ISSN:2320-0790

An International Journal of Advanced Computer Technology

Evaluation of Iceberg Query Using Vector alignment

S.Mahalakshmi[1], T.Sowkarthika2, R.Sindoori[3], S.Sabeetha Saraswathi[4], V.Akila[5]
[1,2,3]Asst Professor, Dept.of Computer Science and Engineering , E.G.S Pillay Engineering College,Nagapattinam
[4]Asst Professor, Dept.of Computer Science and Engineering , SACET, Trichy
PG Scholar, Dept.of Computer Science and Engineering, E.G.S Pillay Engineering College, Nagapattinam

Abstract- Modern computing system requires functionality that often computes aggregate values of interesting attributes by processing a huge amount of data in large databases. Iceberg query is one of the techniques which compute aggregate values in query which is an above user specified threshold. Here the threshold may represent the important and essential factor about the business insights. Usually iceberg query processing algorithm based on tuples scan based approach, which requires intensive disk access and computation, resulting in long pruning time especially when data size is large. The proposed system makes use of bitmap vector to perform query processing which occupies less space. It eliminates the entire databases scanning and processing to evaluate the query. It pruned unwanted processing and saves time and speed up the iceberg query processing significantly by using vector alignment algorithm.

Keywords- Iceberg query, bitmap index, column- oriented database, dynamic pruning, vector alignment.

I. INTRODUCTION

Data mining is the non-trival process to recognize valid, novel and eventually understandable patterns in data, with the extensive use of databases and the explosive growth in their sizes, organizations are faced with problem of information overload. The problem of using massive volumes of data is becoming major problem of all enterprises. Data mining techniques support automatic exploration of data and attempts to source out patters and trends in the data and also infers rules from these patters which will help the user to support review and examine decisions in some related business or scientific area.

The volume of the data base/ Data warehouse is increasing enormously as the need of user requirements are increasing day by day. Most aggregated value represents business knowledge of an organization. This is often required by top officials such as analysts, managers, administrative officers etc to make important decisions. Business Analysts are often responsible to compute and use these aggregate values to compete with present competitive modern business world. Mostly data mining queries are iceberg queries. Iceberg query is one of the techniques which compute aggregate values in query which is an above user specified threshold (T).

Iceberg queries were first studied in data mining field by Min Fang et.al. [5]. The syntax of an iceberg query on a relation R (C1, C2… Cn) is stated below:

SELECT C_i, C_j, …, C_m, AGG(*),

FROM R,
GROUP BY C_i, C_j…, C_m,
HAVING AGG (*) > = T.

Where C_i, C_j,….C_m represents a subset of attributes in R and referred as aggregate attributes. In this paper, we focus on an iceberg query with aggregation function COUNT having the anti-monotone property [1]. Iceberg queries introducing anti-monotone property for many of the aggregation functions and predicates. For example, if the count of a group is below T, the count of any super group must be below T.

Iceberg queries are today being processed with techniques that do not scale well to large data sets. Hence, it is necessary to develop efficient techniques to process them simply. The approaches are classified into tuple-scan and bitwise. First one, a tuple-scan based approach is a simple technique to process an iceberg query. This scheme of evaluation requires at least one table scan to read the data. Hence iceberg query has less efficiency when table size is very large. And also, it is not effectively utilizing the monotone property of iceberg query during its assessment. However, iceberg query is best on reducing the number of passes when the data size is large. In second approach, the iceberg queries are responding using a popular data structure known as bitmap index. A bitmap for an attribute can be viewed as a v×r matrix, where v is the number of distinct values of an attribute and r is the number of rows in the data base. Each value in the column corresponds to a vector of length r in the bitmap, in which the kth position is 1 if this value appears in the k^{th} row, and 0 otherwise.

A way to process an iceberg query using the above bitmap indices is a pair-wise bitwise-AND operation and it is conducted between all distinct values of aggregate attributes. Subsequently, the resultant vector is examined for number of 1's count. If the count is above threshold, then this pair of vector is iceberg result, otherwise bitwise-AND operation is wasted one [6, 9]. This is very inefficient. The next algorithm called naive iceberg processing algorithm which adds pruning step by considering monotone property of iceberg query which prunes the bitmap vectors whose 1's count is below threshold before AND operation. The remaining evaluation process is same as the above algorithm. In another algorithm, the iceberg queries were evaluated quickly by applying pruning step before and after bitwise-AND operation. This type of pruning is called dynamic pruning, because a bitmap vectors will be pruned after several bitwise-AND operations and thus does not need to continue to furnish all the remaining AND operations. Therefore, to improve further the processing speed of the iceberg query, the large numbers of bitmap vectors are to be pruned. Most of the time was spent for AND operations.

