A Density based Priority Queue Strategy to Evaluate Iceberg Queries Efficiently using Compressed Bitmap Indices

Vuppu Shankar
Assoc.Prof.,
Department of Computer Science and Engineering,
Kakatiya Institute of Technology & Science-
Warangal,Andhra Pradesh,India-506 015

C.V.Guru Rao, PhD.
Professor & Head
Department of Computer Science and Engineering
S.R.Engineering College
Warangal, Andhra Pradesh, India - 506371

ABSTRACT
In particular, iceberg query is a special class of aggregation query that computes aggregated values upon user interested threshold (T).The bitmap index is a common data structure for fast retrieval of matching tuples from data base table. These resultant tuples are useful to compute aggregations such as SUM, COUNT, AVG, MIN, MAX, and RANK. In this paper, we propose a density based bitmap pruning strategy to evaluate iceberg queries efficiently using compressed bitmap indices. The strategy prioritizes the vectors to be enter in to priority queue by allowing high density of 1's count that achieve optimal pruning effect. Extensive experimentation demonstrates our proposed approach is much more efficient than existing strategy.

Keywords
Data base, Iceberg query, Bitmap index, Priority queue, Bitwise-AND operation, Threshold.

1. INTRODUCTION
The volume of the data base/ and or Data warehouse is increasing enormously as the need of user requirements are increasing day by day. The most aggregated value represent key information of business such as net profit, total sales, net income etc. This is often required by top officials such as executives, managers, administrative officers and computer analysts to make important decisions of an organization. Business Analysts are often responsible to compute and use these aggregated values to compete with present competitive modern business world. Mostly data mining queries are iceberg queries. In particular, iceberg query is a unique class of an aggregation query that computes an aggregate value above user specified threshold (T).

Iceberg queries were first studied in data mining field by Min Fang et.al. [12]. 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 an iceberg query with aggregation function COUNT having the anti-monotone property [18]. Iceberg queries have an intriguing 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.

The present algorithms existing today in the literature are not that much of effective for processing huge data sets, especially for databases have large cardinality attributes. Therefore it is necessary to develop efficient algorithms to process them easily.

The naïve algorithm was answered an iceberg query by referring all tuples from top to bottom in a database over a single pass. The algorithm was initialized a series of counters in memory to store and count the data values of each unique target attribute value for every single pass of data. However, in realistic it is difficult because database table is several times larger than main memory. Hence it requires more than one pass. Then a large response time is needed.

The improvement over the above algorithm was to sort the relation on the disk then passed them into memory to compute an aggregation. Further it selects aggregation values which are greater than a specified threshold. Still the available memory is less than the table size then the data is to be passed over in more number of passes from the disk. Therefore query evaluation consumes long CPU execution time and inordinately large disk requirements.

To speed up the query processing, Bin He et.al. [1] used compressed bitmap indices (Word Aligned Hybrid). A bitmap for an attribute in a table can be viewed as a matrix having r rows consisting corresponding number of tuples and v columns indicating the number of distinct values of an attribute. If there is a bitmap vector in the kth position of the attribute then the element in the matrix is 1 else 0. A bitwise-AND operation was conducted between aligned pair of bitmaps. The 1's count in resulting vector are above threshold are confirmed as iceberg results. The dynamic pruning is best applied for pruning of bitmaps in this algorithm. However the large time was consumed for bitwise-XOR operations indeed.

To further improve the query evaluation time the deferred XOR strategy was proposed in GS [20]. The strategy deferrs bitwise-XOR operations between original and resultant vectors after AND operation. And it discards XOR operations by pruning the bitmaps whose cardinality is below threshold. i.e. no XOR operations were performed on the bitmaps that are going to prune.

Therefore it is understood that from the above algorithms an efficient bitmap pruning strategies are more needed to process large databases especially a database which consists of high cardinality attributes. The more pruning of bitmaps effects on less computation time which increases the speed of answering the iceberg queries.

In this paper, we propose a special bitmap pruning strategy which is based on high cardinality attributes. The higher order cardinality attributes are placed first in the PQs. The strategy takes an opportunity to prune more number of bitmaps which is possible by selecting a large number of aligned positions from huge cardinality. An efficient dynamic pruning step is added as usual to this strategy in order to achieve optimal pruning effect. Hence, the density based pruning strategy is more efficient than existing strategies while answering iceberg queries using compressed bitmap indices.

