Journal of Engineering Science and Technology Review 12 (6) (2019) 202 - 207

JOURNAL OF
Engineering Science and Technology Review

Research Article

Efficient Computation of Min & Max Iceberg Queries Using Value based Property

Pallam Ravi[1,2],* and D. Haritha[1]

[1]*KLEF,Guntur,India*
[2]*Anurag Group of Institutions, Hyderabad,India*

Received 21 November 2017; Accepted 29 November 2019

Abstract

Data mining and Data warehousing systems use aggregate queries (Iceberg queries), in turn, compute the results based on the constraints on aggregation functions above the user provides threshold. The aggregate functions, MIN and MAX, have anti-monotone property. Based on this property, the computation of the candidate set can be reduced. In this paper, we compute iceberg queries having MIN and MAX without computation of candidate set as per anti-monotone property. We call it as "Value-based Property". Further, we proposed two algorithms namely range and equal algorithms, which compute Min & Max Iceberg query efficiently. Our experiments reveal that the proposed algorithms are 90% more efficient compared to the traditional algorithms and they also work in a distributed environment.

Keywords: Iceberg queries, Bitmap Index, Aggregate Function, Value-based Property.

1. Introduction

Nowadays businesses need to discover hidden trends and relationships from the data. This knowledge helps them in making decisions better so that they can sustain in the competitive market. Data summary using aggregate functions gives more important information to find trends and relationships. High aggregate values give precise information. So, we need to find the data for which the aggregate value is above the user-specified threshold.

Aggregate functions like COUNT, SUM, AVG, MIN, MAX, MEDIAN, RANK and TOP(K) are used in Data Mining and Data warehousing queries for extracting knowledge. A query is needed to compute aggregate functions over the attributes, and it produces the result set based on user constraint. This query is known as Iceberg query [1]. The general form of the Iceberg query is as follows:

```
SELECT A1, A2, An FROM R
GROUP BY A1, A2, An
HAVING agg fun() op T
```

In above Iceberg query, R is relation with attributes (A , A , A), op is operand (<,>,==, ≥,≤.etc) and T is user specified threshold.

Example1: A college principal wants the information about all branches which have exactly 50 admissions. Then the query can be written as:

SELECT Branch Name, COUNT( ) from Admission GROUP BY Branch Name HAVING COUNT()==50

In this Iceberg query, COUNT( ) is an aggregate function and 50 is the user-specified threshold.

In order to compute Iceberg queries over large data, the computation demands more main memory as there are many distinct group aggregate values needed to be stored. To maintain a counter in main memory for each of distinct aggregate values, it consumes time more as the result-set of Iceberg queries is a collection of small sets (this is a tip of Iceberg query).

The general strategies to execute iceberg queries are:

1) Sort the aggregated attributes, calculate the aggregate function and finally, apply threshold constraint. Accord- ing to [11], [25] and [15], all existing modern databases (For example: Oracle, SQL Server, PostgreSQL, Sybase, and column-oriented databases including MonetDB, LucidDB, and Vertica) follow the same aforementioned procedure.
2) Maintain a counter for the each distinct group value.

The first method was not efficient for large datasets because sorting procedure requires many swapping and comparisons. The second method needs to maintain many counter variables. For example, the aggregated attributes namely A, B, C whose cardinalities are 1000, 1000, and 1000 respectively. Then the number of counter variables needed is 1000*1000*1000=1000 million. It is difficult to maintain these counter variables in main memory especially when the dataset is large.

In literature, the tuple scan-based approach [4], which focuses on reducing the number of passes when data size is large, requires at least one table scan to read data from disk. It takes time to answer Iceberg queries more because it does not leverage the property of Iceberg queries. Fern et al. [10] designed a two-level index to process the Iceberg queries which suffer from the massive empty bitwise-and, which was addressed by [26] using dynamic pruning algorithm.

