Journal of Computer Sciences

Original Research Paper

Tracking Pointer and Look Ahead Matching Strategy to Evaluate Iceberg Driven Query

Kale Sarika Prakash and P.M. Joe Prathap

Department of Computer Science and Engineering, St. Peter's University,
St. Peter's Institute of Higher Education and Research, Avadi, Chennai, India

Article history
Received: 30-12-2016
Revised: 14-02-2017
Accepted: 17-04-2017

Corresponding Author:
Kale Sarika Prakash
Department of Computer Science and Engineering, St. Peter's University, St. Peter's Institute of Higher Education and Research, Avadi, Chennai, India
Email: kalesarikaprakash@gmail.com

Abstract: Iceberg driven query is important and common in many applications of data mining and data warehousing. Main property of iceberg driven query is it extracts small set of data from huge database. It works on aggregation function followed by GROUP BY and HAVING clause. Due to involvement of aggregation function execution of iceberg driven query becomes tedious and complex work. Main objective of this research is to improve the performance of iceberg driven query by reducing the time, number of iteration and I/O access required to execute it. Currently available iceberg driven query processing technique faces the problems of empty bitwise AND, OR and XOR operation. Because of these problems they require more time and I/O access to execute query. To overcome above problems this research proposes tracking pointer and look ahead matching strategy to evaluate iceberg driven query. Tracking pointer will initiate the evaluation process as per the priority of vector. Look ahead matching strategy help to identify probable vector instead of generating one by one till the end of vector list. This strategy decides the probability of bitmap vector to be executed. Thus in advance it identifies and avoids unnecessary operations to be performed on bitmap vector. Our experimental result shows that time and number of iteration required to evaluate iceberg driven query using proposed approach is reduced [40 to 50] % even though data size increases. Thus we prove the effectiveness and efficiency of proposed approach to process iceberg driven query.

Keywords: Iceberg Driven Query (IDQ), Tracking Pointer (TP) Strategy, Look Ahead Matching (LAM) Strategy

Introduction

Data warehouse is collection of subject oriented, integrated, non volatile and time variant dataset as described by Inmon (2005). Analysis of such huge database is done by executing complex queries such as IDQ and online analytical processing functions. The basic operation required in data analysis is aggregate functions such as MIN, MAX, SUM, AVG and COUNT. Generally the queries to be executed on data warehouse are the queries with aggregation function followed by HAVING and GROUP By clause, such a query is known as IDQ. It consists of three main parameters such as aggregation function, HAVING clause and GROUP BY clause which makes the query more complex.

Dubey *et al*. (2014) reported that, in addition to the complexity of IDQ, the large volume of data stored in data warehouse lengthens the time needed to execute queries. Hence performance of query in terms of time is most important requirement of any large database system. This research focus on efficient execution of aggregate function as it is a main part of IDQ.

Number of researches He *et al*. (2011; Guru Rao and Shankar, 2012; 2013; Shankar and Guru Rao, 2014; Rao, 2014) work to improve performance of IDQ. But all of them faces the problem of empty bitwise AND operation, XOR operation, OR operation and futile queue pushing.

Proposed research overcomes these problems by using Tracking Pointer (TP) and Look Ahead

Matching (LAM) Strategies. TP strategy work on priority based approach which first analyse the operation to be perform as per the evaluation of query. According to evaluation process it arranges the sequence of operation to be performed. Based on results of current operation it change the priority and perform the remaining operation. Along with TP strategy this research proposes use of LAM strategy. Main task of LAM strategy is to identify useless operation in advance and skip such operations. It uses the concept of probability to predict the chances of bitmap vector to be part of final query result. In this way by performing only required operations it reduces I/O access, iterations as well as time required to execute IDQ. These strategies work on bitmap vector of attribute as per query requirement. The bitmap vectors are in the form of 0's and 1's and our algorithm perform logical operations such as OR and XOR on this bitmap vectors. Executing bitwise operations on 0's and 1's are very much cost effective in term of I/O access and time. In this research it directly helps us to improve the performance of IDQ. Our experimental result proves that performance of our strategy is better than previous algorithms.

This paper is organized into following sections. Section 2 describes review of aggregate function, BI and IDQ. Section 3 focuses on TP and LAM strategy to evaluate IBDQ its workflow diagram and pseudo code. Experimental analysis with graphical representation is described in section 4 and section 5 conclude the paper.

Review of Aggregate Function, Bitmap Indexing (BI) and Iceberg Driven Query (IDQ)

This section highlights the concepts such as aggregate function, bitmap indexing and iceberg driven query. We are using these concepts in our research.

Aggregate Function

Dubey *et al*. (2014; Fang *et al*., 1998) stated that aggregation function across many attributes are commonly used in queries of data mining, data warehousing and online analytical processing. The commonly used queries in data mining and data warehouse are IDQ which perform an aggregate function across attributes and then remove aggregate values that are below some specified threshold value. Generally used aggregation functions are MIN, MAX, SUM, AVG and COUNT.

As stated in research article by Gray *et al*. (1997) there are three different type of aggregate functions. Distributive: An aggregate function F is distributive if there is a function G such that $F(T) = G(\{F(S_i)|i = 1 \ldots n\})$. SUM, MIN and MAX are distributive with $G = F$. Count is distributive with G = SUM.

Algebraic: An aggregate function F is algebraic if there is an M-tuple valued function G and a function H such that $F(T) = H(\{G(S_i)|i = 1 \ldots n\})$. Average, Standard Deviation, MaxN, MinN and Center_of_Mass are all algebraic. For Average, the function G records the sum and count. The H function adds these two components and then divide to produce the global average. Similar technique is apply to find the N largest value, the center mass of group objects and other algebraic functions. The key to algebraic functions is that a fixed size result (an M-tuple) can summarize the sub-aggregation.

Holistic: An aggregate function F is holistic if there is no constant bound on the size of the storage needed to describe a sub-aggregate. That is, there is no constant M, such that an M-tuple characterizes the computation F.

Efficient computation of all these aggregate functions are required in most of the large database applications because processing cost of aggregate function is much higher than that of the other basic relational operations like SELECT and PROJECT.

