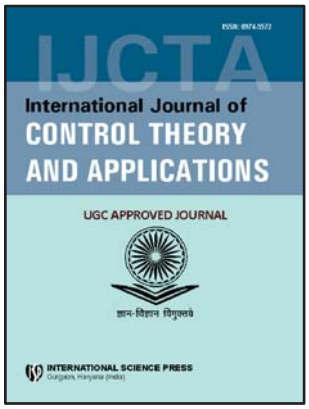

International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 10 • Number 32 • 2017

Look Ahead Matching Strategy an Efficient Approach for Computing Aggregate Function used in Iceberg Query

Kale Sarika Prakash[a] and P.M. Joe Prathap[a]

[a]St. Peters Institute of Higher Education and Research, St. Peters University, Chennai

E-mail: kalesarikaprakash@gmail.com, drjoeprathap@gmail.com

Abstract: Aggregate function plays very important role in analyzing data of data warehouse. Analysis of such a huge data requires execution of complex query such as iceberg query .Iceberg queries are complex as it contains aggregate function followed by HAVING and GROUP BY clause. Processing cost of aggregate function is very high as compare to SELECT and PROJECT relational operation. Efficient execution of aggregate function is very important for the performance of iceberg query. The main objective of this research is to improve the performance of iceberg query in terms of time and iterations required to execute query. Presently available iceberg query processing techniques faces the problem of empty bitwise operations, futile queue pushing and require more table scans which degrade their performance. The model proposed in this research applies concept of look ahead matching on bitmap index of query attributes. Based on the threshold value the analysis of logical operation is done in advance. If result satisfies threshold condition then only remaining part will be evaluated otherwise it will be prune and declare as fruitless operation. In this way by making use of probability concept look ahead matching strategy overcome the problem occurs in previous research. Our experimental result shows 70 to 80 % improvement in performance of iceberg query through our look ahead matching strategy. In this way through this research we are proposing framework for aggregate function used in iceberg query.

Keywords: *Iceberg query (IBQ), Bitwise operations (AND, OR, XOR), Aggregate functions (MIN, MAX, SUM, COUNT), Look Ahead Matching strategy(LAMS), Data warehouse(DW).*

1. INTRODUCTION

Data warehouse is emerging and challenging technology in the field of database management system. It is historical collection of data which help in management's decision making process [1]. In OLAP systems, analysis of data is done by executing different type of queries which run on huge amount of data present in DW [2].The nature of the query to be execute on DW is aggregate function followed by HAVING and GROUP BY clause. Such a type of query is called as IBQ. IBQ perform an aggregate function on attributes specified in query then remove tuples whose aggregate values are below specified threshold value. IBQ's are called so because number of above threshold value tuples are very small and it is on the tip of an iceberg compare to the huge input data set [3].

The main part of any IBQ is aggregate function like COUNT, MAX ,MIN, SUM and AVG . Hence, the performance of query is depends upon the time required to execute aggregate function. Building aggregates on huge data set and executing IBQ efficiently is challenge in front of current researchers.

This research concentrates on efficient execution of aggregate function and IBQ using bitmap indexing mechanism. Bitmap index of required attributes for query processing is in the form of 0's and 1's. Due to this we are performing logical AND,OR and XOR operations as per query requirement. These logical operations can be executed quickly by hardware as they work on 1's and 0's. The major cost in query evaluation is I/O access cost as query is going to execute on large database. Due to highly compress structure, bitmap index require less I/O overheads and large amount of data can be stored in memory for faster evaluation of query [4].The execution cost of logical operations are cheap which directly help in our strategy to improve the performance of IBQ.

The researchers [5, 6, 7, 8, 9] work on improving the performance of IBQ. But they faces the problem of fruitless AND operation, fruitless XOR operation, fruitless OR operation and futile queue pushing. Proposed research overcome above problem by using LAMS during computation of aggregate function and evaluating IBQ. LAMS keep track on probability of pruning the vector from further computation in advance. For this it check intermediate results with threshold condition and take decision related to further processing of query evaluation. This minimizes fruitless bitwise AND,OR and XOR operation which reduces query execution time. Experimental result shows the superiority of IBQ evaluation strategy proposed by this research.

This research paper is arranged in this sequence. Section 2 describes about aggregate functions and IBQ processing, which introduce the concept of aggregate function and IBQ. Section 3 describe the details of LAMS for IBQ evaluation, its implementation details and performance analysis with previous methods. Section 4 is the concluding section of this paper.

2. REVIEW OF AGGREGATE FUNCTIONS AND IBQ PROCESSING

Generally, analysts are interested in summarizing data to determine trends related to their business. This summarized information is useful for top level management of organization for decision making process. For example, the purchase manager of any organization may not be interested in a listing of all computer hardware sales, but may want to know the number of tablet sold in a particular month. In such a situation aggregate functions can assist with the summarization of large volumes of data. Efficient computation of all these aggregate functions are required in most of the database applications. The efficient processing of aggregate function on large database is very important because the base for all IBQ and OLAP operation is aggregate function. If aggregate function is efficient then only query to be execute on large database will perform efficiently.

IBQ refer to a class of queries which compute aggregate functions across attributes to find tuples whose aggregate values are above some specified threshold value. Consider a relation R with attributes *att*1, *att*2… attn, aggregate function AggFun and threshold T on condition C an IBQ has the syntax as below.