Different index techniques used for improving the query execution indexes are: value list, projection index, and bitmap indexes. For MIN and MAX Iceberg queries, bitmap index gives the best performance [17]. In this paper, we used

*E-mail address: satishpallam@gmail.com
ISSN: 1791-2377 © 2019 School of Science, IHU. All rights reserved.
doi:10.25103/jestr.126.25

bitmap indexes. Each bitmap index represents a sequence of bits: 1 and 0s, bit 1 in nth position represents the presence of attribute value in nth record and bit 0 in nth position in the sequence represents the absence of attribute value in nth record. In table- 1, an attribute distinct value A_1's bitmap sequence is 1010010.

Some aggregate functions will have anti-monotone property [23]. The anti-monotone property is defined as follows: if sub set does not satisfy the constraint then its superset also never satisfies the constraint. Like SUM, MIN, MAX and COUNT, in general, Iceberg queries have Equal[24] and Range[23] type queries, with use of bitmap indexes. To answer Rang query in example, it needs six BITWISE AND operations namely **(A1, B1), (A1, B2), (A1, B3), (A2, B1), (A2, B2) and (A2s, B3).**

But, it is not effective because high cardinality aggregate values have large number of unique values. When we use anti-monotone property[2], it reduces the number of BITWISE AND operations to two only, namely (A2 ,B1) and (A2 ,B3). We find MIN and MAX aggregation with monotone property and it has special property which we identified called value- based (see Section 3). By using this value-based property, there is no need to perform bitwise operations among bit map indexes of A and B. It requires selection operation on mark attribute to answer the iceberg query.

The compression bitmap methods [24],[3],[8] and encoding strategies [19] have further broadened the application of bitmap indexes. Bitmap indexes are can be applied to all types of attributes (For example, high-cardinality categorical attributes[23], text attributes[20] and numeric attributes[23],[19]). These representations improve the performance of bitwise operations.

The studies [24] and [3] revealed that the bitmap indexes occupy less space than the raw data and best for range queries[23] and keyword queries[20] and equal queries[24]. It gives better query computation. The bitmap index can be supported by DBMS including Oracle, Sybase etc. The column-databases systems (For example, Vertica, C-store[21], LuciDB) have default bitmap index option.

Word-Aligned Hybrid(WAH) [24] and Byte-aligned Bitmap Code(BBC) are compression schemes, which used query processing without decompression to improve the bitmap operations.

For numeric attributes, bitmaps are created by the values represented as n-bit binary numbers then slice the bits positions of 2^0,2^1,2^2...2^n into the bitmap. In Table-1, the value at 2^0 position bitmaps is 1001011 and 2^1 position bitmap is 1110101.

2. Related Work

The first work on iceberg queries proposed by Fang et al.[9] included various combinations of sampling and hashing techniques by extending the probabilistic techniques hybrid and multi-bucket algorithms in [22]. The disadvantages of these methods are (a) the result set contains" false positive" and "false negative", (b) it is not effective because it uses the tip of Iceberg queries and (c)it does not work for MIN, MAX, and AVERAGE aggregate functions. To solve these problems, Average Iceberg Queries design partitioning algorithm [4] is the better choice.

The aim of Iceberg cube computation algorithm is to minimize the shared computation. To reduce the time of cube generation, one can select a proper order of computing an aggregate by combining the aggregate attributes. It is studied in [6],[1],[12], and [10]. An Iceberg query has the different goal of speeding up the processing of single query.

To improve the performance of MIN, MAX and AVERAGE aggregate functions, K.P.Leela [2] proposed SHA and HHA algorithms which ,in turn, used two phase multi-way merge sort[3]. It computed the aggregate function in merge phase, applied the constraint with highly skewed distribution at- tributes and low and high queries. In [26], the authors proposed dynamic pruning algorithm to compute the Iceberg queries by using bitmap index data structure which, in turn, reduced the number of BITWISE AND operations and eliminated empty BITWISE AND operation for grouping the target attributes by using anti-monotone property and vector alignment algorithm[26]. But it did not use Value-based Property of MIN and MAX functions to answer MIN and MAX Iceberg queries. For evaluating non anti-monotone aggregate Iceberg queries, one can use bitmap number [27]. It can help in reducing the sorting time as it requires only one scan of data and there is no need of computing intermediate candidate sets and aggregate values, which do not use anti-monotone and Value-based Property of MIN and MAX aggregate functions to answer the MIN& MAX Iceberg queries.

