ESE - 589 LEARNING SYSTEMS FOR ENGINEERING SYSTEMS
Stony Brook University

PROJECT 1

IMPLEMENTATION OF STAR CUBING ALGORITHM

Submitted to

Professor Alex Doboli
Department of Electrical and Computer Engineering

Project Members:

Kaushik G Kulkarni - 111486036
Manoj Kumar Gopala -111679371

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Data cube computation is essential task in data warehouse implementation. The precomputation of all or part of a data cube can greatly reduce the response time and enhance the performance of on-line analytical processing. In this project, we present Star- Cubing, that integrates the strengths of the previous algorithms and performs aggregations on multiple dimensions simultaneously. It utilizes a star-tree structure, extends the simultaneous aggregation methods, and enables the pruning of the group-by's that do not satisfy the iceberg condition.

INTRODUCTION

Efficient computation of data cubes has been one of the focusing points in research with numerous studies reported. The previous studies can be classified into the following categories: (1) efficient computation of full or iceberg cubes with simple or complex measures [2], (2) selective materialization of views [8], (3) computation of compressed data cubes by approximation, such as quasi-cubes, wavelet cubes, etc. [9] (4) computation of condensed, dwarf, or quotient cubes [10], and (5) computation of stream "cubes" for multi-dimensional regression analysis [11]. Among these categories, efficient computation of full or iceberg cubes, plays a key role because it is a fundamental problem, and any new method developed here may strongly influence new developments in the other categories.

Previous studies have developed two major approaches, top-down vs. bottom-up, for efficient cube computation. The former, represented by the Multi- Way Array Cube (called Multiway) algorithm, aggregates simultaneously on multiple dimensions; how- ever, it cannot take advantage of Apriori pruning when computing iceberg cubes. The latter, represented by two algorithms: BUC and H-Cubing, computes the iceberg cube bottom-up and facilitates Apriori pruning. BUC explores fast sorting and partitioning techniques; whereas H-Cubing explores a data structure, H-Tree, for shared computation. However, none of them fully explores multi-dimensional simultaneous aggregation. A new iceberg cubing algorithm, Star- Cubing, is proposed, which integrates the top-down and bottom-up cube computation and explores both multi-dimensional aggregation and the Apriori pruning. A new data structure, star-tree, is introduced that explores lossless data compression and prunes unpromising cells using an Apriori-like dynamic sub- set selection strategy. Performance study shows that Star-Cubing outperforms the previous cubing algorithms in almost all the data distributions.

PROBLEM STATEMENT :

The problem of cube computation can be defined as follows. In an n-dimension data cube, a cell $a = (a_1,a_2,...,a_n,c)$ (where c is a measure) is called an m-dimensional cell (i.e., a cell in an m-dimensional cuboid), if and only if there are exactly m ($m \leq n$) values among $\{a_1,a_2,...,a_n\}$ which are not $*$. It is called a base cell (i.e., a cell in a base cuboid) if m = n; otherwise, it is an aggregate cell. Given a base cuboid, our task is to compute an iceberg cube and simultaneously aggregate, i.e., the set of cells which satisfies an iceberg condition, or the full cube if there is no such condition.

RELATED WORK

Data cube computation has been one of the essential and most expensive operations in the field of data warehousing and online analytical processing (OLAP). The process of Data cubing can be mentioned as computation of a set of all group-by's from a base table, thereby facilitating many OLAP operations such as drill-down or roll-up. It is recognized as that data cubing will throw out a huge amount of output. Several efforts have been made to reduce the cost of these operations. They involve algorithms such as Multiway Array aggregation for full cube computations, Bottom-up computation for Iceberg cube computations, Hyper-Tree cubing. Before discussing about our algorithm, let us first review the typical cubing algorithms mentioned above, which includes both top-down and bottom-up categories.

