An Effective Method to Answer OLAP Queries using R*-Trees in Distributed Environment

F. Sagayaraj Francis
Pondicherry Engineering College
Puducherry
India

P. Xavier
Department of Computer Applications, SCSVMV
Enathur
Kanchipuram

ABSTRACT
Evaluation of OLAP queries is one of the challenging tasks in a database system. Attempts are being continuously made to improve the efficiency of the methods that answer OLAP queries. This paper makes one such attempt. This paper proposes a method in a *Hadoop* and *MapReduce* distributed environment. Experimental evaluation gives improved results due to the proposed method.

General Terms
Distributed systems, OLAP, R*-tree.

Keywords
OLAP, R*-tree, *MapReduce*.

1. INTRODUCTION
Data warehouse summarize and integrate voluminous data accumulated over a large period of time in operational databases. On-line Analytical Processing (OLAP) is a technology that applies various set of procedures on data warehouses for analyzing the amassed data for the effective decision making. The data in the data warehouses are modelled as structures generally referred as star schema and snowflake schema. The center of these schemas is the fact table whose rows correspond the consolidated transactions from the operational databases. While a few columns of the fact table capture the measures, the other columns capture the values of the dimensional attributes. These dimensional attributes refer to selectively expressed business perspectives. These dimensional values may also be organized in some hierarchical business perspective. A sample star schema is given in Fig 1. When the dimension tables themselves become fact tables, then the schema is called as snowflake schema. A sample snowflake schema is shown in Fig 2. More conveniently fact tables are modelled as data cubes as shown in Fig 3. Some of the operations that can be carried out on a cube include slicing, dicing, rollup, drilldown and pivot.

We are in an age where the data that are generated is complex, varied and voluminous and come at a rate and from places that are challenging for database systems. The existing traditional Database Management, processing and analyzing systems find it difficult to cope with such tremendous demands. The introduction of *Hadoop* and *MapReduce* [1] provided the opportunity for the management and processing to go distributed. *Hadoop* is a distributed computing framework, where clusters with many computing and storage facilities are dynamically formed. The foundation of *Hadoop* is the *Hadoop Distributed File System* (HDFS). HDFS consists of *name nodes* and *data nodes*. While the *name nodes* manage the storage system, the *data nodes* actually store the data with controlled redundancy. There are generally several *name nodes* and multitude of *data nodes* in a typical cluster of HDFS. The management of the clusters are transparent to the users. The HDFS architecture is given in Fig 4. The jobs are submitted to the *name nodes* as *MapReduce* programs. The execution and the production of results are seamlessly managed by the HDFS from thereon.