These algorithms require various operations including sorting, candidate set computation and sampling. By using these operations, we can not compute result set directly. We can compute result directly by sorting bits. This computation takes many comparisons which is time consuming. We can reduce the time by using "Value-based" property of MIN and MAX functions.

3. Value-Based Property

We identify the special property of MIN and MAX aggregate functions. We called it as value-based. Value-based property is defined as follows: if an attribute (A) value in the record satisfies the aggregate condition then it is included in the result set.

We define Value-based property as follows:

*Let r is record of relation R, If **r[A]** op T is True*
*Then **MIN (A)/MAX(A)** op T is also True so add r to the result set,op is operator (>, <, ==,... etc)*

In literature, as per our knowledge, there are no algorithms using "Value-based" property for answering Iceberg queries constrained on MIN and MAX aggregate functions. By using this property, we can eliminate intermediate computations of aggregate values and candidate set generation, which we explained in Example 2.

Example 2: Teacher wants to find out names of the subjects which are in paper-1 and paper-2 sets and in which students are getting marks > 20.

SELECT paper-1, paper-2, MAX(marks) FROM R GROUP BY paper-1, paper-2 HAVING MAX(marks)>20

Here marks represents total marks of paper-1 and paper-2.

A. Value-based method

Find out marks>20 records in relation Table 1, the result set Table 3, is subset of records having marks>20 in Table 2.

Table 1. Relation Table :R

Roll no	Paper-1	Paper-2	Marks
1	A1	B1	19
2	A2	B3	22
3	A1	B2	15
4	A2	B1	25
5	A2	B3	18
6	A1	B1	17
7	A2	B1	23

With bitmap indexes, it is needed to compute MAX aggregation value (at most of the Paper-1 cardinality * Paper-2 cardinality) six times and the computation requires BITWISE AND between aggregate attributes. By using value-based property, there are no intermediate computations of MAX and MIN aggregate value

Table 2. Records having marks>20

A2	B3	22
A2	B1	25
A2	B3	22
A2	B1	25

Table 3. Results records

A2	B3	22
A2	B1	25
A2	B1	23

Theorem 1:

S = x / x,∀ x ∈ r[a] > T

R = x / x, ∀ x ∈ MIN(MAX)(a) > T Then R ⊂ S .

We proposed a method called Value_Min_Max (see Algorithm 1) for avoiding the computations needed for intermediate computation of aggregate values and eliminating BITWISE AND operations to find candidate sets.

Algorithm 1 Value Min Max

```
1: Input: Aggregate constraint(T); Bitmap slices
   represent- ing aggregate attribute values
2: Output: Records satisfying the aggregate constraints
3: Select the records with aggregate attributes satisfying
   the aggregate constraint (MIN/MAX)
4: compute Aggregate function (MIN/MAX)
5: Eliminate duplicate records
6: Return records
```

In step-2 (line 4), we need to combine same district target attributes sets. it requires small amount of time because Iceberg query results set size is 5% to 10% of a distinct group of data. So, it is computed with less amount of main memory and CPU time.

By using the bitmap numbers generated by bitmap numbers algorithm [5], which is described by Algorithm 2, we can improve the step-1&2 in Value Min_Max algorithm. We find out the records satisfying the constraints by simple BITWISE AND and NOT operations and compute aggregate function effectively. The aggregate conditions are Equality condition and Range condition.

1.EQUALITY AND RANGE QUERIES

Equality and not Equality: MIN(A=T) , MAX(A=T), MIN(A!=T) = NOT(MIN(A=T))

MAX(A!=T) = NOT(MAX(A=T))

Range: MIN(A >= T), MAX(A >= T),MIN(A <= T), MAX(A <= T), MIN(A < T),

MAX(A > T).