1. **Multiway Array Aggregation** –This is a top-down approach, employed for full cube computation which uses a multidimensional array as its underlying data structure. In this algorithm, the computation starts form larger group-by's and then roll down to compute smaller group-by's. First, the large arrays are reduced into small sub-cubes called *chunks* which can fit into the memory, thus saving the amount of memory used. These chunks are further compressed to remove unnecessary chunks from the memory and further, perform simultaneous aggregation on multiple cuboids. The order of computation of the cells must be maintained in a way that minimizes the number the times that each cell is revisited. This technique involves overlapping as intermediate aggregate values are reused for computing the ancestors. The figure (1) shows a top-down approach making up a 3-D data cube with dimensions X, Y & Z.

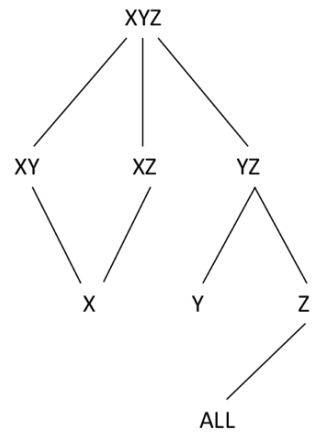


Figure 1 – Top down Approach for a 3-D data cube

This algorithm can be applied to datasets where in the product of cardinalities of the dimensions of that particular dataset is moderate and not high. If the dimensions are high, then this algorithm is unusable since the arrays and intermediate values become large enough not to fit into the memory. Further, Apriori pruning cannot be achieved using this algorithm because the computations do not possess anti-monotonic property.

2. **Bottom-Up Computation (BUC) Algorithm** –BUC employs bottom up computation techniques by expanding the dimensions. Unlike Multiway aggregation, this algorithm is employed for sparse data cube computation. Cubes with lesser dimensions are parents of those with a more dimensions. It constructs the cubes from the apex to the base cuboid which allows the BUC to share the costs of data partitioning and also to prune unnecessary computations by recurring to Apriori pruning strategy based on the anti-monotonic property, that is, if a given cell does not satisfy the Iceberg condition, then no descendant of it will satisfy the condition either. BUC is started by reading the first dimension and then partitioning it based on the distinct values. Further, for every partition, the remaining dimensions are computed in a recursive manner. The figure (2) depicts a bottom up computation approach making up a 3-D data cube with a dimensions X, Y & Z.

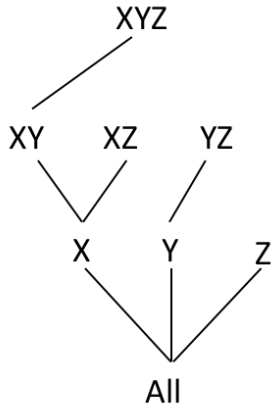


Figure 2 – Bottom Up Approach for a 3-D data cube

BUC is a *divide & conquer* type algorithm, that is, once after a partition is computed, all its descendants are computed before the algorithm switches over to the next partition. The partition process is employed using Linear sorting method called *Counting Sort* (only when cardinalities are not too large), and other times, Quick sort is used. The process of partitioning, sorting and aggregation are costly since recursive partition doesn't reduce the input size. Moreover, BUC is sensitive towards data skew and also to the order of dimensions; performance of BUC degrades as the data skew increases. To improve performance, the most differentiating dimensions must be processed first and then proceed in the order of decreasing cardinalities. Even though BUC shares partition costs, there is no sharing of computation of aggregates between a parent and child group- by's. For Example, in the figure (2), the computation of the cuboid (2) doesn't help the computation of the cuboid (2), that is, the latter has to be computed from the scratch.

3. **H-Cubing Algorithm** – This algorithm uses a hyper-tree structure to facilitate cube computations, wherein every level of the tree represents a dimension in the base cuboid. With this data structure, we can employ two methods, Bottom-up and Top-down traversal to compute the cube with the only difference that the former starts the process at the bottom of H-tree, whereas, the latter starts at the top. In both the methods, the algorithm is first started at a particular level of the tree, examining the group-by's of a particular dimension and then placing that level and the levels above it in the H-tree. Further, aggregation is

carried out and those nodes with a lesser count than the minimum support are skipped, and the algorithm jumps over to the next node via side-link. Even though, simultaneous aggregation, shared processing, Apriori pruning are observed, there is no utilization of intermediate results for computing low dimensions to facilitate the high dimension cuboid computation, and also a weak simultaneous aggregation process is observed.