Hence, in this paper, we are proposing a strategy to achieve optimal bitmap pruning effect by organizing the PQ with initial high 1s count. This is because vectors with initial high counts are probabilistically more likely to avoid unproductive AND operations. The experimental result for a large synthetic data used signifies a considerable improvement and is more efficient iceberg query computation.

II. RELATED WORKS

Iceberg query[9] is a special class of aggregation query, which computes aggregate values above a given threshold. It is of special interest to the users, as high frequency events or high aggregate values often carry more important information.

A. General From of Iceberg Query

The relation R $(C_1,C_2, \ldots ,C_n)$ is: SELECT C_i, C_j, . . . , C_m, AGG (*) FROM R GROUP BY C_i, C_j, . . . , C_m HAVING AGG(*)>= T C_i, C_j, . . . , C_m represent a subset of attributes in R and are referred as aggregate attributes or grouping attributes. "greater than (>=)" is the comparison predicate. AGG represents an aggregation function. With the threshold constraint, an iceberg query usually returns a very small percentage of distinct groups as the output. Most existing query optimization techniques for processing iceberg queries can be categorized as the tuple- scan-based approach, which requires at least one table scan to read data from disk. Reducing the number of passes when the data size is large, is very difficult. Such a tuple-scan-based scheme often takes a long time to answer iceberg queries, especially when the table is very large.

An index-pruning-based approach was developed to compute iceberg queries using bitmap indices. Bitmap indices provide a vertical organization of a column using bitmap vectors. Each vector represents the occurrences of a unique value in the column across all rows in the table. Today's bitmap indices can be applied on all types of attributes e.g., high-cardinality categorical attributes, numeric attributes and text attributes. A compressed bitmap index occupies less space than the raw data and provides better query performance for equal query rang query and keyword query. Nowadays, bitmap index is supported in many commercial database systems e.g., oracle, syase, Informix and is often the default index option in column-oriented database systems.

Bitmap indices [8,11] are to do pair wise bitwise AND operations between bitmap vectors of all aggregate attributes. It is very inefficient because the product of the number of bitmap vectors in all aggregate attributes is large portion of these operations are not necessary Hear developed the dynamic pruning and vector alignment algorithm, we also notices there is another challenge in the dynamic index-pruning –based approach the problem of massive empty bitwise-AND result. When the number bitwise-AND operation produce empty results and the computation time dominates the query processing time.

III. BITMAP INDICES

Bit map indices [3] are efficient, especially for read-mostly or append-only data, and are commonly used in the data warehousing applications and column stores. Compressed bitmap indices [13] are widely used in column-oriented databases, such as C- Store, which contribute to the performance gain of column databases over row-oriented databases World-Aligned Hybrid (WAH) and Byte-aligned Bitmap Code (BBC): are two compression schemes that can be applied to any column and be used in query processing [10,12] without compression. Development of bitmap compression methods and encoding strategies further broaden the applicability of bitmap index.

A. *Bitmap Index and Its Compression*

A	B	C
A2	B2	1.2
A1	B3	2.3
A2	B1	5.5
A2	B2	8.3
A1	B3	3.2
A2	B1	9.4

A1	A2	A3
0	1	0
1	0	0
0	1	0
0	1	0
1	0	0
0	1	0

B1	B2	B3
0	1	0
0	0	1
1	0	0
0	1	0
0	0	1
1	0	0

A2	B2	6.2
A2	B1	1.9
A1	B3	8.2
A2	B2	0.1
A3	B1	3.4
A3	B1	2.0

0	1	0		0	1	0
0	1	0		1	0	0
1	0	0		0	0	1
0	1	0		0	1	0
0	0	1		1	0	0
0	0	1		1	0	0

(a)Table R (b) Bitmap Indices for A, B

Fig. 1.Bitmap index example

SELECT A, B, COUNT (*) FROM R GROUP BY A, B, HAVING COUNT (*)>=2

Fig. 2. An iceberg query with COUNT function

A bitmap for an attribute (column) of a table can be viewed as a v*r matrix, where v is the number of distinct values of the column and r is the number of tuples (rows) in the table. An example of bitmap index is show in Fig. 1. Fig. 1a shows an example relation with a set of attributes. Fig. 1b shows the corresponding bitmap indices on attributes A and B of the table.