The other range conditions are converted into another form like below

Algorithm 2 bitmap numbers

```
1: Input: aggregate attributes values($v_1,v_2,...,v_n$)
2: Output: bitmaps ($P_1,P_2,...,P_k$)
3: for i=1 to n do
4:     $v_i=(b_k,...,b_2,b_1)$
5:for j=1 to n do
6:     if $b_i$==0 then
7:     $P_i$=0
8:     else
19:    $P_j$=1
```

MIN(A > = T) = MIN(A > T-1)

MIN(A <= T) = NOT(MIN(A > T))

MIN(A < T)= NOT(MIN(A > -T)) or NOT(MIN(A > T-1))

It is also applied to MAX function.

We develop algorithms namely Rang_op_t and Equal_op T to compute Range (For example, A>T) and equal queries. These two are explained in next two sections.

A. Equality Condition

Equality conditions are computed as: Let T=($b_k,...,b_2,b_1$), P = sequence of all 1s with length n(number of records) and aggregate attribute values represents bitmaps slice those bits position are 2^1, 2^2,...,2^k ($P_k,...,P_2,P_1$), BITWISE AND between bitmaps, change P based on threshold bits values, if bi=0 then perform NOT of P else simple use P only, for example, T=101 then perform P_3^P_2^P_1. finally, we get n bits sequence of 1s and 0s, bit 1 represents n^{th} record have equal value, these processes are represented in Equal_op_T algorithm (See Algorithm 3).

Algorithm 3: Equal op T

```
1: Input:   aggregation   attribute values   bit
      maps slices($P_k,...,P_2,P_1$ )
2: Output: sequence of 1s, 0s, $n^{th}$ bit 1 of length n, $n^{th}$
   record have equal value.
3: for i=1 to n do
4:     if $b_i$==0 then
5:     else
6:     $P$ =$P$ AND $P_i$
```

Example-3: Teacher want find out subject names in paper-1 and paper-2 where students getting max marks of 20.

SELECT paper-1, paper-2, MAX (marks) FROM R GROUP BY paper-1, paper-2 HAVING
MAX (marks) == 22.
T = 22 (10110) = $b_k, ..., b_2, b_1$

We compute P value using following expression
Finally we get P = (0100000), in tableIV,P have bit 1 at position record id=2

B. Range Condition

Range Iceberg queries are computed as follows. Let T=$(v_k, v_2, v_1)_2$, P= sequence of all 0s with length n(number of records) and aggregate attribute values represent bitmaps slice those bits position are $2^1, 2^2, .2^k$ (P_k, P_2, P_1), it maintain P_I and P_T two intermediate bitmaps then perform P_I = P_I AND P_K until find most significant 1 bits in threshold value bits
(V) then based on V bits if V_i =1 perform P_T=P_T AND P_i
, if V_i =0 P_U =P_T AND P_i and P_I =P_I OR P_U , these process represent in Range_op_T algorithm.

Algorithm 4 : Range_op_T

```
1: Input: n:number of records; $P_k,...,P_2,P_1$: Array of
   bits on which aggregate value is computed; $P_I$ :
   sequence of 0s; $V_1,V_2,...,V_k$ binary represent of
   threshold value T
2: Output: PI: array of bits (0,1), 1 indicates > T
3: k=1
4: while $V_k$!=0 do
5:     $P_I$ =$P_I$ AND $P_k$
6:     k=k+1
7: $P_T$ =$P_k$
8: for i=k+1 to n do
9:     if $V_k$==1 then
10:    $P_T$ =$P_T$ AND $P_i$
11:    else
11:    $P_u$=$P_I$ AND $P_i$
13:    $P_I$ =$P_I$ OR $P_u$
```

Example 4:

SELECT paper-1, paper-2, MAX (marks) FROM paper-1, paper-2 HAVING MAX(marks) > 20
T=20 (10100) = V1, V2, V3, V4, V5

Finally bit 1 in P_I represent all records which satisfy the conditions which shown in table IV.

