Volume 3, Issue 9, September 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Efficient Evaluation of Iceberg Queries Using Priority Queues

Sandeep Reddy Chidirala*
M. Tech, Software Engineering
Dept of Computer Science & Engg.
Kakatiya Institute Of Tech & Sci
Kakatiya University, Warangal, India

Shankar Vuppu
Associate Professor
Dept of Computer Science & Engg.
Kakatiya Institute Of Tech & Sci
Kakatiya University, Warangal, India

Abstract*— An iceberg query is a special class of an aggregate query that computes aggregate values upon a user given threshold (T). An iceberg query evaluation using compressed bitmap index strategy is an efficient strategy which prunes the bitmap vectors. Hence, this index technique is used in our proposed approach. In this paper, we propose an effective strategy of ordering the priority queues with highest initial 1's count to achieve optimal pruning of bitmap vectors. This is because there is a high probability for selection of more common 1 bits in vector. Exhaustive experimentation demonstrates our strategy is much more efficient than existing strategy.

Keywords*— Database, Iceberg query, Bitmap vector, Dynamic pruning, Bitwise-AND operation, Threshold.

I. INTRODUCTION

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. In particular, iceberg queries are unique class of aggregation query that compute an aggregate value above user specified threshold (T).

Iceberg queries were first studied in data mining field by Min Fang et.al. [10]. The syntax of an iceberg query on a relation R (C1, C2… Cn) is stated below:

SELECT Ci, Cj, …, Cm, AGG(*),
FROM R,
GROUP BY Ci, Cj…, Cm,
HAVING AGG (*) > = T.

Where Ci, Cj,….Cm represents a subset of attributes in R and referred as aggregate attributes. AGG represents an aggregation function such as COUNT, SUM, MIN and MAX. The greater than or equal to (>=) is a symbol used as a comparison predicate. In this paper, we focus on an iceberg query with aggregation function COUNT having the anti-monotone property [6]. 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 easily. The approaches are categorized into tuple-scan and bitwise. First one, a tuple-scan based approach is a simple technique to process an iceberg query [5], [14]. This scheme of evaluation requires at least one table scan to read the data from the disk. Hence it offers a long time to answer an iceberg query when table size is very large. And also, it is not effectively utilizing the monotone property of iceberg query during its evaluation. However, this is best on reducing the number of passes when the data size is large. In second approach, the iceberg queries are answered 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 kth row, and 0 otherwise. An immature way to process an iceberg query using the above bitmap indices is a pair-wise bitwise-AND operation 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 declared this pair of vector is iceberg result, otherwise AND operation is wasted. Obviously 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 further improve the processing speed of the iceberg query, the large numbers of bitmap vectors are to be pruned. But it was observed that considerable amount of time was spent due to vain AND operations.

Hence, in this paper, we are proposing a strategy to achieve optimal bitmap pruning effect by organizing the PQ with initial high 1's 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. REVIEW OF RELATED RESEARCH WORK

The research work is reviewed in two subsections A and B. In the first subsection. i. e. A, we provide a review of related algorithm known as Naïve iceberg processing and iceberg processing with dynamic pruning algorithm using bitmap indices which is the focus of this paper for optimization of iceberg queries in second subsection i.e. B.

Iceberg query evaluation: The following algorithm for computing iceberg queries by implementing the monotone property.

Algorithm 1:Icebergnaive (attribute A, attribute B, threshold)
Output: iceberg results
1. VA = null, VB = NULL
2. For each vector a of attribute A do
3. a.count = BIT1_COUNT(a)
4. if a.count>= T then
5. Add a into VA
6. For each vector b of attribute B do
7. b.count = BIT1_COUNT(b)
8. if b.count>= T then
9. Add b into VB
10. R = null
11. for each vector a in VA do
12. for each vector b in VB do
13. r = BITWISE_AND(a, b)
14. if r. count>= T then
15. Add iceberg result (a.value, b.value, r.count)into R
16. return R

According to the above algorithm, for an iceberg query with two attributes A and B having threshold T, at first A's vectors are read and the number of 1 bits in the vector is compared with the threshold (T) and if the count is less than T then it is ignored by using monotone property. The same process is carried for the B's vectors. The BIT1_COUNT function is implemented to count the number of 1's in a vector. The vectors with 1's count less than T are not considered, all the vectors are not pruned by monotone property in above algorithm. Now the BITWISE-AND operation is conducted between all the remaining A's and B's vectors. The count of the resultant vector is compared with T and if it is greater than T then it is added to the iceberg result set else it is ignored. The first phase of the algorithm (i.e., lines 1-9 of the algorithm) only takes the advantage of the monotone property but not the second phase of the algorithm (i.e., lines 10-15). This may result in unnecessary AND operations.