The experimental results for a large synthetic data set used signify a considerable improvement and is more efficient iceberg query computation.

Section II reviews the related research work on computing iceberg queries and conventional bitmap indices. The proposed research work is described in section III. Section IV gives the details of implementation carried out on the proposed research work. The experiments conducted on the implementation are explained in section V. Section VI analyses the obtained results when experiments were performed. In section VII the research work is concluded followed by bibliography

2. REVIEW OF RELATED RESEARCH

Necessity is the mother of invention. This is the time to provide related research works which is introduced in the above section of this paper. This section is divided into two sub sections to describe briefly about the successful algorithms which are helpful to current investigation. In the first subsection, i.e. A the processing of iceberg queries using tuple-scan based approach is reviewed. We also review the related research using bitmap indices in second sub section i.e. B which is the focus of this paper for optimization of iceberg queries.

In recent times, the evaluation of iceberg queries has attracted researchers significantly due to the demand of scalability and efficiency.

2.1 Iceberg Query Evaluation

Processing of iceberg query was first studied by Fang et.al [12] by extending the probabilistic techniques [11] and suggested hybrid and multi buckets algorithms. The sampling and multiple hash function techniques were used as basic building blocks of probabilistic techniques such as scaled-sampling and coarse-count algorithms. They estimated the sizes of query results in order to predict the valid iceberg results. This improves query performance and reduces memory requirements greatly. However, these techniques erroneously resulted in false positives and false negatives. To recover from these errors, efficient strategies are designed by hybridizing the sampling and coarse-count techniques. To optimize the query execution time of hybrid strategies by extending the linear counting probabilistic algorithm for counting the number of unique values in the presence of duplicates. The linear counting algorithm is based on hashing technique allocates a bitmap (hash table) of size m in main memory. All entries are initialized to "0"s. The algorithm then scans the relation and applies a hash function to each data value in the column of interest. The hash function generates a bitmap address and the algorithm sets this addressed bit to "1". The algorithm first counts the number of empty bitmap entries. It then estimates the column cardinality by dividing this count by the bitmap size m and plugging the result.

Jinuk Bae et. al. [6] had proposed a Partitioning Algorithms for computation of Average Iceberg Queries based on a theorem to select candidates by means of partitioning. The characteristics of this POP algorithm are to partition a relation logically and to postpone partitioning in order to use memory efficiently until all buckets are occupied with candidates. All these techniques were tuple-scan based approaches as they require at least one scan of each tuple in the relation. None of them leverage the bitmap indices for query optimization.

A comparison was presented for Collective Iceberg Query Evaluation (CIQE) [7] using three benchmark algorithms such as Sort-Merge-Aggregate (SMA), Hybrid-Hash-Aggregate (HHA) and ORACLE. CIQE indicates that performance on data sets with low to moderate number of targets, and moderate to high skew was better than SMA but, not for low skew and high number of targets. HHA performance was not robust and quite bad when the number of targets was high. In addition, it has implementation difficulties. There was a considerable performance gap between the online algorithms and ORACLE, indicating a scope for designing better iceberg query processing algorithms.

2.2 Bitmap Indices

The concept of bitmap index was first introduced by professor Israel Spiegler et al [19]. Bitmap indices are known to be efficient in order to accelerate the iceberg queries especially used in the data warehousing applications and in column stores. Model 204 [15] was the first commercial product making extensive use of the bitmap index. This implementation was a hybrid between the basic bitmap index (without compression) and the list of row identifiers (RID-list). Overall performance of Model 204 was similar to the index organized as a B+tree. Early bitmap indices are used to implement inverted files [2]. In data warehouse applications, bitmap indices are shown to perform better than tree based index scheme, such as the variants of B-tree or R-tree [13], [15], [17]. Compressed bitmap indices are widely used in column oriented data bases, such as C-store [14] to improve the performance over row oriented data bases. Word-Aligned Hybrid (WAH) [9] and Byte-aligned Bitmap Code (BBC) [3] are two important compression schemes mostly used in query processing with little effort. More importantly, bitmaps are compressed with BBC and WAH can directly participate in bitwise operations without decompression. BBC is effective in both reducing index sizes and query performance. BBC encodes the bitmaps in bytes, while WAH encodes in words. The new word aligned schemes use only 50% more space, but perform logical operations on compressed data 12 times faster than BBC. The development of bitmap compression methods [9], [3] and encoding strategies [16] further broaden the applicability of bitmap index. Nowadays it can be applied on all types attributes such as high cardinality categorical attributes [11], numeric attributes [11], [16], and text attributes [8]. And it is very efficient for OLAP and warehouse query processing [9], [3].