5. Application On Large Data

For large data, bit slice vectors are not able to be stored in main memory. Our algorithms can work on large data. The framework of proposed method is depicted by Fig.1.

As shown in Fig.1, the stage-1 partitions the data so that each partition can be stored in main memory. After partitioning data, our algorithms are run on each partition separately in stage-2. Finally, Stage-3 combines all results produced from Stage-2.

5. Experiment

In this section, we present the results of experiment conducted and compare them with IcebergDP algorithm. We tested the proposed algorithm using synthesized data of a different number of tuples and different values ranges. But, Iceberg queries, in general, are not affected by value distributions[11],[25],[15].

We experimented the proposed Equal and Range Algorithms with different volumes of data (in terms of tuples), threshold (in terms of bits to represent it) and Attribute value range(in terms of bits). The proposed algorithm scale with respect to data size and its independence to distinct groups, distinct values range and number of attributes in a relation.

Table 4. Result Of Equal Query

	Marks	**BIT ARRAY P5,P4,P3,P2,P1**	**P**	**b1=0 P=P AND P1**	**b2=1 P=P AND P2**	**b3=1 P=P AND P3**	**b4=0 P=P AND P4**	**b5=1 P=P AND P5**
1	19	10011	1	0	1	0	0	0
2	22	10110	1	1	1	1	1	1
3	15	01110	1	1	1	1	0	0
4	25	11001	1	0	0	0	0	0
5	18	10010	1	1	1	0	0	0
6	17	10001	1	0	0	0	0	0
7	23	10111	1	0	0	0	0	0

Table 5. Result of Range query

Id	**Marks**	**BITARRAY P_1, P_2, P_3, P_4, P_5**	**P_I**	**V_1=1 P_T = P_1**	**V_2=0 P_I = P_I OR (P_T AND P_2)**	**V_3=1 P_T = P_T AND P_3**	**V_4=0 P_I = P_I OR (P_T AND P_4)**	**V_5=0 P_I = P_I OR (P_T AND P_5)**
1	19	10011	0	1	$0 \vee 1 \wedge 0=0$	$1 \wedge 0 = 0$	$0 \vee 0 \wedge 1=0$	$0 \vee 0 \wedge 1=0$
2	22	10110	0	1	$0 \vee 1 \wedge 0=0$	$1 \wedge 1 = 1$	$0 \vee 1 \wedge 1=1$	$1 \vee 1 \wedge 0=1$
3	15	01110	0	0	$0 \vee 0 \wedge 1=0$	$0 \wedge 1 = 0$	$0 \vee 0 \wedge 1=0$	$0 \vee 0 \wedge 1=0$
4	25	11001	0	1	$0 \vee 1 \wedge 1=1$	$1 \wedge 0 = 0$	$1 \vee 0 \wedge 0=1$	$1 \vee 0 \wedge 1=1$
5	18	10010	0	1	$0 \vee 1 \wedge 0=0$	$1 \wedge 0 = 0$	$0 \vee 0 \wedge 1=0$	$0 \vee 0 \wedge 0=0$
6	17	10001	0	1	$0 \vee 1 \wedge 0=0$	$1 \wedge 0 = 0$	$0 \vee 0 \wedge 0=0$	$0 \vee 0 \wedge 1=0$
7	23	10111	0	1	$0 \vee 1 \wedge 0=0$	$1 \wedge 1 = 1$	$0 \vee 1 \wedge 1=1$	$1 \vee 1 \wedge 1=1$