Bitmap Indexing (BI)

Mei *et al*. (2013) stated that BI technique is most suitable and efficient for read mostly, append only and large size dataset. Because of this feature of BI we are using it in our research.

Jrgens (1999) reported that BI strategy perform better than tree based indexing methods like B Tree and R Tree. In White Paper (2015; 2011) they mentioned that BI has three advantages for using it in data warehouse that it avoids complete table scan, save number of disk access and save computational time. Compressed BI concept is stated by Deliège and Pedersen (2010; He *et al*., 2011) which is appropriate for our research. Our research makes use of this concept which saves the memory and shows the effectiveness of BI for evaluation of IDQ. BI performs effectively as it works on index level rather on original table. This feature help to improve performance in terms of time required to access data from database and memory required to store database. By considering all above features of BI we are using it in our research. We are extending the way of using BI by TP and LAM strategy and improving the performance of IDQ.

Overview of Iceberg Driven Query (IDQ) and its Processing Methods

IDQ refer to a class of queries which compute aggregate function across attributes to find aggregate

value above some specified threshold value. The number of tuples that satisfy the threshold in the having clause is relatively small compared to the large amount of input data. The output result can be seen at the tip of iceberg. Main property of IDQ is it extracts small set of data from huge database. As it extracts small set of data, the time required for extracting such a small data set must be less even though it works on huge database. Our research make use of this property to evaluate IDQ.

Given a relation R with attributes a_1, a_2… a_n, aggregate function AggF and a threshold T, the structure of IDQ is as follow:

SELECT R.a_1, R.a_2… R.a_n, AggF (R.a_1, R.a_2… R.a_n)
FROM relation R
GROUPBY R.a_1, R.a_2… R.a_n
HAVING AggF (R.m) >= T

Suppose, a purchase manager is given a sales transaction dataset. He or she may want to know location wise total number of Products which satisfy threshold condition. To answer this, we can write iceberg driven query as below:

SELECT location, Product Type, Sum (# Product)
FROM Relation Sales
GROUP BY Location, Product Type
HAVING Sum (# Product) >= T

To implement Iceberg driven query, a common strategy in horizontal database is first to apply hashing or sorting to all the data in the dataset, then count all of the location and Product Type pair groups and finally eliminate those groups which do not pass the threshold T. These algorithms generate significant I/O for intermediate results and require large amounts of main memory. They leave much room for improvement in efficiency. Another option to retrieve above data is instead of counting the number of tuples in every location and Product Type pair group at first step, we can generate Location-list: A list of local stores which sell more than T number of products using following query:

SELECT Location, Sum (# Product)
FROM Relation Sales
GROUPBY Location
HAVING Sum (# Product) >= T

Second step we can generate Product Type-list: A list of categories which sell more than T number of products. For example:

SELECT Type, Sum (# Product)
FROM Relation Sales
GROUPBY Product Type
HAVING Sum (# Product) >= T

In this way we can eliminate many of the location and Product Type pair groups. It means that we only generate candidate location and Product Type pairs for local store and Product type which are in Location-list and Product Type-list.

In this research we are making use of the main feature of IDQ that it can answer quickly because of small result set from large database. But current database systems do not take full advantage of this feature. The relational database systems like Oracle, SQL, MYSQL and DB2 uses general aggregation algorithm to answer the iceberg driven query by aggregating all tuples first then evaluating HAVING clause to generate query result. This method require multiple passes of database to generate result which directly affect on the performance of query in terms of time, I/O access and memory requirement.

The concept of Iceberg query was first studied by Fang *et al*. (1998). In this research researchers extend probabilistic technique suggested by Whang *et al*. (1990) and proposes hybrid and multi bucket algorithm. This research combine sampling and multi hash functions to improve the performance of iceberg query and reduce memory requirement. But these algorithms are not suitable for large data sets.

To solve above problem Fang *et al*. (1998) proposes algorithm based on sampling and bucket counting method. This method generates false positive and false negative values which are in the final result but it is not in the constraint list. Focus of this research is to minimize false positive value. Different optimization methods like hashing, multiple hashing and combination of multiple hash functions are used in these algorithms. These methods reduces number of false positive values but it take more time to execute query as it require multiple scan of relation.

To overcome the problem occurred in above research Bae and Lee (2000) introduces method to select candidate values using partitioning and postpone partitioning algorithms. This overcome the problem of multiple scan over relation occurs in sampling and bucket counting mechanism. The performance of these algorithm depends upon the order of data and memory size. If database is sorted then performance is good without considering memory size.

Collective Iceberg query evaluation is proposed by Leela *et al*. (2004) which presents comparison using three methods sort merge aggregate, hybrid hash aggregate and ORACLE. This study proves that

performance of sort merge aggregate is better on data sets with low to moderate number. Hybrid hash aggregate performance was not good when data set is large. There was a considerable performance gap between the online algorithms and ORACLE.

All above mentioned algorithms of iceberg query processing come under the group of tuple scan based, which require multiple table scan to read data from disk. This way of processing iceberg query is time consuming. All above algorithms focuses on reducing number of tuple scan but no one of them uses property of iceberg query.

However Ferro *et al*. (2009) tries to make use of this property of iceberg query and uses BI but it suffers from empty bit wise AND result problem. Researchers He *et al*. (2011) tries to minimize this problem using dynamic pruning and vector alignment algorithm. This work leverages the antimonotone property of iceberg query and develop dynamic pruning algorithm using BI. However they notice that there is problem of massively empty bitwise AND results and extra XOR operation. To overcome this challenge they develop vector alignment algorithm which help to solve empty bitwise AND operation problem. The problem with this algorithm is that all vectors may not have 1 bit at same position and if it is not at same position then all the AND as well as XOR operations are fruitless and time consuming.In this way both the above approaches suffer from fruitless AND as well as XOR operations. Guru Rao and Shankar (2012) try to handle empty XOR operation problem but did not able to solve fruitless bit wise AND operation problem. Both the research He *et al*. (2011; Guru Rao and Shankar, 2012) faces the problem of futile queue pushing and empty bitwise operation.