SELECT: R.att1, R.att2… R.att n, AggFun (F)

FROM: relation R

GROUPBY: R.att1, R.att2… R.att n

HAVING: AggFun Condition (C) >= T

The number of tuples, that satisfy the threshold in the having clause, is relatively small compared to the huge input data set. The IBQ processing is first studied by Min Fang[3] in 1998.Before this probabilistic techniques were used to process queries with aggregate functions as discussed by K.Whang [10]. In research [3] author described multibucket and hybrid algorithm by modifying the concept of probability based technique proposed by researcher in [10].The major limitations of these algorithms are that they are not suitable for large data sets.

To overcome previously mentioned problem [3] suggest algorithm which uses combination of sampling and bucket counting technique. But the limitation of this algorithm is it requires multiple table access to evaluate query. Due to multiple table access ,time required to evaluate IBQ get increase.

IBQ processing is also described by [11] intention of this work is to reduce multiple table scans. It introduces methods to select initial values using partitioning and postpone partitioning algorithm. This overcome the problem of multiple scan on relation which occurs in [3].The result shows that performance of these algorithms get degrade due to data order and memory requirement. In research [12] author proposes different algorithms but each of these algorithms are having some advantages and disadvantages against each other but common problem with all these algorithm is that they all come under the group of continuous tuple scan based strategy.Execution of these algorithm require at least one physical table scan to read data from disk. All above algorithms only focus on how to reduce the number of table scan but none of them make use of properties of IBQ to solve this problem.

To overcome above problem bitmap index is a perfect choice for querying the huge and high dimensional datasets [13].Generating bitmap index of attribute will not affect on the performance of query because generated bitmap is in compressed mode[14]. In [15] researchers make use of IBQ feature as well as BI but it faces the problem of empty bit wise AND result. This problem is handled by[5] using the concepts of run time pruning and finding adjacent vectors from the list. But this approach suffer from fruitless AND as well as XOR operations. Research [6] handle fruitless XOR operation issue but unable to resolve fruitless AND operation issue. Research [5] and [6] uses priority queue concept but suffer from the problem of fruitless queue operations.

In this research we are making use of LAMS which help to reduce fruitless bitwise XOR operation , AND operation ,OR operation and futile queue pushing problem and improve the performance of IBQ.

3. WORK FLOW OF LAMS FOR IBQ PROCESSING

This section highlights the detail work flow model of look ahead matching strategy and its implementation detail.

3.1. Fundamental Strategy

To implement IBQ, a common strategy in database management system is first to apply hashing or sorting to all the data in the dataset, then to count all attribute by group, and finally to eliminate those groups which do not pass the threshold T. But these algorithms create intermediate results and require more memory to store. They leave much room for improvement and efficiency of algorithm .This research avoids counting from all tuples and checking the validity of vectors. The workflow of LAMS is described in this section.Figure.1 shows the details working of LAMS model. Generating the bitmap as per the IBQ attribute is the initial step of LAMS. Once the bitmap is generated the analysis of it using LAMS will start. First step of LAMS is divide the vector into sub parts. Then analysis of each subpart is done on the basis of priority. Priority of first vectors under consideration is decided on the basis of position of 1 bit in vector. After this the logical operation is perform on vectors as per their priority. If result of logical operation satisfies threshold condition then it will consider for further operations otherwise prune it from the vector list. In this way LAMS find the fruitless bitwise operations in advance and discard such a vectors from list. This help to reduce time and I/o access cost of IBQ.