For each distinct values of A and B, there is a corresponding bitmap vector. For instance, Karaborn's bitmap vector is 010010001000, because Karbonn occurs in the second, fifth, and ninth rows in the table. An uncompressed bitmap can be much larger than the original data, thus compression is typically utilized to reduce the storage size and improve performance.

IV DYNAMIC PRUNING

Dynamic pruning algorithm use an iceberg query having two aggregate attributes with COUNT function as the running example. Suppose the iceberg query that we need to answer is as the one in Fig. 2. The data table and bitmap indices are as that in Fig.1. This way is to process this iceberg query on two attributes A and B using bitmap indices is to conduct pair wise bitwise- AND operations between each vector of A and each vector of B.

A.Bitwise-AND operations

Consider the example in the table R, column A has three distinct values —Micromax, Nokia, Samsung and column B has three distinct values: Karbonn, Apple, LG. The bitmap indices are those on the right of Fig. 1. To process the iceberg query in Fig.2, this approach will conduct bitwise-AND operation between nine pair (Karbonn, Micromax), (Karbonn, Nokia), (Karbonn, Samsung), (Apple, Micromax), (Apple, Nokia),(Apple, Samsung), (LG, Micromax), (LG, Nokia), and (LG, Samsung). After each bitwise- AND operation, the number of 1 bits in the resulting bitmap vector is counted. If the number of 1 bits is larger than the threshold it is added into iceberg result set. Threshold, this vector can be pruned.

Consider the bitmap vector Nokia=101101110100 AND Karbonn= 001001010011 of our running example in Fig. 1. When a bitwise-AND is conducted between them, the resulting vector is 001001010000. Also, Nokia becomes 100100100100 and Karbonn becomes 000000000011.After each bitwise-AND operation, the dynamic pruning strategy adds an extra pruning step of monitoring the number of remaining 1s in both bitmap vectors involved. If the number of 1 bit of a modified vector becomes smaller than the iceberg Consider our running example, suppose bitwise-AND operations are first conducted between Micromax and all values in B.(Micromax, Karbonn) and (Micromax, Apple) produce no result. After the bitwise-AND operation between Micromax and LG is done, the number of 1s left in LG is two, which does not meet the threshold in the query. Thus, LG can be pruned. Then, when we process Noika, we only conduct bitwise-AND operations on (Nokia, Karbonn) and (Nokia, Apple). The pair (Nokia, LG) is pruned. Further, Samsung can be directly pruned because it only contains two 1 bits. No bitwise-AND operations are needed between Samsung and B outside of the number of operations is reduced from nine to five.

V VECTOR ALIGNMENT

The dynamic pruning strategy works well for attributes with a relatively small number of unique values, its performance downgrades severely due to the empty bitwise-AND result problem. With the dynamic index pruning strategy alone, many of the bitwise-AND operations produce empty results after a bitwise-AND operation. That is the resulting bitmap vector contains no bits having values 1. Such bitwise-AND operations are fruitless in two aspects -They do not produce valid iceberg result and they do not reduce the number of 1 bit in original vectors for index pruning purpose.

Consider an example suppose a table has 1,000,000 tuple, its attributes A has 10,000 unique values, and attributes has 10,000 unique values. A and B will have 10,000 bitmap vectors each .In the worst case, the total number of pair wise bitwise-AND operation is 10,000*10,000= 100,000,000, Which is 100 times larger than the number of tuples. Since the number of distinct groups is bounded by the number of tuples n in the relation, we need at most n bitwise-AND operation to answer an iceberg query In this example, more than 99 percent of the bitwise-AND operation are useless

First 1-bits position

It refers to the position of the first 1-bit in a bitmap vector.

Priority Queue 1	Priority Queue 2
Nokia	101101110100
Apple	100100100100
Micromax	010010001000
LG	010010001000
Samsung	000000000011
Karbonn	001001010011

Fig. 3. Bitmap vector in priority queues.