Our research overcomes these problems by using TP and LAM strategy. This approach improves efficiency by pruning many groups beforehand. In our strategy the main operations are bitwise AND, OR and XOR which are perform on bitmap vector. Bitmap vector is in the form of 1's and 0's, so these operations can be executed quickly by hardware. The execution cost of these operations is really cheap. As we can see in our experimental result section, our methods and procedure are superior in computing IDQ. In this way this research provide a framework for evaluating IDQ with aggregate functions like MIN, MAX, COUNT and SUM.

Proposed TP and LAM Strategy for IDQ processing

Working Model of TP and LAM Strategy

This section describes the workflow of TP and LAM strategy for IDQ evaluation. Figure 1 shows workflow of TP and LAM strategy.

Initially, bitmap vector is generated. Then TP strategy uses priority based approach to assign priorities to vector. TP assign priority to vectors as per the position of 1's occurring in vector. After finalization of vectors for performing bitwise AND operation LAM strategy will get activate. It help to find out probability of vector whether it will satisfy threshold condition or not. If it recognize that possibility of success is less then it will skip further AND, OR and XOR operation. Then it will move to next probable bitmap vector for further processing. Thus it help to reduce unnecessary burden of performing fruitless bitwise AND, OR and XOR operation. In this way our strategy reduces the empty bitwise AND, XOR and OR operation problem which occur in previous research. Finally the combination of vectors which satisfy threshold condition will be added in RESULT.

Methods used in TP and LAM Strategy for IDQ Processing

This section describe the method used in this research to process the IDQ. Here we are considering an IDQ with two attribute and COUNT aggregate function. The structure of query to be process is as shown in Fig. 2 and 3. This section describe detail processing of query 1 on the table T shown in Fig. 4.

If IDQ given in Fig. 2 is executed on relational database T as shown in Fig. 4 using normal BI strategy to declare the result of IDQ following steps are required:

Step 1: Perform (X AND Y) bitwise AND operation between X and Y Bitmap Vector. In our example as shown in Fig. 4 X and Y vector contain 3 distinct values like (X1,X2,X3) and (Y1,Y2,Y3) therefore to process query1 9(3*3) bitwise AND operations are required. The pair of operations to be perform are (X1,Y1),(X1,Y2),(X1,Y3),(X2,Y1),(X2,Y2),(X2,Y3),(X3,Y1),(X3,Y2),(X3,Y3).

Step 2: Next step is comparing result with the threshold value specified in query. In our example query threshold value is >= 3. The result of each AND operation is compared with threshold value >= 3. The result which satisfy this condition will be included in final IDQ result.

In this way comparison step has to execute 9 times. The final IDQ result contain only two combination (X2,Y3) and (X3,Y2). But in this approach we have to perform bitwise AND operation 9 times, comparing results with threshold also 9 times. If database size increases then the number of fruitless AND operations also go on increase which degrades the performance of IDQ. This is the major limitation of all previous research.

DOI: 10.3844/jcssp.2017.55.67

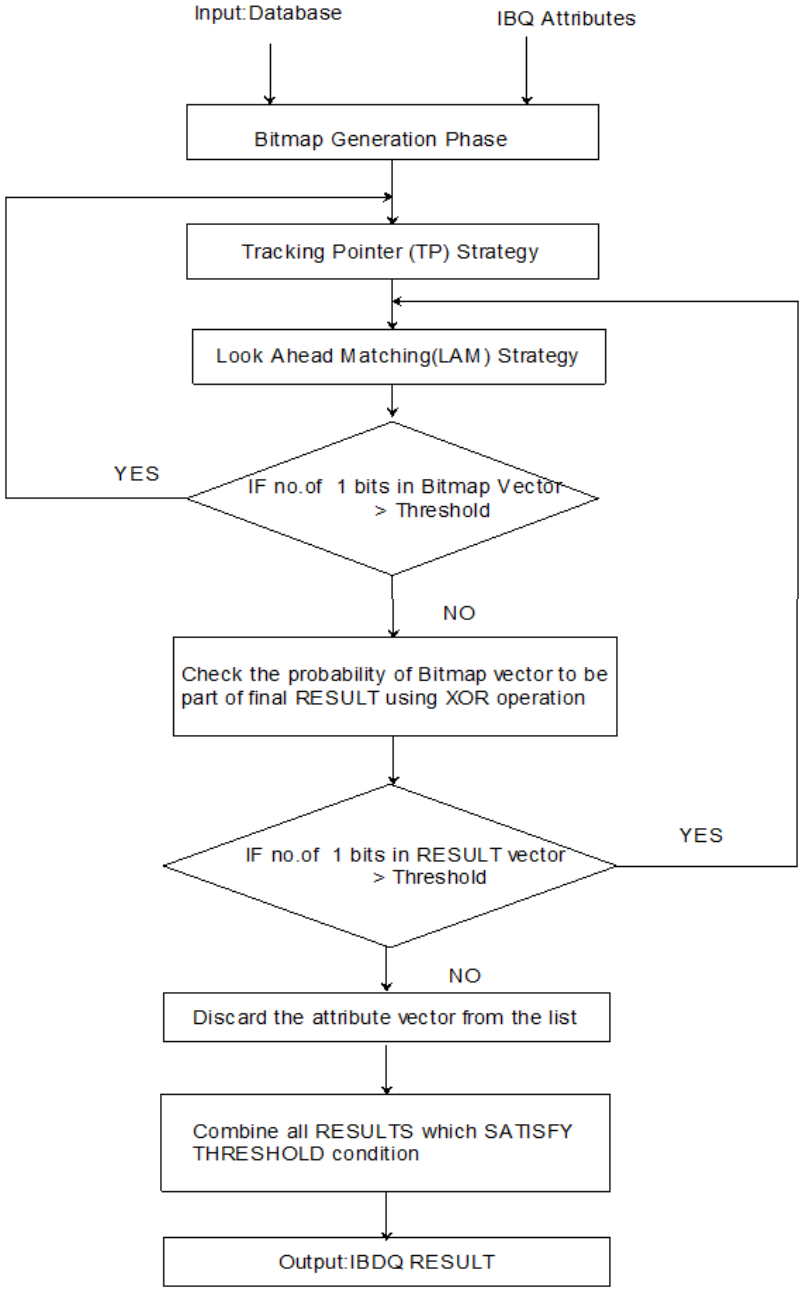

Fig. 1. Workflow diagram

```
SELECT  X,Y, COUNT(*) FROM T GROUP BY X,Y HAVING
COUNT(*)>= 3;
```