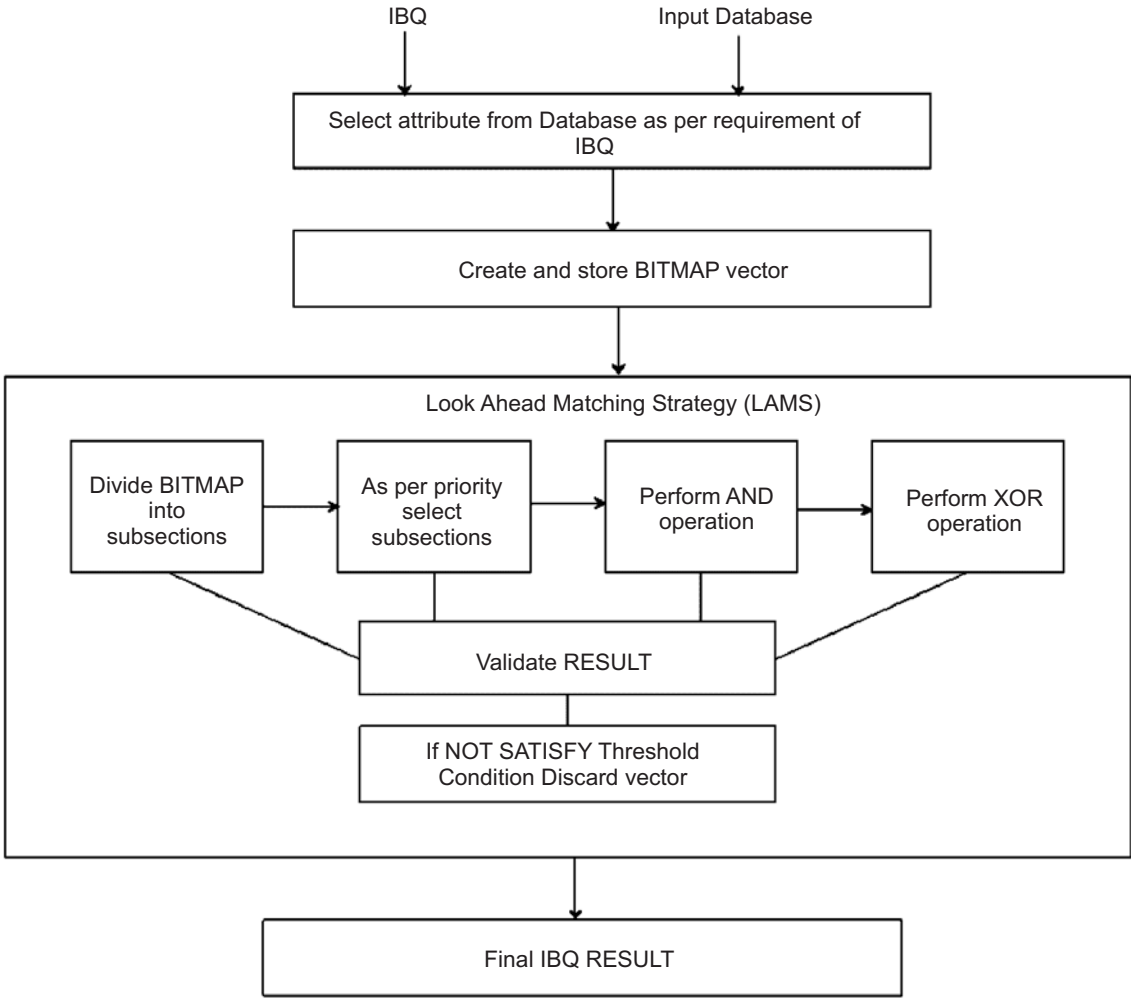

Figure 1: Working model of look ahead matching strategy(LAMS)

3.2. Implementation details of look ahead matching strategy for IBQ evaluation

This section highlights the implementation strategy used in our research .The algorithm of LAMS for IBQ evaluation is work in following sequence.Input to algorithm is (Iceberg Query, Input Database), Iceberg Query parameters are (attribute A1 , attribute A2, threshold T) and output contain combination which satisfy IBQ threshold condition(T).

Algorithm:

1. Generate bitmap vector as per the IBQ attribute list and input database. Generated bitmap is in the matrix format which represent the required input to algorithm in the form of 0's and 1's
2. First strategy to discard the vectors from list. For all vectors

   if (COUNT of 1's bit < T) then discard such a vector from bitmap vector list.
3. Now we will get newly generated bitmap vector list. Assign priority to all the vectors as per the positions of 1's bit occur in respective vector.
4. First clear all queue to be used in processing. PriorityQueueA.clear, PriorityQueueB.clear

5. For each vector x of attribute A do

   if *x*.count >= T then

   x.next1 = First One Bit Position(*x*, 0)

6. PriorityQueueA.Push(*x*)
7. Same logic is applied to push the highest priority vector in Priority QueueB
8. *x*, *y* = Next Aligned Vector(Priority Queue A.clear, Priority Queue B,T)
9. Priority QueueA. clear, Priority Queue B.clear for Location and Product bitmap vector
10. Divide the input vector into subparts.

    Number of subpart = (L/T) where L= length of vector, T = Threshold value .

    P = Number of subparts = {P1,P2,...,Pn}.
11. Start with subpart P1 of each attribute vector Location and Product do
12. Perform Bitwise AND operation
13. If Bitwise_AND_Result > T
14. Then Skip Bitwise AND operation of remaining subpart and send the current combination for XOR operation
15. Else store Bitwise_AND_Result and Perform operation on next subpart. After every operation go on checking Result with Threshold.
16. If all subparts are finished and Bitwise_AND_Result is not > T then simply discard the combination from the list. In this way pruning the vector which does not satisfy the Threshold condition will done.
17. Forward the result for further XOR Operation
18. If Bitwise_XOR_Result > T then put the combination in final IBQ RESULT List. Also generate new vector.

    else discard the combination from the list.
19. Repeat this operation till bitmap vector list will empty.

In this way to implement all type of aggregate operations like COUNT,SUM,MIN and MAX we have used above logic only as per aggregate functions the sequence of the operations will get change . Also while implementing aggregate functions we are considering the threshold value and the condition on which IBQ to be perform and their sequence. This algorithm is implemented using JAVA 7.0 and backend is Oracle 10g.

3.3. Performance Evaluation

The objective of this research is to improve the efficiency of IBQ in terms of number of iterations and time required to execute query. LAMS is work on the bitmap index created on the attributes of the query which help to reduce I/O access cost and time required to access data from database .Due to this feature of bitmap index we are first up all generating bitmap index vector and then applying LAMS. The efficiency of this algorithm is measured using parameters such as number of iterations and time required to evaluate the query. We have conducted the experiment on different tuple size of dataset like 1K,2K,4K and 8K.We evaluate the performance of IBQ using LAMS and previous strategies such as BI strategy (BIS) and Dynamic pruning strategy (DPS). We found significant improvement in performance of IBQ using LAMS.

Figure 2 to Figure 11 represents the performance of LAMS, BIS and DPS for IBQ evaluation. We observed that as we go on increasing the size of database as well as threshold value (T) then also iterations required to evaluate IBQ and time get reduced with LAMS. As shown in Figure 2 to Figure 9 we represent the iteration analysis of SUM and COUNT function. From these graph we observe that there is 70 to 80 % reduction in iterations required to execute IBQ using LAMS also we noticed that as we go on increasing data set double then also iterations required get reduced. In some cases as shown in Figure 2,Figure 3,Figure 5, and Figure 8 even though data size increases the iterations required remain constant.

