21

Therefore ES0 = DISTRICT → Site3;
HOMECELL Site3;
MEMBERS Site3

III. RESULTS AND DISCUSSION

Figure 2. General Performance of memory Optimization

Figure 2 shows a general graph of memory performance, this graph is made up of number of records (x-axis) and time of processing in seconds (y-axis). The blue curve is tracing the performance of result optimization, while the purple curve is tracing normal result and the red curve represent noise in memory frequency.

Figure 3. Initial Optimization Performances
Figure 3 At this point, the initial optimization result is running at 2.9659869194 seconds when the records are 35 while the normal result at records 35 is 1.20855493546. At the first run, the optimization result is observed to be higher than the normal result. Here, the purple curve behaved better than the blue in performance, this is because at some certain intervals the system carries out performance optimization which causes the optimization result to be delayed as shown in the figure 3 above.

Figure 4 Here, at the Subsequent optimization result, the optimization process data is cached in memory resulting to an optimized performance. However, at instances of optimization processes the purple curve still performs better than the blue curve this is because as data were cached in memory to optimized performance the blue curve increases to 3.64363880157 seconds as a result of the noise at 54 records also at first run which is far greater than the normal result 1.20699400902 still at 54 records.

Figure 5. Optimized Performance Result
Figure 5 Here, as optimization was completed at 35 records when the record was queried 3 times the time taken by optimization dropped to 0.014213804303 which shows that the performance of the blue curves is better than the purple curve whereas the time taken by normal result at 35 records becomes 1.20904297829 which is higher than the optimization result as shown in figure 5.

IV. CONCLUSION

Considering data needs and its exponential growth in many organizations, it is of importance to seek ways to optimized data storage and retrieval algorithm. In this work, query decomposition and optimization algorithm on a distributed database network was achieved. The results have clearly shown the effect of optimization on CPU processing time as most database application systems grow slower in processing over time. This growth is due to several reasons including 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 a single database, or two different spots in multiple software environments or platforms. By decomposing and optimizing distributed queries these problems are eliminated thereby achieving a better performance.

This work used the Hill Climbing algorithms to implement a distributed query decomposition and optimization algorithms. The algorithm is used to perform distributed query optimization algorithm on data obtained from a church home cell.

REFERENCES