Fig. 2. Query1: IDQ with COUNT function

```
SELECT  A1, A2,COUNT(*) FROM  R GROUP BY A1, A2
HAVING COUNT(*)>2;
```

Fig. 3. Query2: IDQ with COUNT function

X	Y	Z
X3	Y2	1.00
X1	Y1	4.0
X2	Y3	6.12
X3	Y3	21.2
X3	Y2	2.5
X2	Y3	2.5
X2	Y3	2.21
X1	Y3	15.1
X3	Y2	12.2
X3	Y3	21.6
X3	Y2	15.7
X2	Y1	31.8
X2	Y1	11.1
X3	Y2	21.0

X1	X2	X3
0	0	1
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0
1	0	0
0	0	1
0	0	1
0	0	1
0	1	0
0	1	0
0	0	1

Y1	Y2	Y3
0	1	0
1	0	0
0	0	1
0	0	1
0	1	0
0	0	1
0	0	1
0	0	1
0	1	0
0	0	1
0	1	0
1	0	0
1	0	0
0	1	0

Fig. 4. Table T and its bitmap index

Our research overcomes these problems by making use TP and LAM strategy. It will work in following way:

- First it will decide which attribute vectors to be considered for operation. For this it allocates priority to vector as per first 1 bit position. In case of above example first operation is (X3,Y2) and the output is (10001000101001) it contain 1 bit more than threshold value. So (X3,Y2) is added in final IDQ RESULT
- To check further the probability of X3 and Y2 to be part of IDQ result this strategy perform following operations:

  Generate New X3 and Y2. New X3 = (X3 AND Y2) – X3 and New Y2 = (X3 AND Y2)-Y2. New X3 is (00010000010000) number of 1's does not satisfy threshold condition so vector X3 is removed from list to perform operation in combinations with vector from Y. Similarly Generate New Y2 as per above procedure and it will be (00000000000000) it does not satisfy threshold condition therefore it is also discarded from the list
  In this way through this strategy within first AND operation we discarded two vectors from the bitmap vector list. Now only 4 vectors X1,X2,Y1 and Y3 are present in bitmap vector

- Repeat step 1 on remaining vectors(X1, X2) and (Y1,Y3). Next operation is (X1ANDY1) whose result is (010000000000000) which does not satisfy threshold condition so it will be not in final IDQ RESULT. Now X1 and Y1 is also removed from the list
- Repeat step 1 on remaining vectors(X2) and (Y3). Perform (X2 AND Y3) whose result is (00100110000000) it contain 1 bit more than threshold so add (X2,Y3) in final IDQ RESULT

In this way this strategy require only 3 AND operations and it directly skip fruitless AND, OR and XOR operations. Due to this the computational cost of IDQ in terms of number of iterations required to execute the query get reduce so query processing time is reduced. In the similar fashion we have applied our strategy to solve query 2 shown in Fig. 3. We noticed that the number of AND and XOR operations required to evaluate above queries using our strategy get reduces due to this time required to execute IDQ also get reduced.

Implementation Detail of TP and LAM Strategy for IDQ Processing

This subsection of paper represents the different functions required to implement the methodology specified in last subsection. The pseudo code for TP and LAM strategy for IDQ processing is as below.

Input: (Iceberg driven Query(Attribute X, Attribute Y, threshold T), Table P, Bitmap Vector table of P)
Processing: Processing of algorithm is based on number of distinct values of IDQ attribute and threshold T
Output: (IDQ RESULT)
A] CREATE BITMAP VECTOR FUNCTION

It contains main functions which are used to convert INPUT into OUTPUT. First function is Create BITMAP VECTOR on IDQ attribute. It works on following formula:

> BITMAP VECTOR = [cardinality of Column A+ cardinality of Column B+…….+ cardinality of Column N]* No. of Rows present in Database

Above formula is used to find the Space Complexity of Algorithm. Relationship between each cardinality vector is one to one. The attribute which has this relationship is SET to 1 otherwise 0. In this way complete BITMAP VECTOR is created which is combination of 0's and 1's.

B] TP and LAM based strategy for IDQ evaluation

1. For each bitmap vector of Attribute X COUNT(Number of 1's in each Bitmap vector) if it is > T then only keep such vector in BI. Otherwise discard it from the list. For each bitmap vector of Attribute X find first 1 bit position and accordingly allocate priority.
PriorityQueueX.clear, PriorityQueueY.clear.
For each vector x of attribute X do
If(x.count>= T)then x.next1 = FirstOneBitPosition(x,0)
2. For each bitmap vector of Attribute Y
COUNT(Number of 1's in each Bitmap vector) if it is > T then only keep such vector in BI otherwise discard it from the list.
PriorityQueueX.clear, PriorityQueueY.clear. For each vector y of attribute Y do
If(y.count>=T)then y.next1 = FirstOneBitPosition(y,0)
3. Find first 1 bit position of vector X and Y and accordingly allocate Priority.
If (X.Positionof1Bit > Y. Positionof1Bit)
Then (FirstPriority == X.vector)
Else (FirstPriority == Y.vector)
4. If (X.Positionof1Bit == Y. Positionof1Bit)
Then (FirstPriority == X.vector) as X vector appears first in sequence and Y comes later.
5.PriorityQueueX.Push(x)
6. PriorityQueueY.Push(y)
7. Initiate Tracking pointer strategy: It will check the position of 1's bit in bitmap vector and map it with other attributes 1 bit position. NextMatchVector function will get initiate during this phase.
x,y = NextMatchVector(PriorityQueueX.clear, PriorityQueueY,T)
While x! = NULL &y!=NULL do
PriorityQueueX.Pop
PriorityQueueY.Pop
CurrentResult = BitwiseAND(x,y)
If(CurrentResult.count>= T) then
Add IDQ Result in RESULT(x.value,y.value,CurrentResult.count)
x.count = x.count-CurrentResult.count
y.count = y.count-CurrentResult.count
If x.count>= T then
x.next1 = FirstOneBitPosition(x,x.next+1)
If x.next1! = NULL then
PriorityQueueX.Push(x)
If y.count>= T then
y.next1 = FirstOneBitPosition(y,y.next+1)
If y.next1! = NULL then
PriorityQueueY.Push(y)
Repeat step 7 till allocating priority to vectoe.
x,y = NextMatchVector(PriorityQueueX, PriorityQueueY,T)
8. Initiate Look ahead matching strategy.
If RESULT satisfies THRESHOLD condition then to predict the possibility of positive result look ahead matching strategy is used. This help to reduce fruitless AND, OR and XOR operation. It prune the vector as it identify that this vector will not able to produce positive result. In this way this module skip further operational overhead of IDQ processing.
9. GENERATE new vectors by performing OR operation between RESULT and the new vector which is already part of RESULT.
New X Vector = Old X vector- Current Result Vector
New Y Vector = Old Y vector- Current Result Vector
10. If (New X OR Y Vector) satisfy Threshold condition then perform step 7 on newly generated vector otherwise skip the respective attribute from the vector list.
This step helps to identify the possibility of vector to be part of IDQ RESULT further.
11. Repeat step 7-10 till the vector list will be empty.
12. Final IDQ RESULT is generated.