STAR CUBING ALGORITHM

In this section, we discuss the Star cubing algorithm which is a mixed approach that integrates both the top-down approach (Multiway Array aggregation) and bottom-up approach (BUC) in order to compute Iceberg cubes. Star-Cubing utilizes a hype-tree like structure called 'Star-tree' for cube computation. The bottom-up approach is used on the global computation order, whereas, the sublayer underneath is based on the top-down approach that explores the notion of shared dimensions, which enables it to successfully partition parent group-by's and use the Apriori -kind pruning on the child group-by's that do not satisfy the Iceberg Condition. In general, the strengths of both the algorithms, Multiway array aggregation & BUC (Bottom-Up Computation), are combined to achieve simultaneous aggregation and Apriori pruning, at the same time. The basic approach of star cubing is as shown in figure (3) below.

Star cubing explores the concept of shared dimensions, which is defined as a set of the prefix dimensions of the parent node and the set of maximum prefix dimensions of all nodes in the subtree. For example, as shown in figure (3), a subtree which is denoted as XY/X instruments to us that cuboid XY has a shared dimension of X and further XY/X represents 2 cuboids, where XY is the upper bound of all cuboids in that particular subtree and X is a cuboid, which is the lower bound of all the cuboids in that particular subtree. The integration of top-down approach is achieved by such a node, XY/Y where XY is used for aggregation(simultaneous) and Y is utilized for pruning. The shared dimensions grow in a bottom up fashion. These shared dimensions possess an anti-monotonic property, which means that if the aggregate value on the shared dimension does not satisfy the Iceberg condition, then all those cells extending from that shared dimension, cannot satisfy that condition either. The complete intuition of this algorithm is that, if we can compute the shared dimensions before computing the actual cuboid, then we can those to achieve Apriori pruning. Star Cubing algorithm helps us to achieve pruning while still aggregating simultaneously on multiple dimensions.

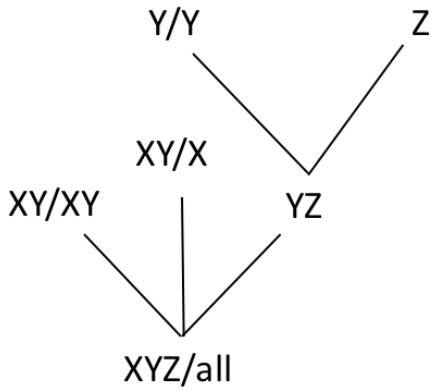


Figure 3 – Star Cubing - Top down computation with bottom up growing shared dimensions for a 3-D data cube

The basic steps of Star cubing algorithm are as follows:

- This algorithm uses a tree like data structure, similar to a H-tree to store the different cuboids, forming a base cuboid tree.
- Next, it collapses the common prefixes in order to save memory. Then, a count is kept at a node.
- The tree is traversed, to retrieve a particular tuple of values. If the single dimensional aggregates on a particular attribute value does not satisfy the Iceberg condition, then those values are replaced with a *Star* *. Such attributes are called star attributes and their corresponding nodes in the cell tree are called star nodes.
- Let us consider for a situation where the minimum support, in general, iceberg condition is asked to be greater than or equal to 2. First, 1-dimension aggregation is performed on the values in the original table. All the values less than 2 are replaced with a *, and further all the star's (*'s) are collapsed together. Now, the resulting table will have all such attributes replaced with a star (*) attribute. With regards to the iceberg computation, this new table formed is a lossless compression of the original table.
- A star tree is now constructed with the help of the compressed table also the star table, which consists of those one attributes which are now termed as starred attributes. The Star tree generated is also a lossless compression of the original cell tree.