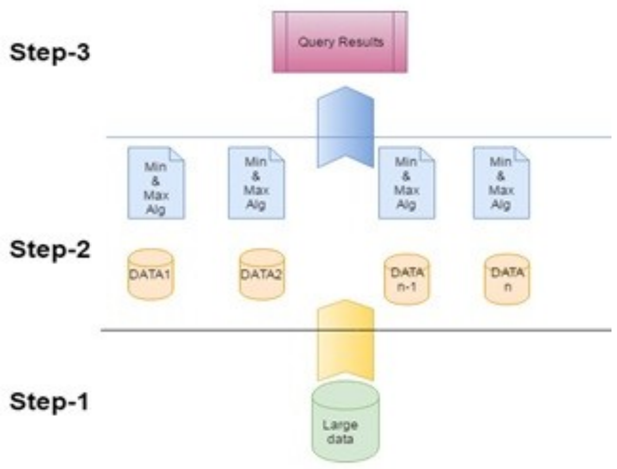

Fig. 1 Framework for proposed method

A.Experimental Setup

The experiments are conducted on a machine with an Intel i3 processor and 4 GB RAM running on Windows 10. The algorithms are implemented in Java.

In our experiment, we assume that binary representation (bit slice) of the aggregated attributes have built-in offline. This is a reasonable assumption because it takes less than one minute in building the binary representation of 10 million tuples.

B. Performance of Range and Equal algorithm on different Number of tuples

We experimented with 2,4,6,8 and 10 million tuples of synthesized data using constraint threshold and aggregate attribute size. For 10 million tuples, we could execute the query in 16 milliseconds, for equal type query 17 milliseconds, for Range type query which is more efficient than IcebegDP algorithm [26] which takes 18 minutes. It means that the total computation time is drastically reduced from 18 minutes to 17 mile seconds. This is because of there is no need of computing intermediate aggregate values and used BITWISE AND, OR operations to answer Iceberg queries depicted by Fig 2 & fig 3.

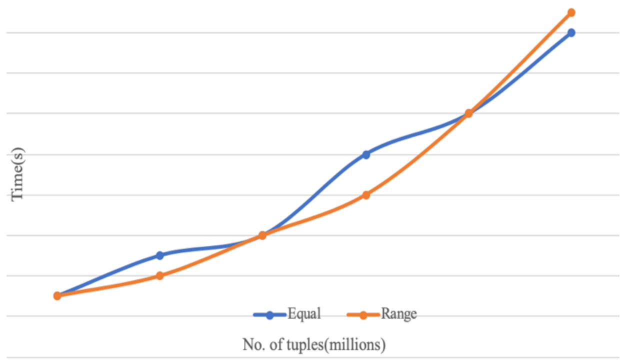

Fig.2. Perfomance graph for no. of tuples (millions).

C:Performance of Range and Equal algorithm of different thresholds

In this study, we experimented with 10 million tuples of data keeping threshold value constant. With 20,40,60 and 100 bits assigned to aggregate attribute values, we executed range query in 17 mile seconds which is constant for all values of aggregate attribute, but equal Iceberg query is linearly depend- ing on number of bits which is executed in 63 milliseconds for 100 bits used for Aggregate Attribute values, which is depicted by Fig. 4.In fig 5 Performance graph in terms of aggregate different threshold values of range and equal algorithm using 10 million records ,keep threshold value constant ,change no of tuples ,the performance shows in fig 6.