Iceberg query evaluation: The below algorithm gets advantage for computing iceberg queries by implementing the monotone property in both phases.

Algorithm 2: Iceberg processing with dynamic pruning
Icebergdynamicpruning (attribute A, attribute B, threshold T)
Output: iceberg results
1.VA = null, VB = NULL
2. For each vector a of attribute A do
3. a.count = BIT1_COUNT(a)
4. If a.count>= T then
5. Add a into VA
6. For each vector b of attribute B do
7. b.count = BIT1_COUNT(b)
8. if b.count>= T then
9. Add b into VB
10. R = null
11. while VA± null and VB± null
12. a,b = choosevectorpair(VA, VB)
13. r = BITWISE_AND(a, b)
14. if r. count>= T then
15. Add iceberg result(a.value, b.value, r.count) into R
16. a.count = a.count-r.count
17. b.count = b.count-r.count
18. if a.count< T then
19. Remove a from VA
20. if b.count< T then

21. Remove b from VB
22. return R

The first phase of the algorithm is same as the algorithm icebergnaive. In the second phase after conducting BITWISE-AND operation the count of the resultant vector r is compared with T and if it satisfies the threshold then it is added to iceberg resultset. Then the count of A's and B's vectors is decreased by r's count and if any of their remaining count is less than T, the vector is removed to avoid unnecessary computations.

The next section presents a proposal to effectively prune the bitmap vectors by ordering PQ with high initial 1's count in the above stated process.

III. PROPOSED RESEARCH WORK

This section proposes the research work to be carried out on the topic under investigation in two subsections. The first subsection i.e., A, an algorithm for ordering the bitmap vectors in PQ with highest initial 1's count. The second subsection i.e., validates the proposal using a sample database.

Proposed Algorithm : Iceberg query processing by ordering the bitmap vectors in PQ with high initial 1's count.

IcebergqueryprocessingwithorderedPQ anddynamicpruning (Priority queue A, Priority queue B, threshold T)

Output: iceberg results

1. PQA = null, PQB = null
2. For each vector a of attribute A do
3. a.count = BIT1_COUNT(a)
4. if a.count>= T then
5. PQA.push(a)
6. for each vector b of attribute B do
7. b.count = BIT1_COUNT(b)
8. if b.count>= T then
9. PQB.push(b)
10. R = null
11. while PQA± null and PQB± null
12. a,b = selectvectorpair(PQA, PQB)
13. r = BITWISE_AND(a, b)
14. if r. count>= T then
15. Add iceberg result(a.value, b.value, r.count) into R
16. a.count = a.count-r.count
17. b.count = b.count-r.count
18. if a.count< T then
19. PQA.pop(a)
20. if b.count< T then
21. PQB.pop(b)
22. return R

The algorithm operates in three phases. In the first phase of the algorithm i.e., from statement 1 to 9, prioritizes the bitmap vectors of each attribute with high initial 1's count. The second phase of the algorithm i.e., from statement 10 to 15 are implemented by selectvectorpair function to process the iceberg query. A bitmap vector 'r' is calculated by conducting BITWISE-AND operations on the selected pair of bitmap vectors from each PQ's. If the number of 1's in 'r' is greater than T, the vectors are added to the iceberg resultset. The third phase (statement 16-21), here the 1's count of each bitmap vector is computed by decreasing the r's count from original vector. The modified count is compared with threshold and if it is above T, it remains at the same position in the PQ otherwise it is pruned. The process is repeated until either of the PQ gets emptied.

Validation of algorithm:

This subsection demonstrates the validity of the proposed strategy using an iceberg query having two aggregate attributes with COUNT function.

SELECT A, B, COUNT (*) FROM R GROUP BY A, B HAVING COUNT (*) >3;

Fig2: An Iceberg query with count function

The above listed query is assigned to fetch the count of attribute A and B values from the database table R under a specified parameter called threshold T which is greater than the value 3 should be selected. The iceberg query that we need to answer is in fig 2, the data base table R and bitmap indices are those in fig 3a and fig 3b.