Number of 1s in LG is not larger than 2

Priority Queue 1	Priority Queue 2

Micromax	010010001000
LG	010010001000
Nokia	001001010000
Karbonn	001001010011

Fig. 4. Bitmap vector after first vector alignment.

Apple is removed. Since our AND operation will update the original vectors, the first 1-Bit position will update the original vectors, the first 1-bit position of a vector may thus change after the AND operation

A.Vector alignment

Two bitmap vectors are aligned if their first 1-bit positions are the same [2].

Consider our example in Fig. 1 and the query in Fig. 2. Fig. 3 shows the priority queues for attributes A and B. It is not necessary to put the vector Samsung in A's priority queue because Samsung only contain two 1 bits and can be pruned directly.

B.Algorithm1: Iceberg Processing with Vector Alignment and Dynamic Pruning

It has two phases. In the first phases, we prioritize bitmap vectors of each attribute by their first 1-bit positions. The function first1bitposition is to find the position of the first 1-bit.

```
Iceberg PQ (attribute A, attribute B, threshold T)
Output: Iceberg results.
$PQ_A$.clear, $PQ_B$.clear
for each vector a of attribute A do
a. count = BIT1_COUNT(a)
if a.count >= T then
a.next1 = first1BitPosition(a, 0)
$PQ_A$.push (a)
for each vector b of attribute B do
b.count = BIT1_COUNT(b)
if b.count >= T then
b.next1=first1BitPosition(b, 0)
$PQ_B$. push(b)
R =θ
a,b = nextAlignedVectors($PQ_A$, $PQ_B$, T)
while a ≠ null and b ≠ null do
$PQ_A$.pop
$PQ_B$.pop
r = BITWISE_AND (a, b)
if r.count >= T then
Add iceberg result (a.value, b.value;
r.count) into R
a. count = a. count - r.count
b.count = b.count - r.count
if a. count >= T then
a.next1 =first1BitPosition(a, a.next1 + 1)
if a.next1 ≠ null then
$PQ_A$.push (a)
if b.count >= T then
b.next1 ≠ first1BitPosition(b, b.next1 + 1)
if b.next1 ≠null then
$PQ_B$.push (b)
a, b = nextAlignedVectors($PQ_A$, $PQ_B$, T)
return R
```

*C.Algorithm2: First 1 bit positio***n.** its shows the detail of the First 1 bit position function. BIT1_COUNT is used to count the number of 1s in a.

```
first1BitPosition (bitmap vector vec, start position pos)
Output: the position of the first 1 bit position in vec,
starting from position pos

len =0
for each word w in vector vec do
if w is a literal word then
if len<=pos AND len +31>pos then
for p=pos to len +30 do
if position p is 1 then
return p
else if len>pos then
for p=len to len+30 do
if position p is 1 then
return p
len+=31
else if w is a 0 fill word then
fillLength =length of this fill word
len+=fillLength*31
else
fillLength=length of this fill word
len+=fillLength* 31
if len>pos then
return pos
return null
```

D. Algorithm 3. Find Next Aligned Vectors

```
nextAlignedVectors (priority queue $PQ_A$, priority queue
$PQ_B$, threshold T)
Output: two aligned vectors a Є $PQ_A$, b Є $PQ_B$
1: while $PQ_A$ is not empty and $PQ_B$ is not empty do
2: a = $PQ_A$.top
3: b = $PQ_B$.top
4: if a.next1 = b.next1 then
5: return a, b
6: if a.next1 > b.next1 then
7: $PQ_B$.pop
8: b.next1, skip = first1BitPositionWithSkip
  (b, a.next1)
9: b.count = b.count-skip
10: if b.next1≠ null AND b.count >= T then
11: $PQ_B$.push(b)
12: else
13: $PQ_A$.pop
14: a.next1, skip=first1BitPositionWithSkip
  (a, b.next1)
15: a.count = a.count-skip
```

```
16: if a.next1 ≠null AND a.count >= T then
17: PQ_A.push(a)
18: return null, null
```

E.Optimization

To improve the performance, two additional optimization techniques are developed: 1) by using tracking pointers to accelerate vector relevant operations, and 2) by using a global filter vector to reduce useless queue pushing.

VI. IMPLEMENTATION

In this suite of experiments, we tested icebergDP and icebergPQ on data sets with zipfian distribution. We varied the data size from 1 to 8 million tuples. We didn't test icebergDP with larger data set because its performance is already very slow when the data size is 8 million.