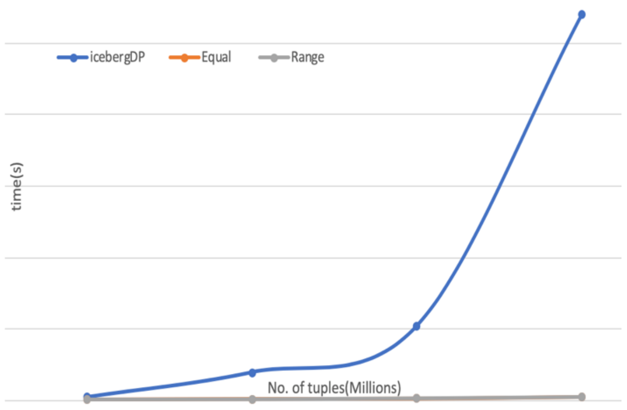

Fig. 3 Comparison graph of proposed and existing algorithms

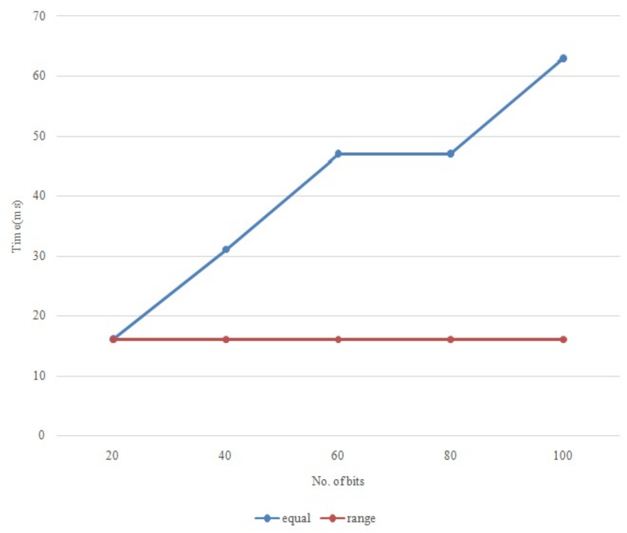

Fig 4. Performance graph in terms of aggregate attribute(no of Bits) values of range and equal algorithm

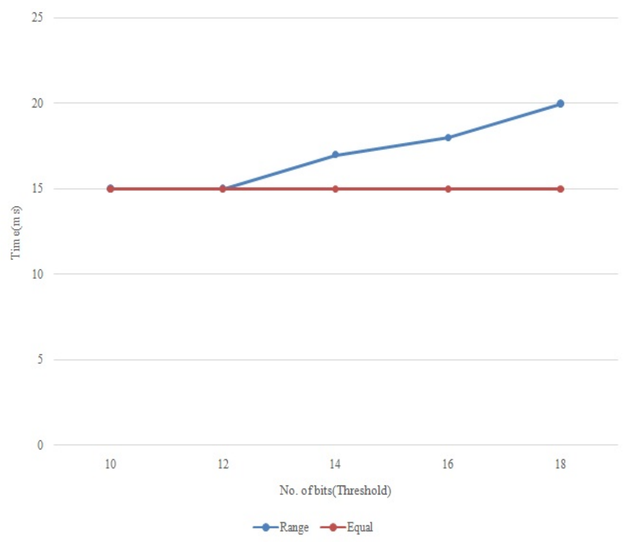

Fig 5. Performance graph in terms of aggregate different threshold values of range and equal algorithm

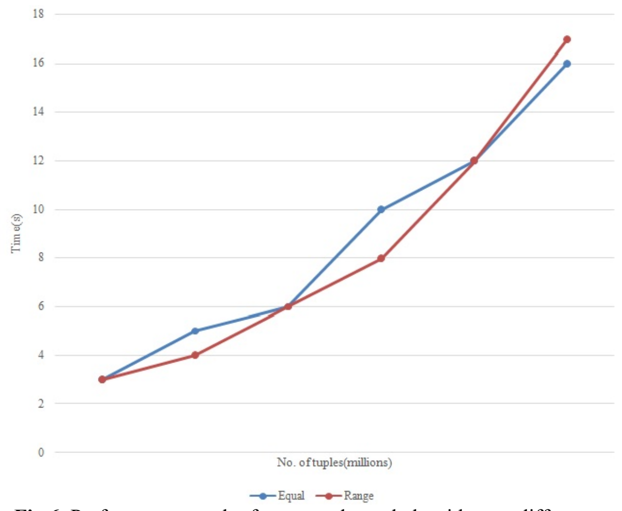

Fig 6. Performance graph of range and equal algorithm on different no of tuples

6. Conclusion

With value-based property of MIN / MAX Aggregation functions, we reduced the computation time of MIN /MAX Iceberg queries up to 98% with our Equal and Range algorithms. These work on bits slices of attributes values. Our algorithms are independent of the number of attributes and number of distinct groups in datasets. There is no intermediate computation of aggregate values which improved computation of the aggregate value of MIN and MAX algorithm.