Time analysis is represented in Figure 10 and Figure 11 for SUM and COUNT function. In case of COUNT function the time required to execute IBQ using LAMS get reduced 70 to 80% compare to previous methods such as BIS and DPS. Whereas in case of SUM function the reduction is not in all cases. As the way of evaluating SUM function is different than COUNT, MIN and MAX function. The performance of SUM function is also depend upon the nature of data present in data set. In this way we have evaluate the performance of IBQ for all aggregate functions like MIN,MAX,SUM and COUNT.

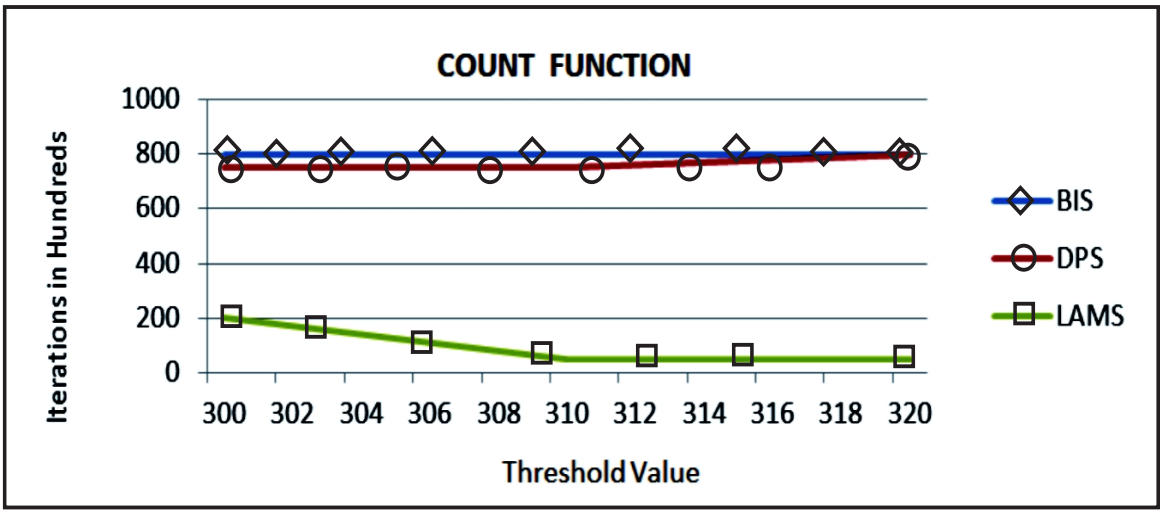

Figure 2: Iteration analysis of COUNT function on 1K data size

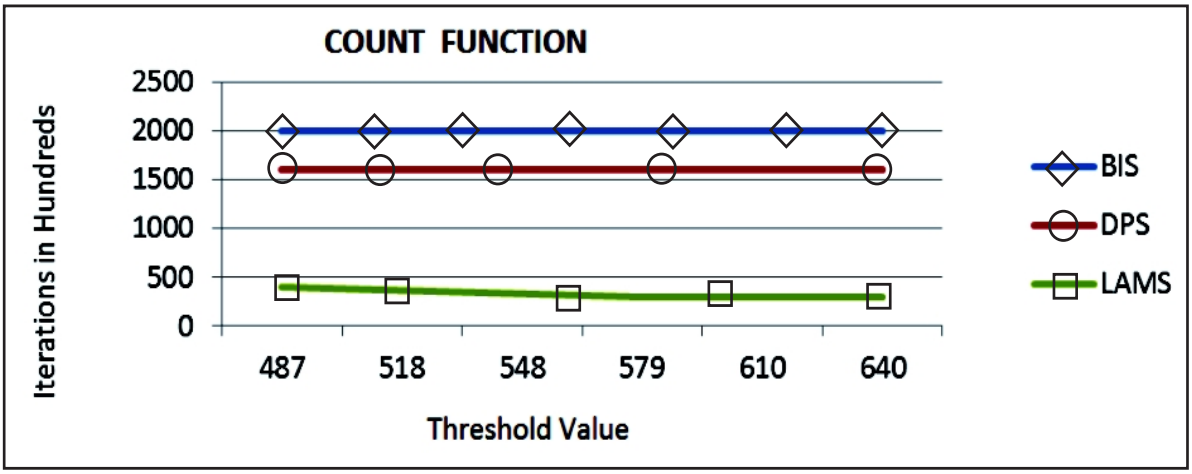

Figure 3: Iteration analysis of COUNT function on 2K data size

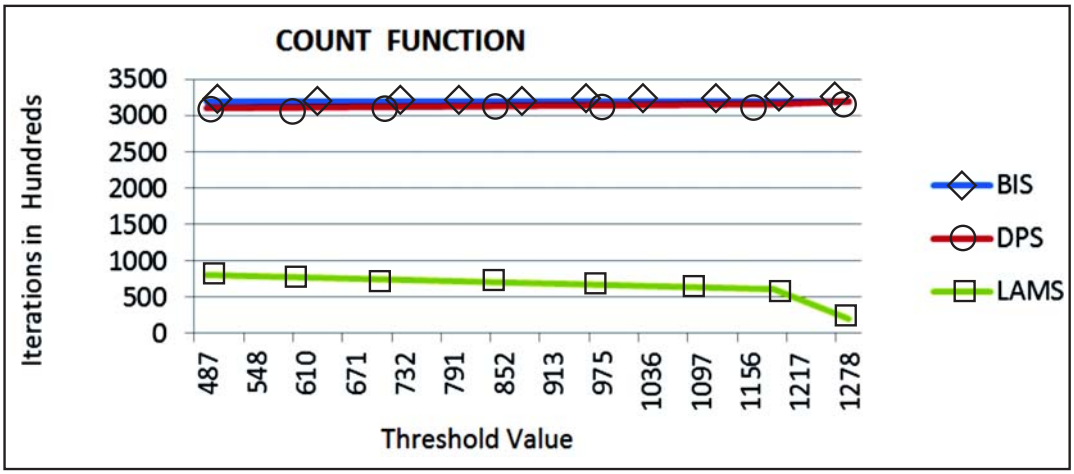

Figure 4: Iteration analysis of COUNT function on 4K data size

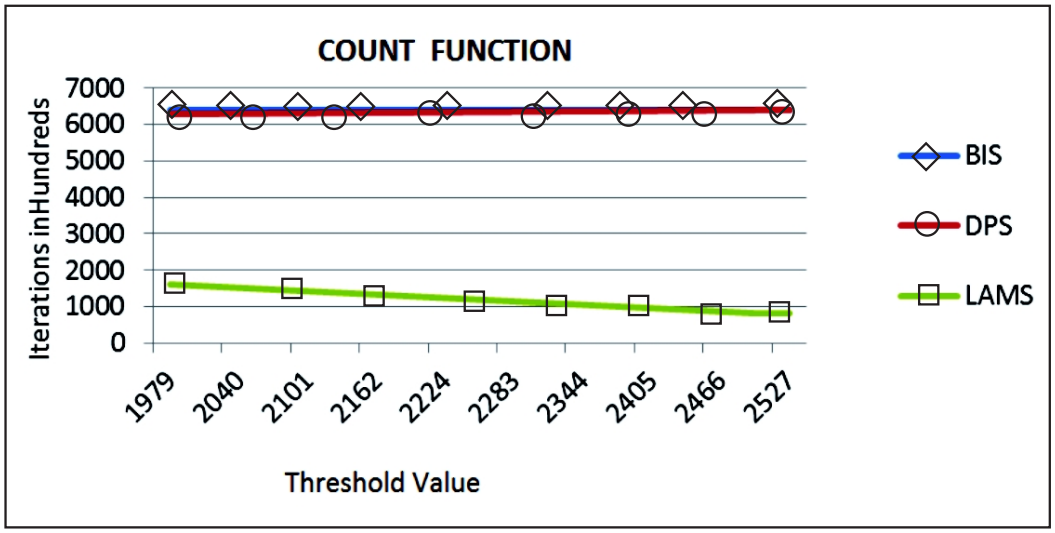

Figure 5: Iteration analysis of COUNT function on 8K data size

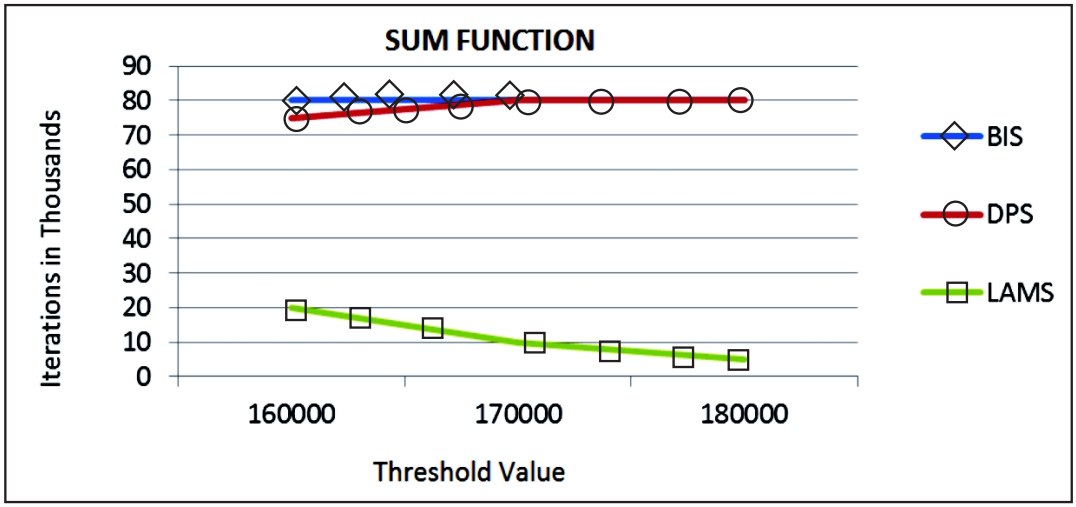

Figure 6: Iteration analysis of SUM function on 1K data size