Multiway Star Tree aggregation – With this generated star tree, the process of aggregating is started by traversing the tree in a top-down manner. Traversal algorithm used will be depth first. The process is as follows:

- Depth first search is first started at the root of the base star tree. Then, at every new node in the DFS, corresponding star trees that are the descendants of the current tree is created according to the integrated traversal ordering.
- Next, the count in the base tree are carried over to the new sub-trees. When the DFS reaches the leaf, backtracking is started. Before tracking back, the algorithm notices that all the possible nodes in shared dimensions are visited.
- On every backtracking branch, the count in the corresponding trees are given as output, further, the tree is destroyed and also the node in the base tree is destroyed.

We might come across some issues during this process like cost, time complexity etc. To encounter such issues, some techniques are employed. They are:

- Node Order - Search operation is a time-consuming operation. In order to reduce this cost, the nodes in a star tree must be sorted in alphabetical order in every dimension. Such an ordering will facilitate easy location during traversal.
- Child tree pruning – Useless child trees, which are at one level lower than the current tree must be pruned.
- A non-star node in the base tree can become a star node in the child tree. To avoid this, it is required that all the nodes are checked again during the construction of child trees.

The Flowchart of the Star-Cubing Algorithm is as show in the figure (4) below.

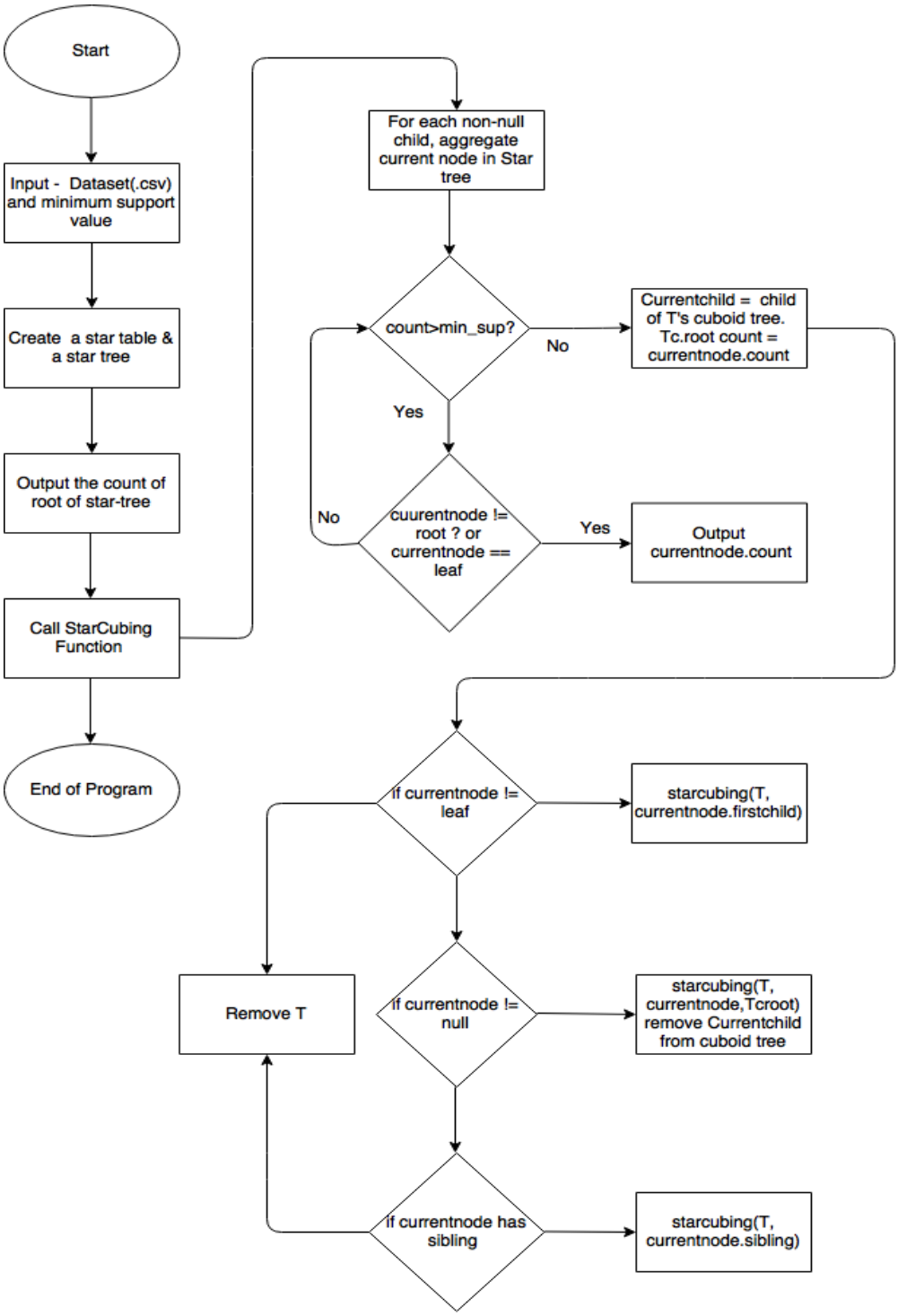