Above algorithms are implemented using JAVA 7.0 platform with ORACLE 10 g as backend database. The experiment is performed on corei 3 processor with 4 GB DDR-III RAM. Experimentation is done on synthetic database with tuple size of 5, 10, 20, 40, 50 and 80 K. We have executed IDQ's with different aggregate functions like CONNT, SUM, MIN and MAX.

Experimental Analysis

This section describes the result of experiment conducted on TP and LAM strategy for IDQ evaluation and previous strategies like Bitmap Indexing Approach (BIA) and dynamic pruning approach. Parameters consider for comparison and to measure the performance of IDQ are database size, threshold value, number of iterations required to execute query, time and aggregate functions. As we have seen in section 3 that the number of AND, OR and XOR operation required to execute IDQ get

reduced in case of TP and LAM approach. This is reflected in our actual result obtained from this framework. We observe that number of iterations as well time required to execute query get reduced.

The graphical illustration is shown in Fig. 5 to 14 for COUNT, SUM, MIN and MAX aggregate functions. We have compare the performance of TP and LAM approach with the BIA and DPA suggested in previous work He *et al*. (2011; Guru Rao and Shankar, 2012; 2013). We observe that as we go on increasing size of data set and threshold value then also query performance is goes on increasing which is as shown in Fig. 5 to 8. Practically the performance of query is also depend on the nature of data present in database that is, if data is uniformly distributed in database then it may take more time.

The main feature of IDQ is it extracts small data from huge dataset. As data to be extracted is small so time required for extracting it must be less. But with previous approaches we noticed that as data size increases the time required to extract data is also increases. Based on our experimental result we have proved that through our approach even though data size increases then also IDQ response time get reduced proportionally. We are using bitmap indexing technique which help to handle huge data effectively as describe in Jrgens (1999; White Paper, 2015; 2011). This is also noticed through our experimentation as data size is go on increasing the percentage of response time is reduced. As shown in Fig. 6, 8 and 10 which represent time analysis we observe that for small data set size i.e., 5, 10 k and up to 20 k difference in time required is reduced only 10-20% but as we go on increasing dataset size from 20, 40 to 80 k difference in time required is reduced to 45-50%. This indicates that our strategy is well suitable for large data set. Through our experimental result we have proved that TP and LAM based approach using BI is helpful to improve the performance of IDQ. In this way we have developed the frame work for COUNT, SUM, MIN and MAX aggregate function used in IDQ.