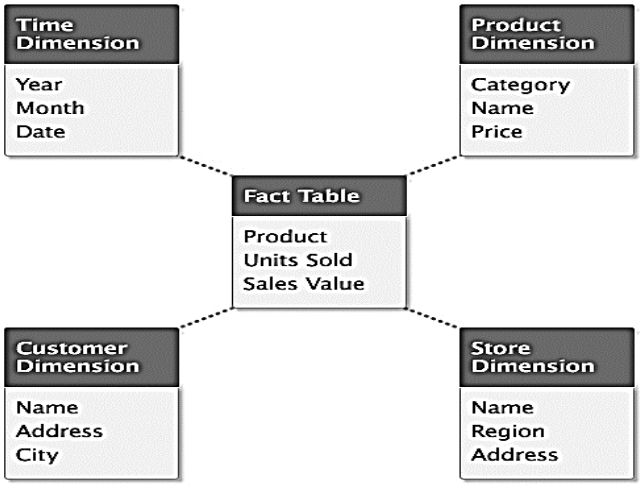

Fig 1: **A sample star schema**

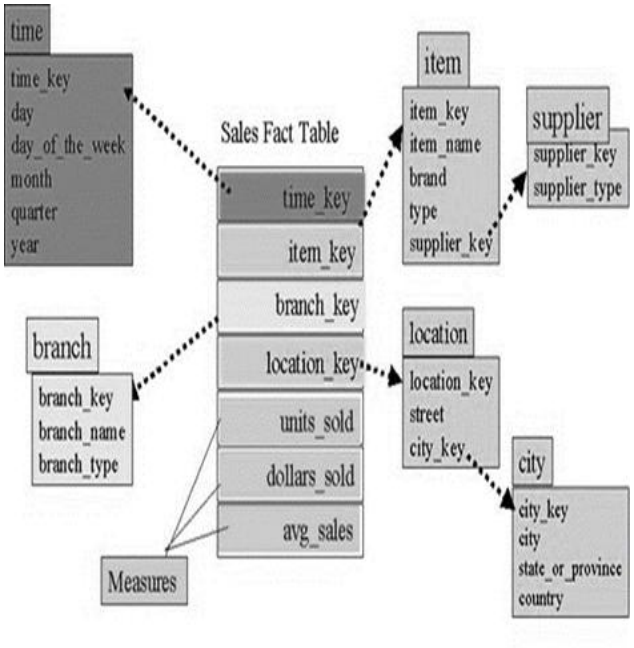

Fig 2: A sample snowflake schema

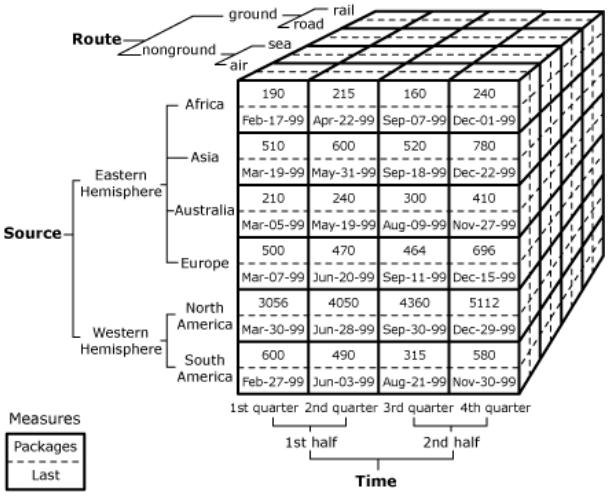

Fig 3: A sample OLAP cube

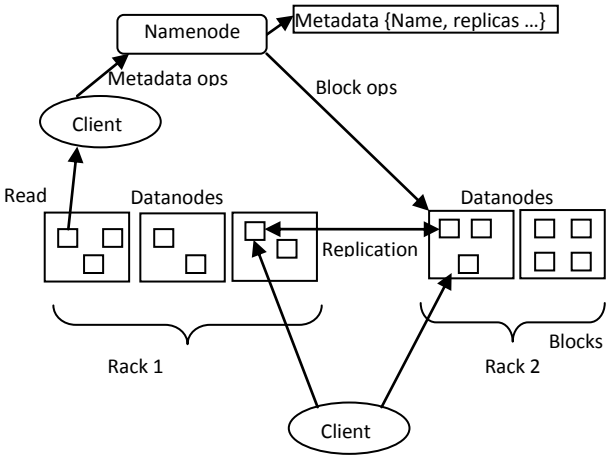

Fig 4: HDFS Architecture

MapReduce is a programming paradigm for *Hadoop* distributed computing framework. A *MapReduce* program consists of a pair of user defined *map* and *reduces* functions. The map function is invoked for every record in the input data sets and produces a partitioned and sorted set of intermediate results. The reduce function fetches sorted data, from the appropriate partition, produced by the map function and produces the final output data. Conceptually, they are: *map*(k_1, v_1) → *list*(k_2, v_2) and *reduce*(k_2, *list*(v_2)) → *list*(k_3, v_3). *Combine* (k_2, *list*(v_2)) → *list*(k_2, v_2) is an optional intermediate stage for combining or merging the results of *map* function.. An overview of *MapReduce* is given in Fig 5.