Figure 4 – Flowchart of Star Cubing Algorithm

IMPLEMENTATION & EVALUATION

We analyzed our algorithm on 2 datasets from UCI Machine Learning Repository

1) Dataset: Air Quality Data Set

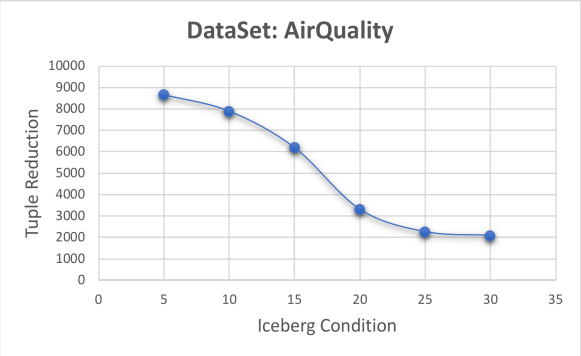
The dataset contains 9471 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensory Device. The device was located on the field in a significantly polluted area, at road level, within an Italian city. Data were recorded from March 2004 to February 2005 (one year) representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses.

Data Set Characteristics:	Multivariate,TimeSeries	**Number of Instances**	9471
Attribute Characteristics:	Real	**Number of Attributes:**	15

The dataset was run on Mac book pro with 16 GB RAM and i7 Processor.

Iceberg condition	Actual size	Reduced size	Execution time	Memory used(Bytes)
5	9471	8654	0.549854 s	6356992
10	9471	7887	0.355509 s	6070272
15	9471	6187	0.266486 s	5541888
20	9471	3309	0.289535 s	4726784
25	9471	2263	0.240005 s	4579328
30	9471	2087	0.236022 s	4420288

Table 1: Dataset Airquality : Execution parameters



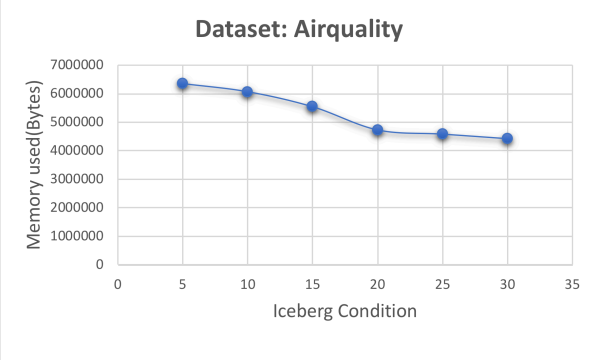


Figure 5: Tuple Reduction vs Iceberg Condition(Airquality) *Figure 6: Memory used vs Iceberg Condition (Airquality)*

2) Dataset: Page Blocks Classification

The 5473 examples comes from 54 distinct documents. Each observation concerns one block. All attributes are numeric. Data are in a format readable by C4.5.

Data Set Characteristics:	Multivariate	**Number of Instances:**	5473
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	10

The dataset was run on Mac Book pro with 16 GB RAM and i7 Processor.