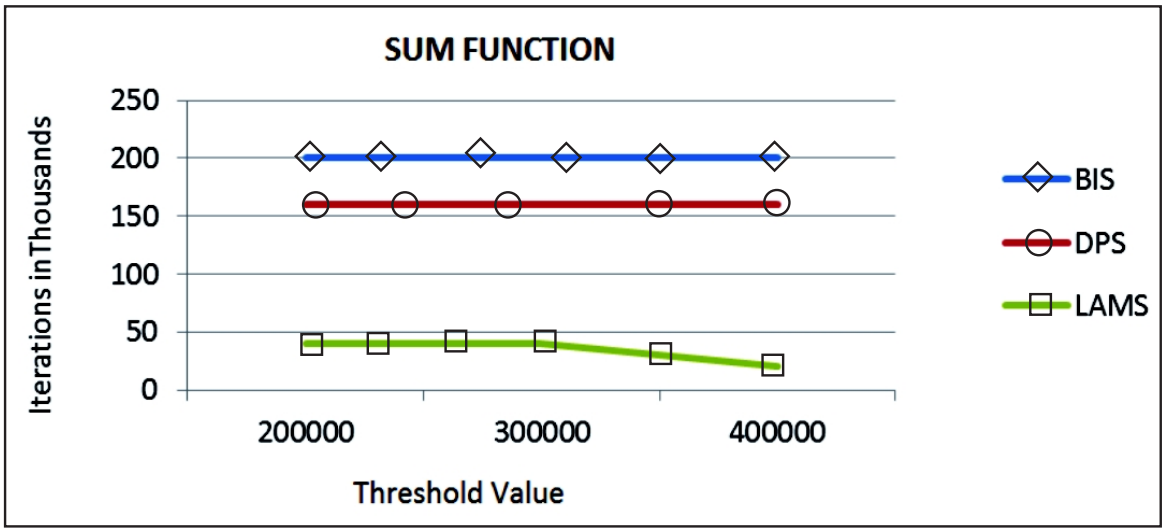

Figure 7: Iteration analysis of SUM function on 2K data size

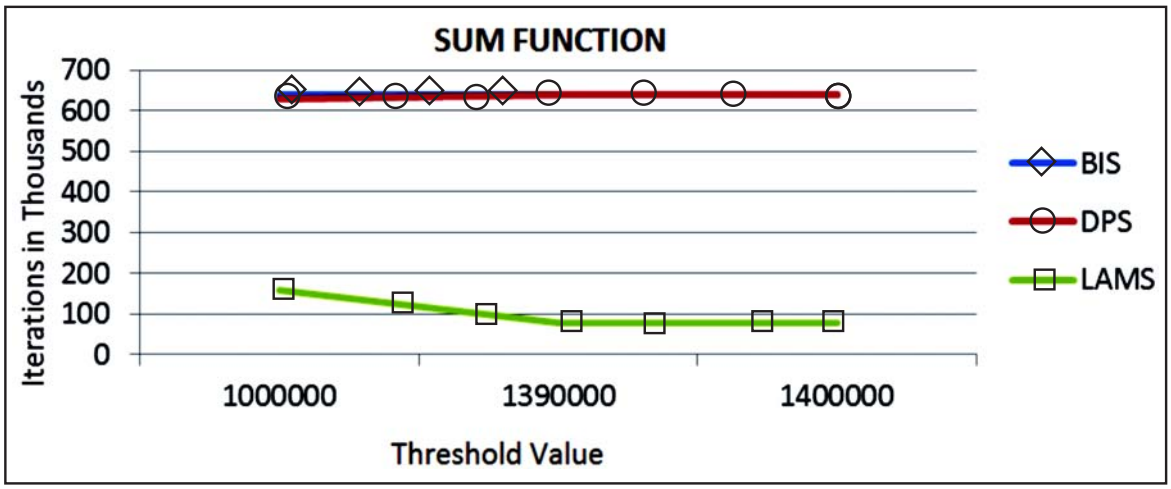

Figure 8: Iteration analysis of SUM function on 4K data size

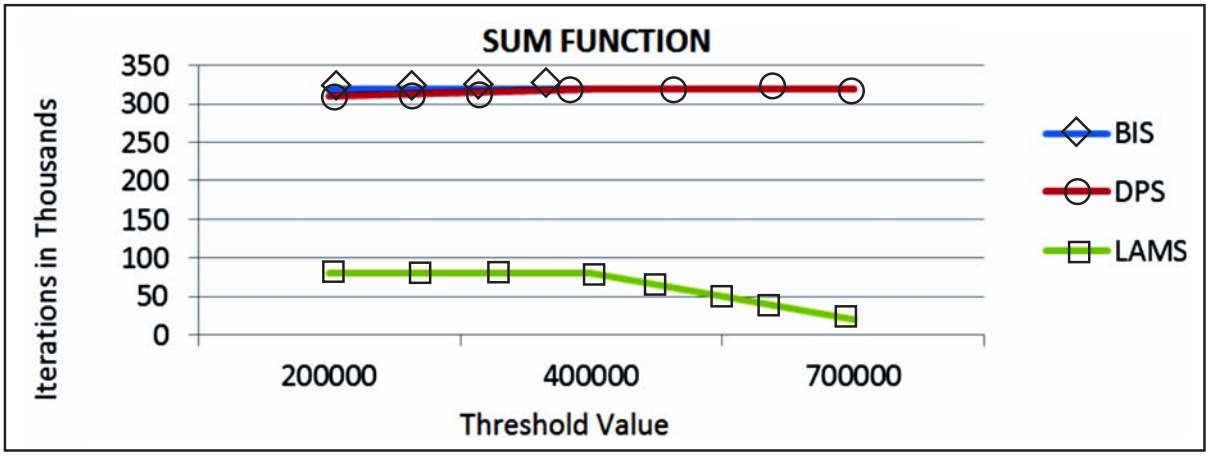

Figure 9: Iteration analysis of SUM function on 8K data size

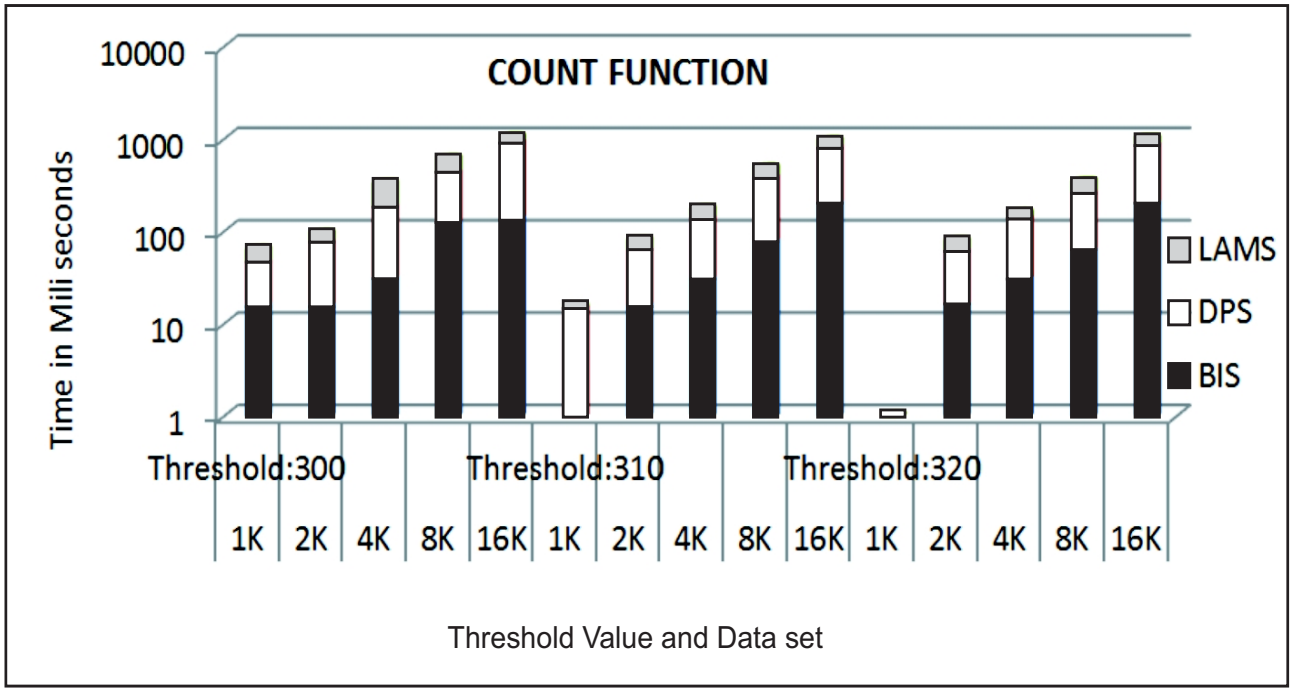

Figure 10: Time analysis of COUNT function

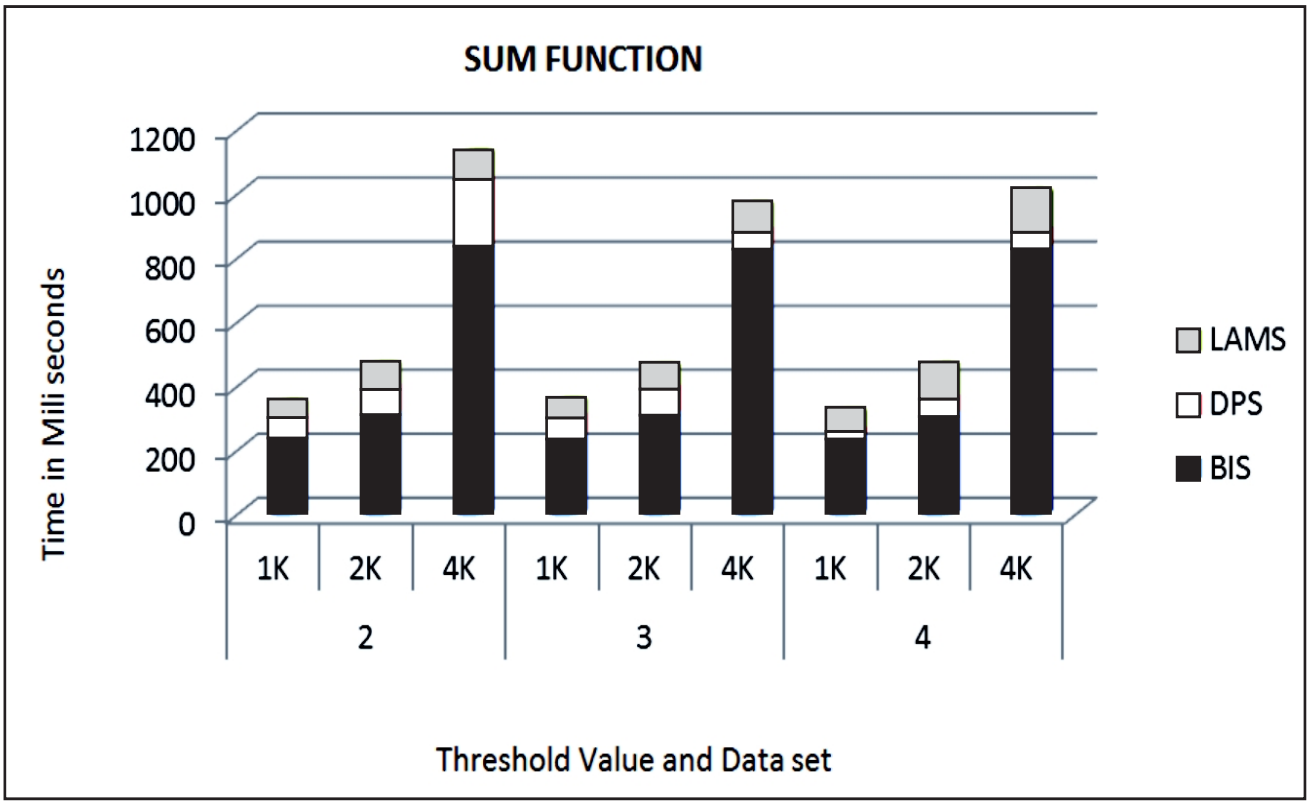

Figure 11: Time analysis of SUM function

4. CONCLUSION

Normally, intention of DW query is analyzing some parameters against other parameters from same database. Almost for all analysis task aggregation is the main function. Processing and executing queries which contain aggregate function in traditional way is time consuming process which require more I/O access and time. In this paper we have proposed look ahead matching strategy for iceberg query evaluation which help to execute such a complex queries efficiently. Look ahead matching strategy makes use of probability concept. It helps to identify the vectors in advance which will not contribute to final result. Such a