Hsiao H et.al, [4] proposed an approach of executing the iceberg queries efficiently using the compressed bitmap index and developed an effective bitmap pruning strategy for processing iceberg queries. The index-pruning based approach eliminates the need of scanning and processing the entire data set (table) which speeds up the iceberg query processing.

Bin He et al. [1], proposed an efficient vector alignment algorithm by exploiting the property of bitmap indices. This algorithm completely solves an empty bitwise-AND results problem by pruning the bitmaps whose resultants are zero

vectors. This is the best pruning strategy in the literature available so far and it opens for more scope in evaluation process of iceberg queries.

The deferred XOR strategy in GS [1] further prunes the bitmaps whose cardinality is not more than threshold after every AND operation. This is possible by deferring the XOR operations from the suite of AND operation. Thus deferred strategy was took a scope to prune the bitmaps in order to improve query evaluation process.

The novel idea behind this pruning approach is there is a probability for selection of more aligned positions from large number of 1's count in a bitmaps is possible. This theory is proved mathematically for several cases previously in studies of probability theory for selecting approximate/exact values. This is the best case to reduce time complexity of solving iceberg queries.

The next section presents a proposal to arrange the vectors in the priority queue by high cardinality ordered bitmaps.

3. PROPOSED REASERCH WORK

This section proposes the research work to be carried out on the topic under investigation in the following three sub sections. In the first sub section i.e., A, the block diagram of the proposed research work model is described. In the second sub section i.e. B, an algorithm for pruning the bitmaps by computing latest cardinality of bitmaps for reinsertion in to PQs. The last sub section i.e., C validates the proposed algorithm on sample database.

3.1 The Proposed Block Diagram

In fig, the DB indicates database which is input to the proposed algorithm. The PQA and PQB in the diagram denote priority queues A and B respectively. The priority queues are commonly used to place the bitmap vectors prioritized by latest cardinality of bitmaps. The AND block is specified to perform the bitwise-AND operation between pair of bitmap vectors chosen from PQA and PQB. The diamond shaped box used to check whether the ones count of resultant vectors are above or below threshold. If it is 'Y' send this pair to iceberg result set and 'N' send to computational block. The rectangular computational block is used to compute the latest counts of the bitmap vectors that are inputted by AND operation. The outputs of computed values are inputted for once again checking of whether these counts are above or below threshold using diamond shaped check box. Based on the output of check box the vectors are going to be prune or reinsert into priority queues. Then with all the reinserted vectors with its latest counts is subjected to realize the priority queues are restructured after each AND operation. The same process is continued till the end of either of these queues becomes empty.

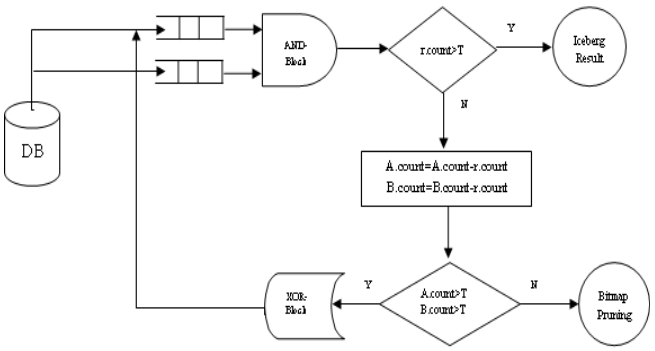

Fig.; 1.Proposed Block diagram for research model

3.2 Proposed Algorithm pruning of bitmap vectors using priority queues with high density counts

An algorithm is proposed in this subsection to evaluate an Iceberg query by pruning the bitmap vectors dynamically using high counts:

IcebergDQ (attribute A, attribute B, threshold T)