Iceberg condition	Actual size	Reduced size	Execution time	Memory used (Bytes)
5	4926	4368	0.291783 s	6242304
10	4926	2641	0.283041 s	5517312
15	4926	1456	0.270488 s	5238784
20	4926	699	0.256029 s	4972544

Table 2: Dataset Page Blocks Classification: Execution parameters

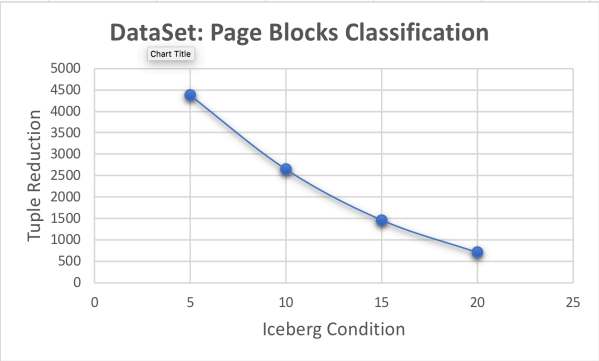


Figure 7: Tuple Reduction vs Iceberg Condition (page Blocks Classification)

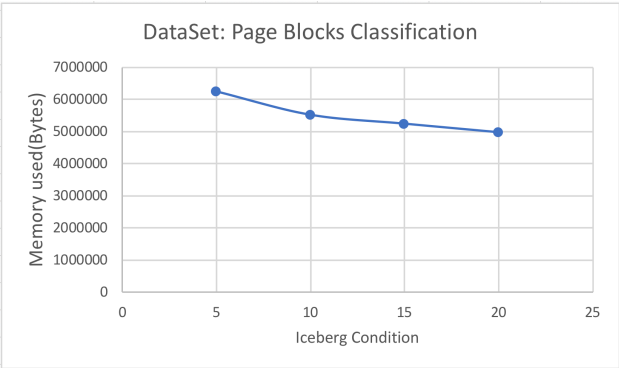


Figure 8: Memory used vs Iceberg Condition (page Blocks Classification)

DISCUSSION:

The paper, ***"Linear Programming Based Optimization for Robust Data Modelling in a Distributed Sensing Platform"*** discusses a procedure to construct data models with high robustness by using samples that are acquired from a grid network of embedded sensors with limited resources like bandwidth & buffer memory. The data models are in the form of Ordinary Differential Equations in this paper. The local data models are constructed using lumping state variable, and further, the local models are collected centrally to produce a global data models. The basic purpose of lumping is to reduce communication traffic by combing some of the less important traffic data, thereby minimizing errors due to data loss & time delays. The errors describe the types of inaccuracies that occur during a data model construction using distributed embedded sensing networks. Further, using the error bounds, the lumping levels and the parameters of the networked sensing platform are computed, which further are used to set threshold values by local schemes at the embedded sensing nodes to decide modelling actions. Further, aggregation methods are implemented in a specific path manner, node to node, focused on reducing data traffic in order to lower the power and energy consumption, also avoiding transmission of redundant data.

For this paper, the star cubing algorithm can be modified and implemented in the following way:

1) An iceberg condition must be defined for those variables which are less important. The base cuboid tree consisting of all the sensor nodes must be traversed to determine all the lumping variables which do not meet the iceberg condition are removed from the cuboid tree and put into a temporary array. After completion of the traversal, the temporary array consisting of all the lumped values, is sorted. Further, the median element of the array is taken out and it is put as the star node in the star-tree, while star-tree construction. In general, the median value is now in place of the star node, in a star tree. Using a temporary array and sorting it increases the space and time complexity of the algorithm.

2) The aggregation part of the algorithm will more or less remain the same. As the values are being pruned, as in this case, lumped, for a particular cuboid, those values passing the iceberg condition will be aggregated according to a predefined path.