As shown in Fig.4, the performance of icebergPQ is magnitudes faster than icebergDP

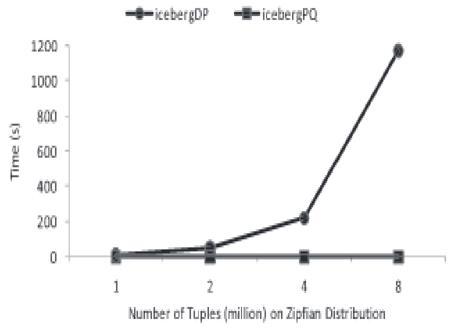

Fig.4 Performance of icebergDP and icebergPQ.

It demonstrates the severe performance issue triggered by the empty bitwise- AND results problem discussed before. With 1 million tuples, icebergPQ only needs 0.404 seconds to finish processing, while icebergDP needs 10.688 seconds. icebergPQ also scales well when the data size increases. It only takes 11.36 second with 8 million tuples, while icebergDP takes more than 18 minutes. The performance of icebergDP is unacceptable for practical data sizes.

VII. CONCLUSION

Evaluation of iceberg query using vector alignment present an efficient algorithm for iceberg query processing using compressed bitmap indices. The superior performance over existing schemes and it does not on any particular compression method. By this approach we can save disk access by avoiding tuple-scan on a table with a lot of attributes, save computation time by conducting bitwise operations and leveraging the anti monotone property of iceberg query to develop aggressive pruning strategies.

There are several issues that we consider as future work. First, we would like to investigate the processing of iceberg queries without the antimonotone property, e.g., queries with AVERAGE functions. For this type of queries, even if a pair of values (a;b)does not satisfy the predicate, its superset (a;b;c) may still satisfy the predicate, which makes pruning much harder.

REFERENCES

[1] Agrawal.R, Imielinski T., and Swami A.N., "Mining Association Rules between Sets of Items in Large Databases," Proc.ACM SIGMOD Int'l Conf. Management of Data, pp. 207-216, 1993.

[2] Bin He,Hui-I Hsiao,Ziyang Liu,Yu Huang and Yi Chen —Efficient Iceberg Query Evaluation Using Compressed Bitmap Index , vol. 24,. NO.9, September 2012.

[3] Chan C.Y.and Ioannidis Y.E., "Bitmap Index Design and Evaluation,. -Proc.ACM SIGMOD int'I Conf. Management of Data,1998.

[4] Fang M., Shivakumar N.,. Garcia-Molina H, Motwani R., and Ullman J.D., "Computing Iceberg Queries Efficiently," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 299-310, 1998

[5] Ferro A, R.Giugno, P.L.Puglisi, and A.Pulvirenti,"Bitcube: ABottom-Up Cubing Enginerring, "Proc. Int'l Conf. Data Warehousing and knowledge Discovery (DaWak), pp. 189-203, 2009.

[6] Graefe G., Query Evaluation Techniques for Large Databases," ACM Computing Surveys, vol.25, No.2, pp.73-170,1993.

[7] Han J., Pei J.. Dong G., and Wanng, K.,"Efficient Computation of Iceberg Cubes with Complex Measures, Proc. ACMSIGMOD Int'l Conf. Management of Data,pp1-12,2011.

[8] Jrgens M, "Tree Based Indexes versus Bitmap Indexes: A Performance study," Pro.Int'l Workshop Design and Management of Data Warehouses (dmdw), 1999.

[9] Larson P.-A," Grouping and Duplicate Elimination: Benefits of Early Aggregation, "Technical Report MSR-TR-97-36, Microsoft Research, 1997.

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

[11] O'Neil P.E. and Graefe G., "Multi-Table Joins through Bitmapped Join Indices," SIGMOD Record, vol. 24, no. 3, pp. 8-11, 1995.

[12] Stockinger K, J. Cieslewicz, K. Wu, Rotem D., and Shoshani A., "Using Bitmap Index for Joint Queries on Structured and Text Data," Annals of Information Systems, vol. 3, pp. 1-23, 2009

[13] Wu K, Otoo E.J., and Shoshani A., "On the Performance of Bitmap Indices for High Cardinality Attributes," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 24-35, 2004.