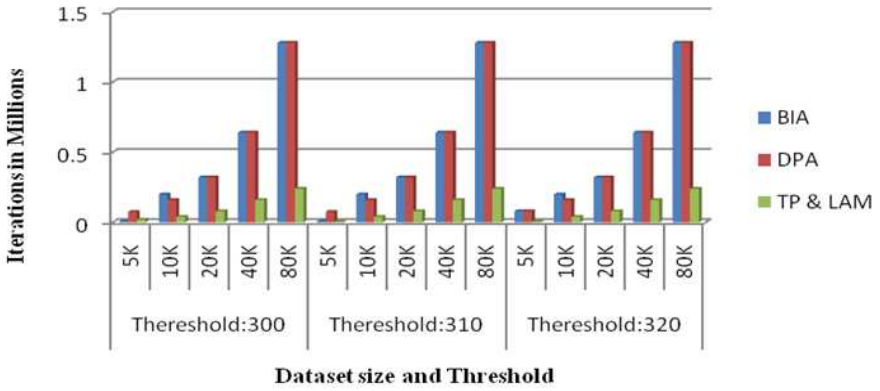

Fig. 5. Iteration Analysis of COUNT function based on Dataset size and Threshold

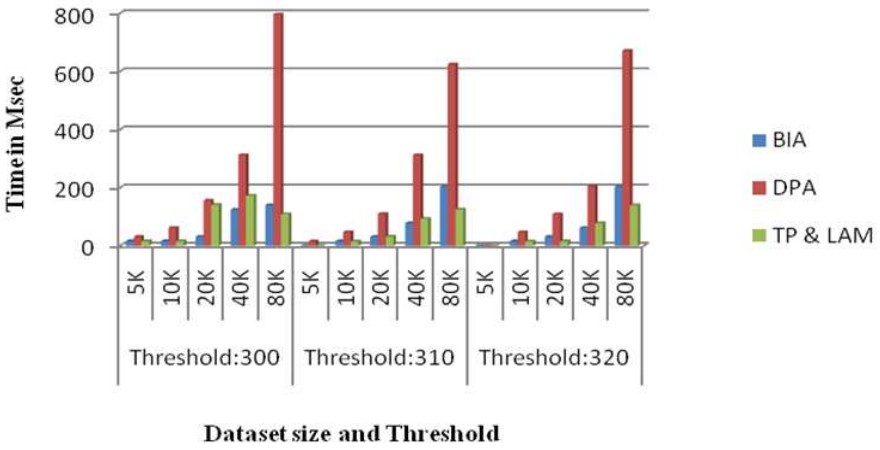

Fig. 6. Time Analysis of COUNT function based on Dataset size and Threshold

DOI: 10.3844/jcssp.2017.55.67

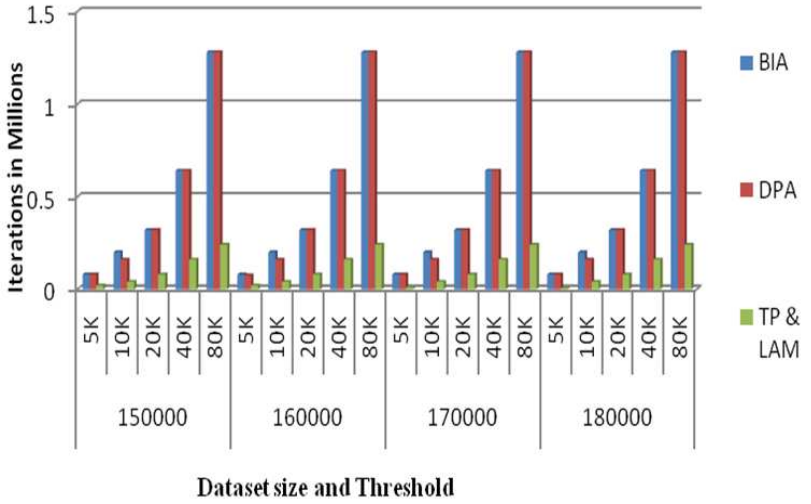

Fig. 7. Iteration analysis of SUM function based on dataset size and threshold

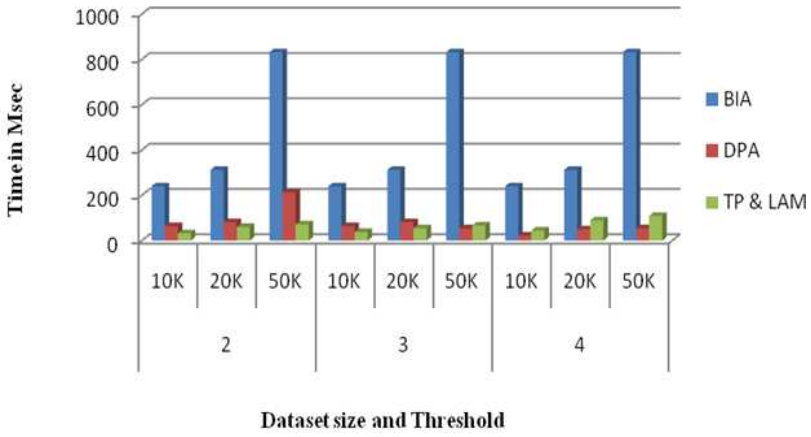

Fig. 8. Time analysis of SUM function based on dataset size and threshold

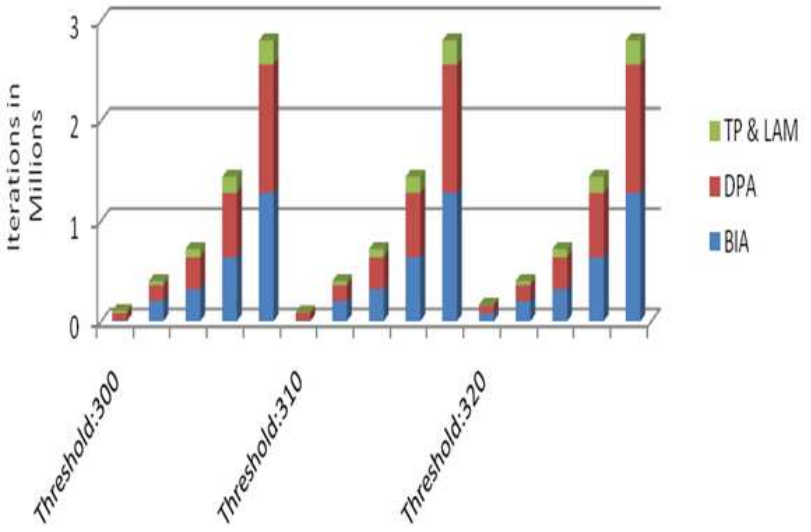

Fig. 9. Iteration analysis of COUNT function based on threshold

DOI: 10.3844/jcssp.2017.55.67

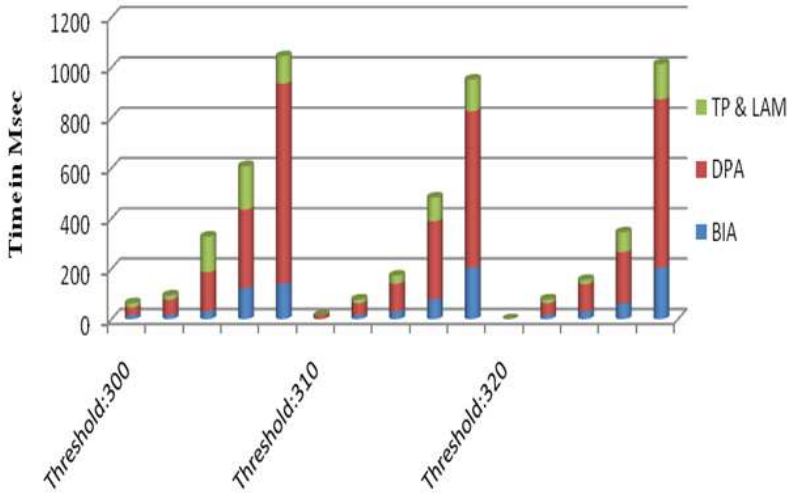

Fig. 10. Time analysis of COUNT function based on threshold

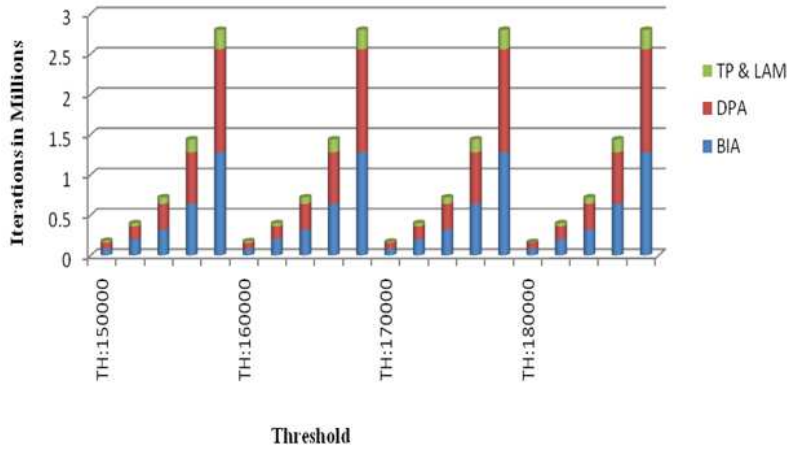

Fig. 11. Iteration analysis of SUM function based on threshold

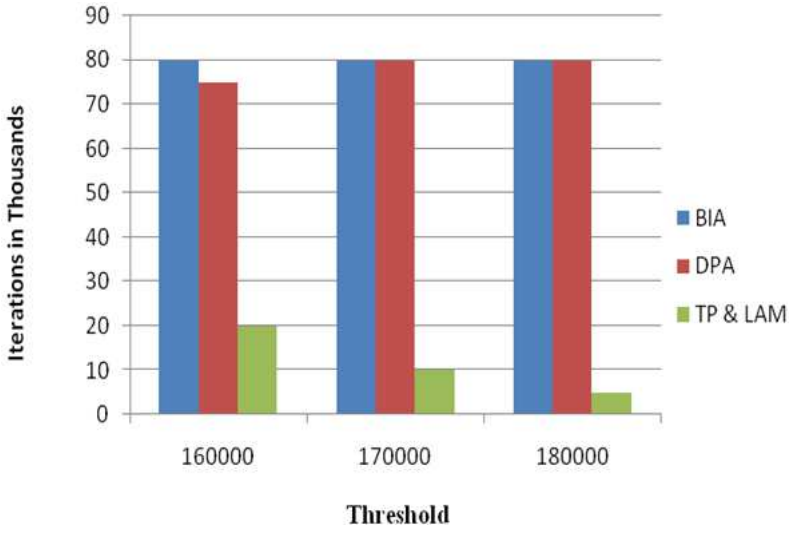

Fig. 12. Iteration analysis of MIN function based on threshold

DOI: 10.3844/jcssp.2017.55.67