SOFTWARE IMPLEMENTATION:

The Algorithm was implemented in C++ programming language. The main functions which were used in our implementation are

Function	Description	Inputs	outputs
Check_icebergCondition	Checks if the iceberg condition is satisfied or not if satisfied keep the same value if not change the value by *value	CSV file data, attribute index, Attribute frequency, star representation, iceberg condition	Void function (changes the value in the base table)
create_compressed_basetable	Will eliminate the redundant rows and provide the compressed table	Base table	Compressed table

Table 3: Software implementation

Data Structures used:

Maps:

Maps are used to replicate associative arrays. Maps contain sorted **key-value** pair, in which each key is unique and cannot be changed, and it can be inserted or deleted but cannot be altered. Value associated with keys can be altered. We can search, remove and insert in a map within O(n) time complexity.

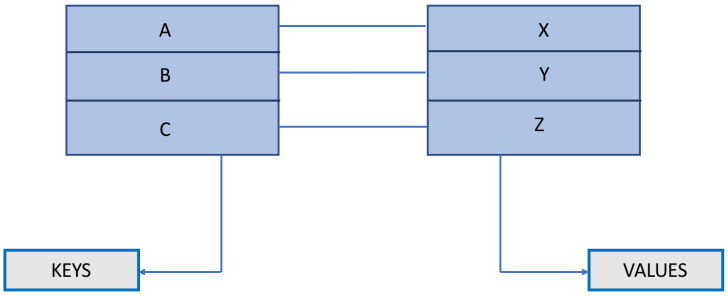


Figure 9: Map Flow

CONCLUSION

Star cubing algorithm provides an efficient, high performance cube computation strategy for different types of data distributions by integrating both the top-down and the bottom-up cube computation approach. Two major optimization techniques that are worth to be mentioned of are: 1) The concept of shared dimensions and shared aggregation; 2) Apriori pruning. No previous algorithm has fully explored both the optimization methods in one single algorithm, thus saving the memory used and thereby, reducing the cost of operation. The star cubing algorithm implemented proves to be faster than BUC and H-cubing when it comes to full cube computation and in case of sparse cube, it proves to be faster than all three previous algorithms mentioned. In case of iceberg computation, Star cubing is faster than BUC and also the speedup is higher when the minimum support value decreases. Thus, Star cubing algorithm provides high performance in various types of data distributions.

BIBLIOGRAPHY

[1] Star-Cubing: Computation of Iceberg cubes by top-down and bottom-up integration. Dong Xin, Jiawei Han, Xiaolei Li, Benjamin W Wah, UIUC, Urbana, IL.

[2] S. Agarwal, R. Agreal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. VLDB'96.

[3] Data Cubing Algorithms – Comparative study Data Cubing Algorithms, Beena Mahar; Proceedings of 3rd national conference, India Com- 2009.

[4] C-cubing: Efficient Computation of closed cubes by Aggregation-Based Checking. Dong Xin, Zheng Shao, Jiawei Han, Hongyan Liu. UIUC, Urbana, IL.

[5] Different Cube Computation Approaches: Survey Paper; Dhanashri S Lad, Rasika P Saste; IJCSIT, Vol. 5 (3), 2014.

[6] Linear Programming – Based Optimization for Robust Data Modelling in a distributed sensing platform. Anurag Umbarkar, Varun Subramanian, Alex Doboli, IEEE Transactions on Computer-Aided Design of Integrated Circuits & systems, Vol. 33, No.10. Oct 2014.

[7] MM Cubing: computing Iceberg cubes by factorizing the lattice space. Zheng Shao, Jiawei Han, Dong Xin, 16th international conference on scientific & Statistical database.

[8]E. Baralis, S. Paraboschi, and E. Teniente. Materi- alized view selection in a multidimensional database. VLDB'97, 98-12.

[9] D. Barbara and M. Sullivan. Quasi-cubes: Exploit- ing approximation in multidimensional database. SIG- MOD Record, 26:12-17, 1997.

[10] W. Wang, J. Feng, H. Lu and J. X. Yu. Condensed Cube: An Effective Approach to Reducing Data Cube Size. ICDE'02.

[11]Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang, Multi-Dimensional Regression Analysis of Time-Series Data Streams. VLDB'02.