ABSTRACT

Growing demand for massive data [1] processing and analysis applications has motivated the researchers worldwide to invent new and efficient techniques for data retrieval. Depending upon the query fired, accessing this big data may take large time. It is also observed that most of the queries are fired on very few amounts of the data. A mechanism must be developed to handle this few data and the respective queries. The queries can be matched using semantic analysis. Also processing this massive data may take a considerable amount of time.

The information is retrieved from the massive data everytime a query is fired. The same procedure gets repeated, if the same query is fired again. We propose a method to temporarily store these results into some other data base and if some similar query is fired, access this data base later to fetch the results just before the main query execution.

Here we propose a method for fast execution of these repeating queries to retrieve the useful data using semantic processing and Query Optimization.

Index Terms- Massive Data, Query Optimization, Semantic Processing.

I. INTRODUCTION

In the starting days, traditional databases such as MySQL, Oracle, and DB2 etc were the main area of interests for most of researches on query optimization. Later they started storing more and more data into the distributed environments [1]. And hence became the main area of interest for many research theories and methods. The development of cloud computing and cloud storage technologies is leading the query optimization of massive data to be more popular in the recent researches and studies. The processing of such massive data covers non only query optimization for processing uncertain data but also structured data processing, non-structured data processing and semi-structured data processing. MapReduce is mainly used in the massive non-structured data processing. For improving the processing efficiency to the massive data, many other internet big data computing framework such as Hadoop, Hadoop++, Spark, CrowdDB and Yale university's HadoopDB are proposed based on the MapReduce [3][1], in the past years. Improving the query efficiency to massive data is the only objective of all such new computing frameworks is. Query optimization methods play a big role for improving the efficiency to this massive data processing.

Semantic Matching [4] focuses on the interactions among word-level meanings in sentence to determine the possible meanings of it. Semantic query optimization not only reduces unnecessary data transmission but also provides local optimization to the sub queries. Using semantic rules about data for transforming a query into a more efficient as well as semantically equivalent query is the main idea behind semantic query optimization. We say two queries are semantically equivalent if they return identical answers from a database that is consistent with the semantic knowledge [4]. It is difficult to encode useful semantic knowledge hence semantic query optimization is not widely used in practice.

While accessing the information from the massive data, some queries repeat more often, some repeat a very few times while some don’t repeat at all. In order to reduce the time, the results of such frequently repeating queries can be stored into some temporary database or buffer and later it can directly be retrieved when the similar query is fired. These queries can partially or exactly be similar to each other. The matching between these queries is calculated using Semantic analysis.

This paper discusses about the method to retrieve the data faster by fast execution of ad-hoc queries using offline analysis [2] and semantic matching. The similarity measure of the queries is calculated by deciding a threshold. The input is taken as ad-hoc queries, then these queries are checked for semantic matching, then the threshold is checked, and the result is fetched depending on the threshold.

II. RELATED WORK

A lot of work has already been done in the field of data mining and semantic analysis. This paper is motivated from the SemanQuery Architecture [1], by Guigang Zhang et al and the S^4 System [2] by Xiao Yu et al. First paper introduces the semantic analysis of the queries from the big query network. If the big query network has the query plans similar to that of the query plans of the user, SemanQuery will get the query plans' query paths in this big query network and execute those query plans respective to the query paths. If query plans of the query fired by the user are not found by the SemanQuery architecture in the big query network, it will create a new big query network by adding these new query plans to big query network.

In the S^4 System a sub graph query is given as an input, and a list of sub graphs that satisfy the query criteria is retrieved.

Index Terms- Massive Data, Query Optimization, Semantic Processing.
from an information network. Off-line data mining results are produced by the S4 System as indices, the semi-structure information encoded in information networks is utilized by it and the answers to the semantically similar sub-structure queries are given efficiently. The S4 system uses Structured Indexing and Similarity Indexing.

III. PROPOSED APPROACH

- **Assumptions**
  i. A network of all the previously executed queries
  ii. A small network of semantically matched queries with comprehensive information.
  iii. A big structured database.

There are two parts in the figure given below:

The left part includes a big data, a network of all the previously executed queries and a small network of semantically matched queries, i.e. all the assumed part. Some temporary buffer is used to store the results of these queries so as to fetch them directly when a similar query is fired.

The right part is discussed below:

a. This part takes as input the user queries.
b. The next part performs the runtime analysis of this query, in which these queries are checked for semantic matching.
c. Query evaluation is performed depending upon the semantic matching (similarity measure) score.
d. A threshold is decided to calculate the similarity measure.
e. The result that is stored is fetched directly without executing the original query if the similarity measure is greater than the threshold. Otherwise, the query is forwarded for normal execution.
f. Thus the time required to fetch the similar data again and again from the massive data will be saved.

This paper mainly focuses on Similarity Measure i.e. Semantic Matching and Validation part. The assumptions include a big query network of previously fired queries, massive structured data and some simple input query for a start.

Threshold and various semantic techniques are used to calculate similarity measures. Views can be used to store the temporary results. Later these views can be easily and efficiently updated whenever a new result is added to it.

The intermediate results can be fetched from the view. We can limit the range of this view i.e. our temporary database in order to save from the memory management problem. If the range of the present view is smaller than that in the result, the view will be updated to store this new range. This will speed up the information retrieval from the massive data as the original database is much bigger than the view.

Two separate databases are needed to update continuously for every new query being executed, one for the fired queries that are stored and the other to store the results temporarily.

IV. EXECUTION AND ANALYSIS

1. Initially when the Query Network is empty and a query is fired, the query is executed normally and the result is stored into a buffer and the Query is added into the Query Network. The User Interface looks like the figure given below:
2. When the similar query is fired with the less or equal amount of data than that is stored into the buffer,

The result is fetched directly from the buffer, without the query being fired on the big database as shown below:

3. When the similar query is fired with more amounts of data than that is stored into the buffer,
The query is fired on to the big database again, but the Query Network is updated with the new limits and the buffer is filled with the new result, as given below:

```
item_no stock
01 459
02 466
03 472
04 479
05 486
06 493
07 499
08 506
09 513
10 520
```

**Analysis:**

```
Analysis of 100 Queries
```

![Analysis of 100 Queries](image)

VI. ACKNOWLEDGMENTS

Our thanks to IJARECE for allowing us to modify templates they have developed.

VII. REFERENCES


[2] Query-Driven Discovery of Semantically Similar Substructures in Heterogeneous Networks, Xiao Yu, Yizhou Sun, Peixiang Zhao, Jiawei Han Department of Computer Science, University of Illinois at Urbana-Champaign.


VIII. Author Profiles

1. Vaibhav Jain has received his B.E. degree in Computer Engineering from MAEER’s Maharashtra Academy of Engineering, Alandi, Pune, Pune University in 2011. He has studied Mtech in Computer Science and Engineering from Shri Ramdeobaba College of Engineering and Management.
(Autonomous), Nagpur. Currently he is working with Persistent Systems Ltd., Pune. His research interests include query optimization and semantic analysis.

2. **Mangesh Wanjari** has received his B.E. in Computer Technology from Nagpur University in 2002. He has received his Master of Technology (MTech) in Computer Science and Engineering from VNIT, Nagpur in 2009. After having some industrial experience he has joined the teaching field. He is an associate professor in Ramdeobaba College of Engineering and Management, Nagpur. His research interests include Database Technologies, Query Optimization and Semantic Analysis.