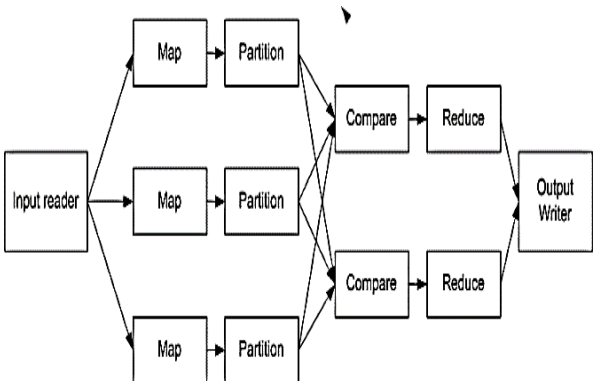

Fig 5: Overview of MapReduce paradigm

An R-tree [2] is a multidimensional and dynamic indexing structure. It organizes the index in a hierarchical fashion. It partitions the data and organized them in a structured space that are overlapping and whose bounding edges are orthogonal to the axes of the space. The R*-tree [3] is an improved model of the variants of R-tree. It forms the basis of all the multidimensional indexing methods based on data partitioning that are in existence today. A sample 2-dimensional data set and the corresponding R*-tree is shown in Fig 6. The terms R-trees and R*-trees are used interchangeably in this research synopsis document. The R-tree uses a *d*-dimensional Minimum Bounding Rectangles (MBRs) to cover the objects in a *d*-dimensional space. In Fig 6,the MBR with identity *R3* demonstrates the coverage of an object by a MBR. These MBRs are grouped together in leaf nodes according to their spatial proximity, which are then recursively grouped in higher levels up to the root.

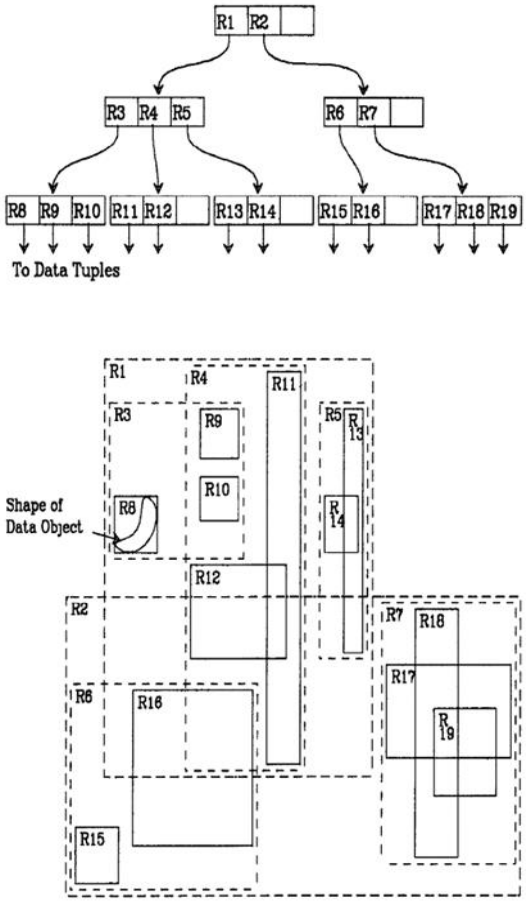

Fig 6: A sample 2D space and the corresponding R-tree

Every leaf node of the R*-tree contains a minimum of *M/2* index records and a maximum of *M* index records unless it is the root. *M* is the order of the R*-tree that specifies the maximum number of entries that can be accommodated in one node. For each index record *(I, tuple-identifier)* in a leaf node, *I* is the smallest rectangle that spatially contains the *d*-dimensional data object represented by the tuple-identifier. In every non-leaf entry *(I, child-pointer)* in a non-leaf node, *I* is the smallest rectangle that spatially contains the rectangles in the child node. Non-leaf MBRs are also called as directory rectangles. The root node has at least two children unless it is a leaf. All leaves appear on the same level. MBRs as well as partitions of R*-trees overlap each other. The space occupied

by a node is called as node region. During the process of insertion and deletion the R-tree attempts to minimize the total area covered by directory rectangles, minimize the overlap between directory rectangles, minimize total margin length of directory rectangles, optimization of storage utilization by increasing the fill ratio of the MBRs and minimize the number of node splits. A comprehensive discussion on R-trees and their clones are given in [4], [5] and [6].