A	B	C
A1	B2	500
A3	B1	600
A2	B2	700
A3	B1	500
A2	B2	600

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

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

A3	B1	700	0	0	1	1	0	0
A1	B2	700	1	0	0	0	1	0
A2	B2	600	0	1	0	0	1	0
A1	B1	500	1	0	0	1	0	0
A2	B2	700	0	1	0	0	1	0
A3	B1	600	0	0	1	1	0	0
A2	B3	500	0	1	0	0	0	1
A1	B1	600	1	0	0	1	0	0
A3	B3	700	0	0	1	0	0	1
A3	B2	500	0	0	1	0	1	0

Database table(R) Bitmap Index A Bitmap index B

The bitmap vectors are entered into priority queues say PQA and PQB in the order of their preliminary high 1 bit counts as A3, A2, A1 and B2, B1. The 1's count in vector B3 is not satisfying the iceberg threshold, so it is directly pruned. Then the vectors A3: 010101000010011 and B2: 101010110100001 are chosen from the priority queues PQA and PQB. Now a bitwise-AND operation is performed between them. The 1's count of the resulting vector r: 000000000000001 is computed and then compare it with T as 1. This is below threshold. As the result is not satisfying the threshold the pair of bitmap vectors A3 and B2 is not an iceberg result. Now the 1's count of A3 and B2 is decreased by 1's count of the resulting vector. Then the new 1's count of the vectors A3 and B2 is compared with T as 5 and 6 respectively which are greater than T. Therefore both the vectors remain in the same position in queue (Since according to our approach ordering of bitmap vectors takes place only once i.e., there is no rearrangement of vectors in queue when its 1's count gets decreased). Now a bitwise-AND operation is conducted between the vectors A3 chosen for PQA and B1 chosen from PQB. The resulting vector is r: 010101000010000. The 1'count in the r is computes as 4, it passes the threshold. So the vector pair A3 and B1 are declared as iceberg result and add them into iceberg result set along with its count value. Now the 1's count of vectors A3 and B1 is updated as 1 and 2 respectively. Hence both the vectors are pruned.The process is continued till either of the queues gets emptied.

The algorithm performs a total of three AND operations. Of which two produced the result tuples as A3, B1 with count as 4 and A2, B2 with count as 4. In the next section we implement the proposed algorithm by various modules.

IV. IMPLEMENTATION

This section describes the distinctive modules that were proposed in previous section, the details are as follows:

4.1 Generate Bitmap Vectors

The first module i.e., GenerateBitmapvectors disintegrate the given iceberg query into relations and then generates bitmap vectors fro each aggregate attribute in that elation. In this process the bitmap vetors uses the existing word aligned hybrid compression scheme(WAH) and arranges all the bitmap vectors into compresses words.

4.2 Select vector pair

This module selects two bitmap vectors one each from PQA and PQB arranged with high initial 1's count to conduct bitwise-AND operation.

4.3 IcebergqueryprocessingwithorderedPQ anddynamicpruning

This module uses the selectvectorpair function to choose two high 1's count bitmap vector and perform bitwise-AND operation between them. The count of the resulting vector is computed and if the count of 'r' is greater than T then the pair is added to the iceberg resultset else discarded. Further the number of 1's in resulting vector 'r' is now compared with 1's count in both vectors and if the count is greater than T then the vector remains in PQ in the same position else the vector is pruned. The above modules are implemented using JAVA for the purpose of experimentation. The next section explains the experiments conducted on this implementation.

V. EXPERIMENTATION AND RESULTS

This section describes the experiment carried out on the implementation described in the previous section under a specified iceberg threshold values that increase from 1000 to 10000. The first row in table 1 indicates different thresholds. The second, third and fourth rows correspond an execution time made in icebergPQ and icebergDB (density PQ) and icebergWD(without density) functions respectively.

Table 1 tabulates the different execution times in seconds for the iceberg result set with respect to icebergPQ, icebergDB and icebergWD functions. The first row contains different thresholds. The second, third and fourth row lists out the number of seconds required to execute icebergPQ, icebergDB and icebergWD functions respectively.

Table 1: Execution times with different thresholds

Threshold	**1000**	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**	**8000**	**9000**	**10000**
icebergPQ	2.563	1.985	1.922	1.750	1.437	1.453	1.281	1.172	1.297	0.953
icebergD	1.73	1.15	0.96	0.96	0.812	0.95	0.95	0.938	0.922	0.87

B	5	7	9	9		3	3			5
icebergWD	1.750	1.343	1.156	1.079	1.031	0.984	0.969	0.969	0.937	0.906

The above tabulated results are analyzed and the efficiency of the strategy developed is presented in the following graph.