Input: {attribute A, attribute B, threshold T}

Output: Iceberg results

```
1: PQA.clear, PQB.clear
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: a, b = nextAlignedVectors (PQA, PQB, T)
12: while a <> null and b<> null do
13:         PQA.pop
14:         PQB.pop
15:         r = a and b
16:         if r.count >= T then
17:           Add iceberg result (a. value, b.value, r.count) into R
18:         a. count= a. Count- r.count
19:         b.count= b.count- r.count
20:         if a. count >= T then
21:                   a = a XOR r
22:                   PQA.push (a)
23:         if b.count >= T then
24:                   b = b XOR r
25:                   PQB.push (b)
26: a, b = nextAlignedVectors (PQA, PQB, T)
27: return R
```

The above IcebergDQ algorithm is described in a density priority queue strategy to GS [20]. The algorithm operates in two phases. In the first phase, it prioritizes the bitmap vectors in PQ by density of 1's count. The function BIT1_COUNT generates the count of 1's in a bitmap vector which is defined as density of that vector (from line number 1 to 9). In second phase, the nextalignedvector function returns all aligned vectors pair from the inputted bitmaps . Then an Iceberg query is processed by conducting bitwise-AND operations between vectors that are chosen from each respective PQs as top positioned. After every AND operation, the 1's count in resultant vector is examined to determine whether this count is above threshold or not. If it is above T, that vector pair is added to iceberg result set along with its count value in line number 17 of alg. Then the original vectors are decreased by r's count. The XOR operations are deferred to conduct immediately after AND operation such that no XORing is taken place for the bitmaps whose 1's count is below threshold that is shown in line numbers 21 and 24 of alg. After XORing, the bitmaps are updated. If the cardinality of updated bitmaps are above threshold will be re entered in to respective PQs. The rest of the bitmaps are pruned out. The same process is repeated until either of the PQs becomes empty.

3.3 Validation of Density Priority queue strategy on sample database

This subsection demonstrates the validity of the proposed density based queue strategy and evaluates an iceberg query having two aggregate attributes with COUNT function.

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

Fig2: An Iceberg query with count function

A	B	C
A2	B2	1.23
A1	B3	2.34
A2	B1	5.56
A2	B2	8.36
A1	B3	3.27
A2	B1	9.45
A2	B2	6.23
A2	B1	1.98
A1	B3	8.23
A2	B2	.11
A2	B1	3.44
A3	B1	2.08

Database table(R)

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

Bitmap Index -A

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

Bitmap index-B

Fig. 3.: Sample data base table (R) and Bitmap indices of an aggregate attributes A and B.

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 2 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.

The bitmap vectors are entered into respective priority queues say PQA and PQB in the high order of their density of 1 bits are as A2, A1 and B1, B2, B3. Here, the vector A3 is not been entered because its density is less than threshold. Hence it is directly pruned by monotone property of iceberg query. The top positioned bitmap vectors A2: 101101110100 and B1: 001001010011 are chosen from PQA and PQB respectively. Now the vectors are subjected to verify the alignment between them. If they are aligned, bitwise-AND operation is conducted, and obtained resulting bitmap vector 'r' as 001001010000. We compute the density in the resulting vector 'r' as 3 and compare with T. This is above threshold (where T is equals to above 2). Therefore, this pair of bitmap vectors A2 and B1 is identified as iceberg result and added to the iceberg result set along with its count value. Further these vectors are also sent to the computational block to compute the latest density by subtracting the density of result vector 'r' from each vector. The same is formulated as follows: A2.count = A2.count- r. count which is equals to 4 and B1.count = B1. count - r. count is 2. Then the count of the both vector are compared with threshold. The vector A2 is passing the threshold where as the vector B1 do not. That means the future reference of vector B1 is no need. Hence, it is pruned. Whereas the vector A2 is reinserted into PQA with its latest count by changing the aligned 1's as 0 by conducting the bitwise-XOR operations between vector A2 and resultant vector 'r' . Therefore PQA is required to restructure and arranging them up on latest counts. Then vectors in PQA and PQB are modified as A2 and A1 & B2 and B3. The same process is continued until either of priority queues becomes empty.

The next section describes the implementation details of all the modules in the java platform.

4. IMPLIMENTATION

This section describes the various modules that were proposed in the previous section and the details:

4.1 Generate Bitmap

The first module Generate Bitmap responses to input query shown in fig 2. This generates the equivalent bitmap values in terms of sequence of 1 and 0 bits of each attribute value specified in the iceberg query (IBQ). Then the generated bitmaps are stored in the in the series of arrays in compressed format. In the process the bitmap vectors are used existing word aligned hybrid compression scheme and arranges all the bitmap vectors into compressed words. These compressed bitmaps are placed in PQs ordered by high cardinality computed through BIT11-COUNT( ).