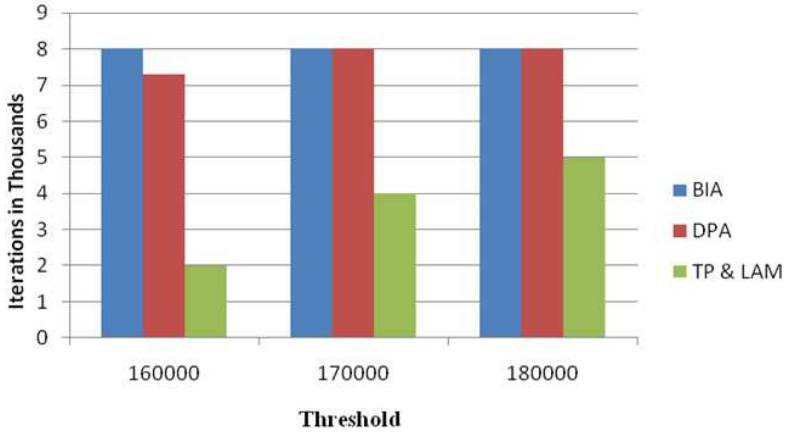

Fig. 13. Iteration analysis of MAX function based on threshold

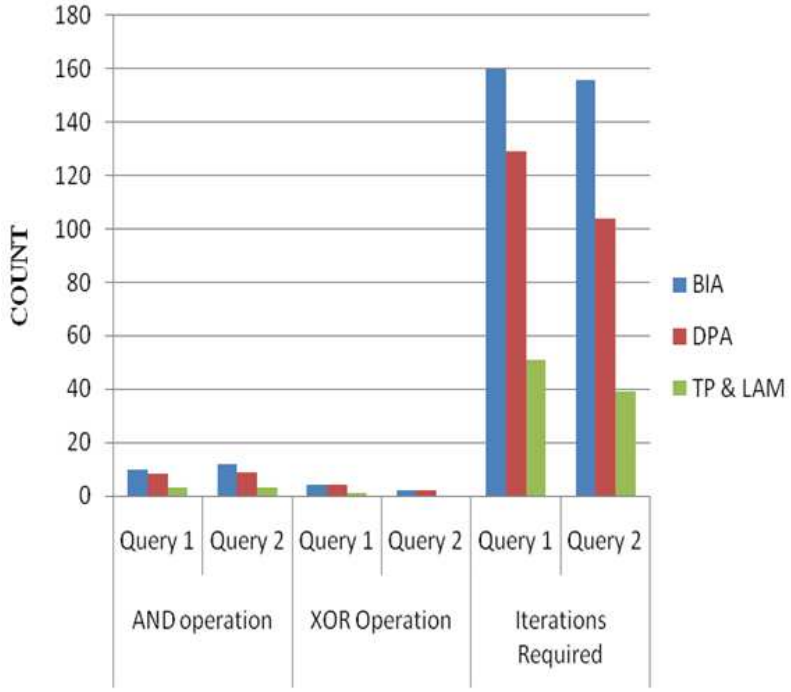

Fig. 14. Query1 and query 2: AND operation, XOR operations and iteration analysis

Figure 9 to 13 shows the iteration and time analysis against TH: Threshold value. In this case also we observe that even though threshold value increases then also the iteration count get reduced for large data set. Only time against threshold value is not directly decrease because it considers time to load huge data set. Also this analysis is based on data available in dataset with similar group. It may possible that with fewer threshold group huge data may present. So in that case it is not always possible that as threshold value increases time and iterations required get reduced. We observe that performance of query also depend upon the aggregate function used in query. IDQ with SUM aggregate function required more time compare to MIN, MAX and COUNT aggregate function.