2. LITERATURE REVIEW

One of the initial methods that was suggested to manage the data warehouse for answering OLAP queries is to compute all enumerable groupings of measures accommodating the dimensions and their hierarchies. This technique may not be feasible because of huge space and time constraints. Hence several partial materialization techniques [7] were suggested to identify the frequently queried hyper-cubes and frequently computed measures to apply OLAP computations that saved time and space. Nevertheless these techniques could not deal with the ad-hoc requirements that were required to cater the need for the user requirements. One of the noteworthy method that uses ad-hoc grouping used aggregate R-tree [8, 9] or *aR*-tree. The method augments the R-tree and constructs the features for finest granularity of the OLAP dimensions. While the structures of both trees are similar, the difference is that along with directory information that is available in R-tree, aggregation results are also stored in *aR*-tree for the cube represented by the node. A sample *aR*-tree is given in Fig 7. The method uses a branch-and-bound algorithm that accesses a minimal number of tree nodes in order to compute the top-*k* groups.

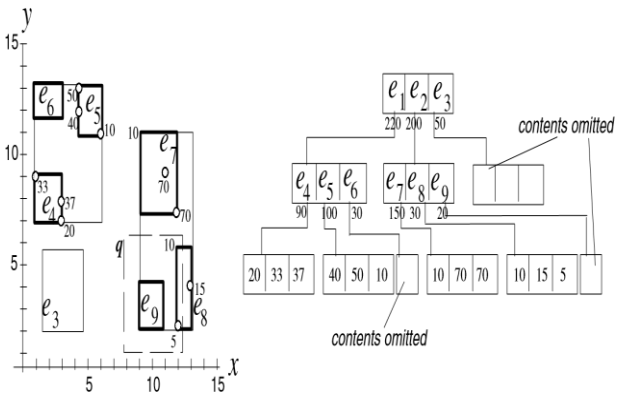

Fig 7: A sample aR-tree

A recent approach that has been reported in the literature for a cloud based real-time OLAP system uses the distributed PDCR tree [10]. It uses a cloud infrastructure consisting of multi-core processors. The distributed PDCR tree data structure supports multiple dimension hierarchies with efficient query processing for multitude hierarchical dimensions that are central to OLAP systems. It is particularly efficient for complex OLAP queries that need to aggregate large portions of the data warehouse. A *MapReduce* based architecture for OLAP computations has been proposed in [11]. It extends the existing preliminary batch processing model for OLAP to streaming data. The method improves the efficiency by reducing the job completion times and improving the system utilization. The baseline methods for evaluating OLAP queries over distributed systems have been proposed only on whole datasets and not on organized datasets such as R*-trees.

3. PROPOSED METHOD

The proposed method to evaluate the OLAP query constructs a stream of mappers and reducers depending on the number of dimensions of the cube. For an *n*-dimensional cube *n* mappers and *n* reducers are constructed. Let the mappers be identified as $M_2, M_3, , M_n$. Let the reducers be identified as $R_1, R_2, ., R_{n-1}$. In general Mapper M_i reads an *i*-dimensional cube and split the it into *i-1* dimensional cubes. The key of each *i-1* dimensional cube is one of the values belonging to the domain of the dimension. The cubes are scuffled into proper reducers. Reducer R_i computes the aggregate values of the cubes supplied to it. The above method is described in detail below.

An R-tree is a nothing but a cube organized in a hierarchical structure. Hence the mappers, instead of exhaustively processing the entire cube, process only small portions that are represented by MBRs. The subtrees of the R*-tree are fed into each mapper identified as M_n. These mappers generate a unique key for every value in each dimension giving *<key, n-1 dimensional cube>*, where key is the value in consideration. These *n-1* dimensional cubes are fed into reducers identified by R_{n-1}. These reducers compute the aggregation values of *n-1* dimensional cubes. These cubes, after the computation is over, are read by mappers identified by M_{n-1}, they in turn generate *n-2* dimensional cubes which are read by reducers identified by R_{n-1}. This process is continued till *1*-dimensional aggregates are obtained.

For the evaluation of an OLAP query on a sub cube, the sub trees that intersect the sub cube are considered and the above said process in applied of the selected sub trees.