4.2 Next Aligned Vectors

This module nextAlignedVectors is used to align two bitmap vectors from each of the priority queue with first 1-bit positions by comparing them by using first 1 bit position algorithm. If the bitmaps are found as aligned returned to main algorithm for further processing. Otherwise, next bitmap vector is chosen from PQs to repeat the same process.

4.3EfficientBitmapPruningbyprioritqueues with high 1's count

This is the main part of algorithm EfficientBitmapPruning module accepts two aligned bitmap vectors as input. A bitwise-AND operation is performed on these aligned vectors. The resulting bitmap vector 'r' is examined for the count of number of 1's. If the number of 1's is greater than T then the pair is added to the iceberg result set. Otherwise bitmap vectors are pruned. Further the number of 1's in resulting vector 'r' is now compared with 1's count in v1, v2 and if the count is greater than T then a bitwise-XOR operation is performed on the same and reinserted in to corresponding PQs prioritized by its updated density value. Else the bitmap vectors are pruned.

The above modules are implemented using JAVA for the purpose of experimentation. The next section explains the experiments conducted on this implementation.

5. EXPERIMENTAL EVALUATION

5.1 Experimental set up

This section describes the experiments carried out on the implementation described in the previous section under a specified iceberg threshold values.

The experiments are conducted on a following computer system with i3 processor of 2.4 GHz, 1.0GBmain memory running on Micro soft Windows XP ver., and all algorithms are implemented in java platform.

The experiments are carried out repeatedly a minimum of 3 to 4 times of same threshold with synthetic data set consists of 1.2 millions in zipfian distribution.

The experiments are conducted for high threshold values such as from 1000 to 10000 and the cardinality is taken as 99999 unique valued attributes.

5.2 Experimentation

First, the iceberg query is responsible to select the similar records with A and B aggregate attributes from the database table R which are having a threshold value ranging between 1000 and 10000. Then, the experiment is to be conducted by firing an iceberg query as stated in Fig .2 on the database table which consists of 1.2 millions of tuples with two attributes A and B and COUNT as an aggregation function. The first function i.e.

GenerateBitmaps accepts all these tuples as input. This function first produces one bitmap vector of each distinct value of an aggregate attribute and aligned the words in a compressed model. Then the vectors are arranged with high density 1's count order in PQA and PQB through special function called PQ. These two Density priority queues are given as input to EfficientBitmapPruning function which repeatedly calls the NextAlignedVector function and First1bitposition function as an internal to it in to main program until any one of the Density priority queue becomes empty. Each time the NextAlignedVector function returns two top most aligned bitmap vectors from each Density queue to the main program i.e.,efficienticebergqueryevalautionwithDQs. From all such records which are having a COUNT value greater than 1000 to 10000 are generated as an output. The experiment is repeated for different iceberg thresholds by keeping the same number of tuples in a database table and noted the results with density PQs and without density PQs and as well as iceberg PQ.

The next section lists out the results of the experimentation and analyses them.

6. RESULTS AND ANALYSIS

This section describes the results obtained in our experiments conducted in the previous section and are tabulated in table 1.

The first row in table 1 indicates different thresholds. The second and third rows correspond an execution time made in icebergPQ, icebergDB (density PQ) and icebergWD(without density PQ) 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 and third 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
icebergPQD	2.563	1.985	1.922	1.750	1.437	1.453	1.281	1.172	1.297	0.953
icebergDB	1.735	1.157	0.969	0.969	0.812	0.953	0.953	0.938	0.922	0.875
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.