Figure 14 shows the analysis of query 1 and query 2 of Fig. 2 and 3 respectively. Here we have calculated the number of AND and XOR operation required to solve query manually using BIA, DPA and our TP and LAM strategy. We have also tested both the query on our framework and calculated iterations required to execute the query 1 and query 2. Here also we noticed that performance of our strategy is better compare to old strategies.

Thus with this experimental analysis we prove that performance of IDQ with TP and LAM strategy is better than all previous strategies.

Conclusion

Basic requirement of decision support and knowledge discovery systems is to compute aggregate values of interesting attributes by processing a huge amount of data. In particular, IDQ is a special type of aggregation query that computes aggregate values above a user specified threshold values. Proposed research makes use of TP and LAM strategy and uses bitmap indexing technique for processing IDQ. For efficient evaluation of aggregate function we used TP concept and LAM approach which helps to increase the speed of IDQ. On the basis of experimental results we found that number of iterations required and time required to execute query get reduced to 40-50% even though dataset size increases. Experimental results prove the superiority of our strategy by comparing it with previous research like BIA and DPA. It overcome all the problems occur in previous research such as empty bitwise AND, OR and XOR operation and number of table scan required to execute IDQ. Our research concentrates only on structured data. In future by extending this concept for unstructured data proposed strategy will helpful for big data analysis.

Acknowledgment

The authors wish to thank anonymous reviewers for their valuable, insightful comments that improve the content of this original research paper.

Author's Contributions

Kale Sarika Prakash: The main responsible author for conducting the literature review and drafting the complete research paper. Contributed in design, implementation and testing of framework.

P.M. Joe Prathap: Contributed in preparation, organization and supervision of complete paper.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that the other author has read and approved the manuscript and no ethical issues involved.

References

Bae, J. and S. Lee, 2000. Partitioning algorithms for the computation of average iceberg queries. Proceedings of the 2nd International Conferences on Data Warehousing and Knowledge Discovery, Sept. 4-6, Springer, London, UK, pp: 276-286. DOI: 10.1007/3-540-44466-1_27

Deliège, B. and T.B. Pedersen, 2010. Position list word aligned hybrid: Optimizing space and performance for compressed bitmaps. Proceedings of the 13th International Conference on Extending Database Technology, Mar. 22-26, ACM., Lausanne, Switzerland, pp: 228-239. DOI: 10.1145/1739041.1739071

Dubey, A., A. Kamal and S.C. Gupta, 2014. Effects of aggregation and data size on query performance and memory requirements of a data warehouse. Proceedings of the International Conference on Optimization, Reliabilty and Information Technology, Feb. 6-8, IEEE Xplore Press, pp: 99-104. DOI: 10.1109/ICROIT.2014.6798300

Fang, M., N. Shivakumar, H. Garcia-Molina, R. Motwani and J.D. Ullman, 1998. Computing iceberg queries efficiently. Proceedings of the 24rd International Conference on Very Large Data Bases, Aug. 24-27, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA., pp: 299-310.

Ferro, A., R. Giugno, P.L. Puglisi and A. Pulvirenti, 2009. BitCube: A bottom-up cubing engineering. Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery, Aug. 31-Sept. 02, Springer, Linz, Austria, pp: 189-203. DOI: 10.1007/978-3-642-03730-6_16

Gray, J., A. Bosworth, A. Layman and H. Pirahesh, 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. Data Min. Knowl. Discovery, 1: 29-53. DOI: 10.1023/A:1009726021843

Guru Rao, C.V. and V. Shankar, 2012. Efficient iceberg query evaluation using compressed bitmap index by deferring bitwise-XOR operations. Proceedings of the IEEE 3rd International Advance Computing Conference, Feb. 22-23, IEEE Xplore Press, pp: 1311-1316. DOI: 10.1109/IAdCC.2013.6514418

Guru Rao, C.V. and V. Shankar, 2013. Computing iceberg queries efficiently using bitmap index positions. Proceedings of the International Conference on Human Computer Interactions, Aug. 23-24, IEEE Xplore Press, pp: 1-6. DOI: 10.1109/ICHCI-IEEE.2013.6887811

He, B., H.I. Hsiao, Z. Liu, Y. Huang and Y. Chen, 2011. Efficient iceberg query evaluation using compressed bitmap index. IEEE Trans. Knowl. Data Eng., 24: 1570-1589. DOI: 10.1109/TKDE.2011.73

Inmon, W.H., 2005. Building the Data Warehouse. 4th Edn., Wiley, ISBN-10: 0764599445, pp: 576.

Jrgens, M., 1999. Tree based indexes versus bitmap indexes: A performance study. Proceedings of the International Workshop Design and Managementof Data Warehouses, (MDW' 99).

Leela, K.P., P.M. Tolani and J.R. Haritsa, 2004. On incorporating iceberg queries in query processors. Proceedings of the 9th International Conferences on Database Systems for Advances Applications, Mar. 17-19, Springer, Jeju Island, Korea, pp: 431-442. DOI: 10.1007/978-3-540-24571-1_40

Mei, Y., K. Ji and F. Wang, 2013. A survey on bitmap index technologies for large-scale data retrieval. Proceedings of the 6th International Conference on Intelligent Networks and Intelligent Systems, Nov. 1-3, IEEE Xplore Press, pp: 316-319. DOI: 10.1109/ICINIS.2013.88

Rao, V.C.S., 2014. Efficient iceberg query evaluation using set representation. Proceedings of the Annual IEEE India Conference, Dec. 11-13, IEEE Xplore Press, pp: 1-5. DOI: 10.1109/INDICON.2014.7030537

Shankar, V. and C.V. Guru Rao, 2014. Cache based evaluation of iceberg queries. Proceedings of the International Conference on Computer and Communications Technologies, Dec. 11-13, IEEE Xplore Press, pp: 1-5. DOI: 10.1109/ICCCT2.2014.7066694

Whang, K.Y., B.T.V. Zanden and H.M. Taylor, 1990. A linear-time probabilistic counting algorithm for database applications. ACM Trans. Database Syst., 15: 208-229. DOI: 10.1145/78922.78925

White Paper, 2011. An oracle white paper. Oracle Database 11g for Data Warehousing and Business Intelligence. Oracle

White Paper, 2015. An oracle white paper. Oracle database 12c-Built for Data Warehouse, Oracle.