vectors directly discarded from the bitmap vector list which helps to solve fruitless bitwise AND,OR and XOR operation problem occurs in previous research.LAMS works on bitmap index which help to reduce I/O access cost. The performance of IBQ increases 70-80% in case of look ahead matching strategy. Thus our experimental results help us to prove the superiority of our research compare to all previous strategies. In this way we have proposed the complete framework for COUNT,SUM,MIN and MAX aggregate function. The focus of this research is only structured database but in future we can apply the same logic for query processing on unstructured data.

REFERENCES

[1] Inmon, William H. Building the data warehouse. Wiley.2005

[2] Kazi , B. Radulovic, D. Radovanovic, and Lj Kazi. "MOLAP data warehouse of a software products servicing Call center." In MIPRO. 2010 ,Proceedings of the 33rd International Convention, pp. 1283-1287. IEEE, 2010

[3] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman," Computing iceberg queries efficiently" VLDB Conf., pages 299-310, 1998

[4] Parth Nagarkar,"Compressed Hierarchical Bitmaps for Efficiently Processing Different Query Workloads",IEEE International conference on Cloud Engineering ,DOI 10.1109/IC2E.2015

[5] Bin He, Hui-I Hsiao, Ziyang Liu, Yu Huang and Yi Chen, "Efficient Iceberg Query Evaluation Using Compressed Bitmap Index", IEEE Transactions On Knowledge and Data Engineering, vol 24, issue 9, sept 2011, pp.1570-158

[6] C.V.Guru Rao, V. Shankar,"Efficient Iceberg Query Evaluation Using Compressed Bitmap Index by Deferring Bitwise-XOR Operations " 978-1-4673-4529-3/12/$31.00c 2012 IEEE

[7] C.V.Guru Rao, V. Shankar, "Computing Iceberg Queries Efficiently Using Bitmap Index Positions" DOI: 10.1190/ICHCI-IEEE.2013.6887811 Publication Year: 2013 ,Page(s): 1 – 6

[8] Vuppu.Shankar, Dr.C.V.Guru Rao," Cache Based Evaluation of Iceberg Queries", IEEE International conference on computer and communication technologies(ICCCT),2014 DOI: 10.1109/ICCCT2.2014.7066694 ,Publication Year: 2014

[9] Rao, V.C.S. , Sammulal, P.," Efficient iceberg query evaluation using set representation", (INDICON), 2014 IEEE DOI: 10.1109/INDICON.2014.7030537 Publication Year: 2014 , Page(s): 1 – 5