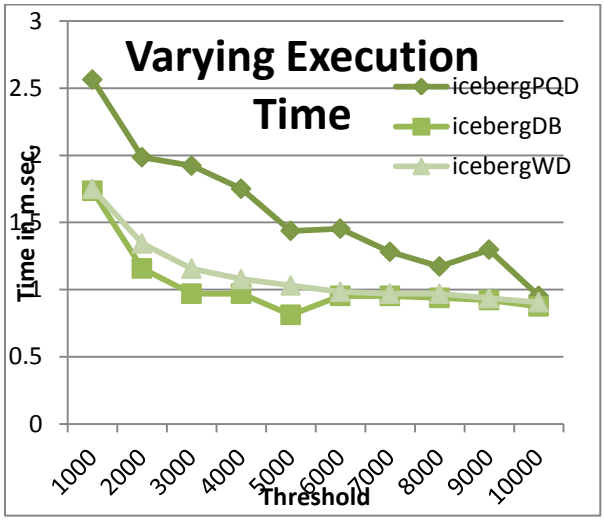

Fig 4: Execution times in icebergPQ, icebergDB and icebergWD varying thresholds.

The above results are concluded in the next section.

7. CONCLUSION AND FUTURE SCOPE

This paper presents an efficient bitmap pruning strategy which is based on order of high cardinality in PQ for processing an iceberg query using compressed bitmap indices. The strategy allows flow of vectors to enter in to PQs on high 1' count achieved more benefit for large pruning of bitmaps. The pruned vectors inherently improves response time of answering IBQs. There are several research directions in optimizing the iceberg query such as preprocessing of unwanted bits from bitmaps, exclusion of fruitless bitwise-AND operations and optimal priority queue management.

8. ACKNOWLEDGEMENTS

Our thanks to my collogue Mr.B.Hanmanthu, Assistant Professor ,Department of CSE KITS-warangal. and my M.Tech.(SE) student Mr.Sandeep Reddy who have contributed their time to materialize this document.

9. REFERENCES

[1] 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

[2] 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.

[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] Hsiao H, Liu Z, Huang Y, Chen Y, "Efficient Iceberg Query Evaluation using Compressed Bitmap Index", in Knowledge and Data Engineering, IEEE, Issue: 99, 2011, pp:1.

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

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

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

[8] 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.

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

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

[11] 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.

[12] M.Fang, N.Shivakumar, H.Garcia- Molina, R.Motwani and J.D.Ullman."Computing Iceberg Queries Efficiently". In VLDB, pages 299–310, 1998.

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

[14] M.Stonebraker, D.J.Abadi, A.Batkin, X.Chen, M.Cherniack, M.Ferreira, E.Lau,A.Lin, S.Madden, E.J.O'Neil,

P.E.O'Neil, A.Rasin, N.Tran and S.B.Zdonik.C-Store: "A Column-oriented DBMS". In VLDB, pages 553–564, 2005.

[15] P.E.O'Neil."Model204 Architecture and Performance". In HPTS
Pages 40–59, 1987.

[16] P.E.O'Neiland D.Quass. "Improved Query Performance with Variant Indexes". In SIGMOD Conference, pages 38–49, 1997.

[17] P.E.O'Neil and G.Graefe. "Multi-Table Joins Through Bitmapped Join Indices". SIGMOD Record, 24(3):8–11, 1995.

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

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

[20] 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.

AUTHOR'S PROFILE

Vuppu Shankar obtained his Bachelor's degree in Computer Technology from Nagpur University of India. Then he obtained his Master's degree in Computer Science from JNTU Hyderabad, and he is also life member of ISTE. He is currently an Associate Professor at the Faculty of Computer Science and Engineering, Kakatiya Institute of Technology & Science (KITS), Kakatiya University-Warangal. His specializations include Data mining and Data warehousing, Databases and networking. His current research interest in computation of an iceberg cubes and evaluation of iceberg queries efficiently.

Dr.C.V.Guru Rao obtained a Bachealor's in Electronics & Communications Engineering from Nagarjuna University, Guntur, India during December 1981. He is a Double Postgraduate in Electronic Instrumentation as well as Information Science & Engineering from Regional Engineering College, Warangal and Motilal Nehru Regional Engineering, Allahabad. Later he was awarded a Ph.D (Computer Science & Engineering) from Indian Institute of Technology, Kharagpur during June 2004. He is currently working as Professor and Head, Department of Computer Science & Engineering at SR Engineering College, Warangal. His research interests are Computer Science & Engineering with specialization in VLSI and Embedded Systems, Software Engineering, Data Mining and Warehousing. He had published 72 research papers in Journals/Conferences of national/international repute. He is a Life Member of Indian Society for Technical Education, Instrument Society of India. He is also the member of Institution of Engineers (India), IETE, MIEEE-CS (USA), MACM and MCSI. He was elevated as Senior Member of IEEE (USA).