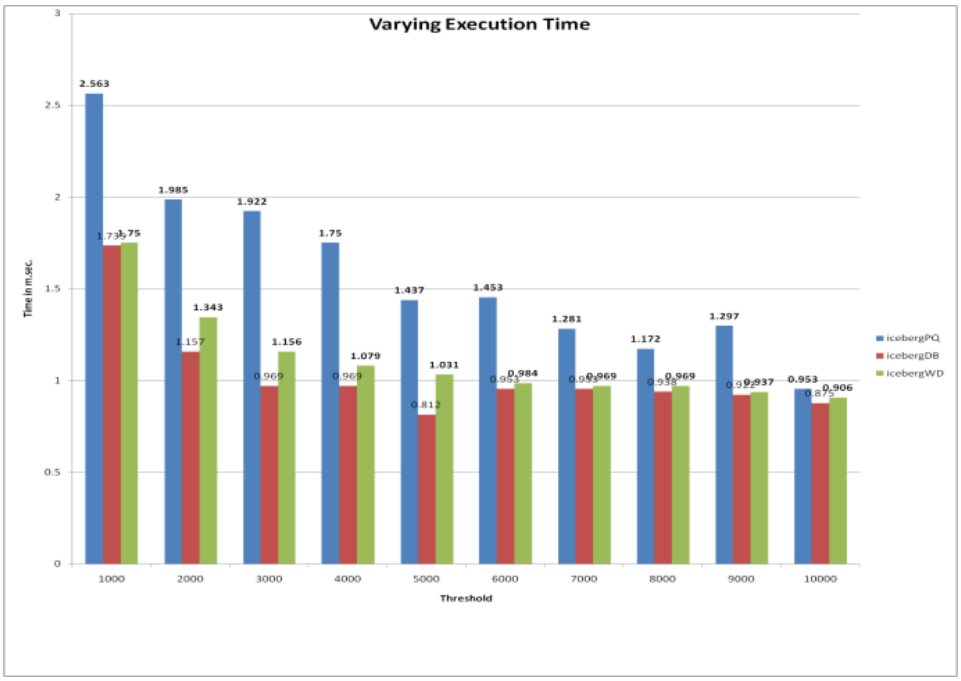

Fig 4: Execution times in icebergPQ, icebergDB and icebergWD varying thresholds.

The graph showing an execution times of three different algorithms by indicting with different colors such as the iceberPQ algorithm i.e., blue color line, green is without high 1's count and red is with high 1's count achieves the best pruning of bitmap vectors compared to both algorithms. The above results are concluded in the next section.

VI. CONCLUSION

This paper presents a new efficient strategy for pruning of bitmap vectors using compressed bitmap indices. Our strategy makes iceberg query evaluation process more effective by conducting the bitwise-AND operation between the initial high 1's count bitmap vectors and thus large number of bitmap vectors gets pruned which in turn reduces the unwanted AND operations.

REFERENCES

[1] D.E. Knuth, "The Art of Computer Programming : A Foundation for computer mathematics" Addison-Wesley Professional, second edition, ISBN NO: 0-201-89684-2, January 10, 1973.

[2] 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.1570-1589

[3] G.Antoshenkov, "Byte-aligned Bitmap Compression", Proceedings of the Conference on Data Compression, IEEE Computer Society, Washington, DC, USA, Mar28-30,1995, pp.476

[4] Jinuk Bae,Sukho Lee, "Partitioning Algorithms for the Computation of Average Iceberg Queries", Springer-Verlag, ISBN:3-540-67980-4, 2000, pp: 276 – 286.

[5] K.Wu,EJ.Otoo,and A.Shoshani, "On the Performance of Bitmap Indices for High Cardinality Attributes", VLDB, 2004, pp: 24–35.

[6] R.Agarwal,T.Imilinski,andA.Swami."MiningA ssociation Rules between Sets of Items in Large databases". In SIGMOD Conference, pages 207-216, 1993.

[7] J.Baeand, S.Lee, "Partitioning Algorithms for the Computation of Average Iceberg Queries", in DaWaK, 2000.

[8] K. P. Leela, P. M. Tolani, and J. R. Haritsa."On Incorporating Iceberg Queries in Query Processors", in DASFAA, 2004, pages 431–442.

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

[10] K.Wu,E.J.Otoo and A.Shoshani. "Optimizing Bitmap Indices with Efficient Compression", ACM Transactions on Database System, 31(1):1–38, 2006.

[11] M.Jrgens "Tree Based Indexes vs. Bitmap Indexes: A Performance Study" In DMDW, 1999.

[12] K.-Y.Whang, B.T.V.Zanden and H.M.Taylor."A Linear-Time Probabilistic Counting Algorithm for Database Applications". ACMTrans.Database Syst., 15(2):208–229, 1990.

[13] Spiegler I; Maayan R "Storage and retrieval considerations of binary databases". Information processing and management: an international journal 21 (3): pages 233-54, 1985

[14] Dr.C.V.Guru Rao and V.Shankar,"Efficient iceberg query evaluation using compressed bitmap indices by deferring bitwise-XOR operations" 3rd IEEE,on Advanced Computing Conference, 22nd &23 rd Feb 2013,New Delhi, India,pp.1374-79.