[10] K.Y. Whang, B.T.V. Zanden, and H.M. Taylor, "A Linear-Time Probabilistic Counting Algorithm for Database Applications," ACM Trans. Database Systems, vol. 15, no. 2, pp. 208-229, 1990

[11] J. Bae and S. Lee, "Partitioning Algorithms for the Computation of Average Iceberg Queries," Proc. Second Int'l Conf. Data Warehousing and Knowledge Discovery (DaWaK), 2000

[12] K.P. Leela, P.M. Tolani, and J.R. Haritsa, "On Incorporating Iceberg Queries in Query Processors" Proc. Int'l Conf. Database Systems for Advances Applications (DASFAA), pp.431-442, 2004

[13] Ying Mei, Kaifan Ji*, Feng Wang," A Survey on Bitmap Index Technologies for Large-scale Data Retrieval" 978-1- 4799-2808-8/13 $26.00 © 2013

[14] F. Delie`ge and T.B. Pedersen, "Position List Word Aligned Hybrid: Optimizing Space and Performance for Compressed Bitmaps," Proc. Int'l Conf. Extending Database Technology (EDBT), pp. 228-239, 2010

[15] A. Ferro, R. Giugno, P.L. Puglisi, and A. Pulvirenti, "BitCube: A Bottom-Up Cubing Engineering,"Proc. Int'l Conf. Data Warehousing and Knowledge Discovery (DaWaK), pp. 189-203, 2009.