The above method may be improved by writing an one-level recursive mapper that breaks the *n*-dimensional cube into *n-1* to *1*-dimensional cubes at the first instant itself.

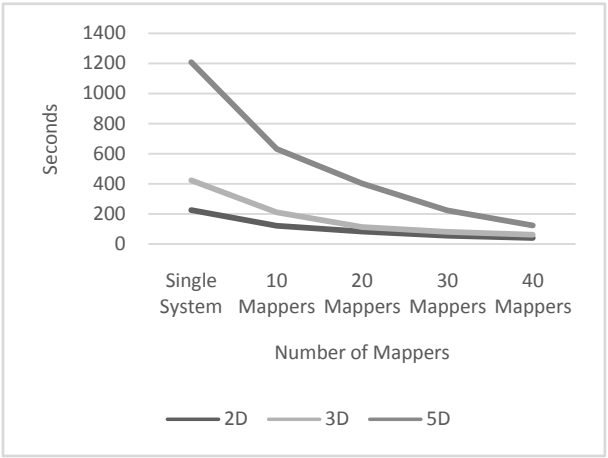

Fig 8: Comparison of the time taken to evaluate the OLAP queries

4. EXPERIMENTAL RESULTS

For conducting the experiments three data cubes of dimensions 2, 3 and 5 were setup. Each dimension had 50 distinct values. The 'cell' had a float value. The average() aggregate function was evaluated. The experiments were conducted on a *Hadoop* cluster by varying the number of mappers. The results are shown in Fig 8. The time taken to evaluate the OLAP query was steadily falling down as the number of mappers increased. This is obvious because of the inherent advantages of distributed processing supported by *Hadoop*. Repetition of experiments for sub cubes reflected the trend that is presented in the graph.

5. CONCLUSION

Evaluation of OLAP is a challenging task. This paper has proposed a method for evaluating the OLAP queries on a data cube that is represented as R-tree in a *Hadoop* and *MapReduce* environment. The results endorse the obvious that faster evaluation of OLAP queries can be achieved by distributed processing.

Even though the new method proposed in this paper gives appreciable results, a lot of redundancies of the sub cubes were observed after the *map* process. These redundancies have to be effectively controlled to further improve the performance of the new method.

6. REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: simplified data processing on large clusters", Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation, vol. 6, 2004.

[2] Guttman, "R-trees: A dynamic index structure for spatial searching," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 47-57, 1984.

[3] N. Beckmann, H. -P. Krieger, R. Schneider and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 322-331, 1990.

[4] V. Gaede and 0. Guenther, "Multidimensional access methods," ACM Computing Surveys, vol. 30, no. 2, pp. 170-23 1, 1998.

[5] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos and Y. Theodoridis, "R-trees have grown everywhere," Technical Report, Available at http://citeseer.ist.psu.edu/706599.html (2003).

[6] S. Brakatsoulas, D. Pfoser and Y. Theodoridis, "Revisiting R-tree construction principles," Proceedings of the 6th ADBIS Conference, pp. 149-162, 2002.

[7] Z. X. Loh, T. W. Ling, C.-H. Ang, and S. Y. Lee. Analysis of pre-computed partition top method for range top-k queries in OLAP data cubes. Proceedings of CIKM, pp. 60–67, 2002.

[8] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao., "Efficient OLAP operations in spatial data warehouses", Proceedings of SSTD, pp. 443-449, 2001.

[9] Nikos Mamoulis, Spiridon Bakiras andPanos Kalnis, "Evaluation of Top-k OLAP Queries Using Aggregate R–Trees", Springer-Verlag LNCS 3633, pp. 236–253, 2005.

[10] F. Dehne, Q. Kong, A. Rau-Chaplin, H. Zaboli and R. Zhou, "A Distributed Tree Data Structure For Real-Time OLAP On Cloud Architectures", Proceedings of the Big Data Conference, pp. 499-505, 2013.

[11] Tyson Condie, Neil Conway, Peter Alvaro and Joseph M. Hellerstein, "Online Aggregation and Continuous Query support in MapReduce", Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 1115-1118, 2010.