The range type queries are independent of the threshold value and a number of bits used to aggregate attribute values. Our algorithms are good with data of any size. For large data, first, it needs to be partitioned, later, the query with our algorithms apparently on part of data needs to be executed and finally, result sets are to be combined.In the future, we will work on reducing the computation time of large vector of bits for performing Bitwise AND, OR and NOT operations.

References

[1] S.Agarwal, R.Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. ”On the computation of multidimen- sional aggregates.” In VLDB, vol. 96, pp. 506-521. 1996.

[2] R.Agrawal, T.Imieliski, and A.Swami. ”Mining association rules between

[3] sets of items in large databases.” In Acm sigmod record, vol. 22, no. 2, pp. 207-216. ACM, 1993.G.Antoshenkov. ”Byte-aligned bitmap compression.” In Data Compres- sion Conference, 1995. DCC’95. Proceedings, p. 476. IEEE, 1995.

[4] J.Bae and S.Lee. ”Partitioning algorithms for the computation of average iceberg queries.” Proc. Second Intl Conf. Data Warehousing and Knowl- edge Discovery (DaWaK), pp. 276-286, 2000.

[5] M. Beeler, R.W. Gosper, and R. Schroeppel, HAKMEM, technical report, Massachusetts Inst. of Technology, Cambridge, 1972.

[6] K.S. Beyer and R. Ramakrishnan, Bottom-Up Computation of Sparse and Iceberg CUBEs, Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 359-370, 1999.

[7] C.Y. Chan and Y.E. Ioannidis, Bitmap Index Design and Evaluation, Proc. ACM SIGMOD Intl Conf. Management of Data, 1998.

[8] F. Deliege and T.B. Pedersen, Position List Word Aligned Hybrid: Optimizing Space and Performance for Compressed Bitmaps, Proc. Intl Conf. Extending Database Technology (EDBT), pp. 228-239, 2010.

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

[10] 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.

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

[12] J. Han, J. Pei, G. Dong, and K. Wang, Efficient Computation of Iceberg Cubes with Complex Measures, Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 1-12, 2001.

[13] M. Jrgens, Tree Based Indexes versus Bitmap Indexes: A Performance Study, Proc. Intl Workshop Design and Management of Data Warehouses (DMDW), 1999.

[14] D.E. Knuth, The Art of Computer Programming, second ed Addison- Wesley Professional, Jan. 1973.

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

[16] K.P. Leela, P.M. Tolani, and J.R. Haritsa, On Incorporating Iceberg Queries in Query Processors, Proc. Intl Conf. Data-base Systems for Advances Applications (DASFAA), pp. 431-442, 2004.

[17] P.E. ONeil, Model 204 Architecture and Performance, Proc. Intl Work- shop High Performance Transaction Systems (HPTS), pp. 40-59,1987.

[18] P.E. ONeil and G. Graefe, Multi-Table Joins through Bit-mapped Join Indices, SIGMOD Record, vol. 24, no. 3, pp. 8-11, 1995.

[19] P.E. ONeil and D. Quass, Improved Query Performance with Variant Indexes, Proc. ACM SIGMOD Intl Conf. Management of Data, pp. 38- 49, 1997.

[20] 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, vol. 3, pp. 1-23, 2009.

[21] M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E.J. ONeil, P.E. ONeil, A. Rasin, N. Tran, and S.B. Zdonik, C-Store: A Column-Oriented DBMS, Proc. Intl Conf. Very Large Data Bases (VLDB), pp. 553-564, 2005.

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

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

[24] K. Wu, E.J. Otoo, and A. Shoshani, Optimizing Bitmap Indices with Efficient Compression, ACM Trans. Database Systems,vol. 31, no. 1, pp. 1-38, 2006.

[25] W.P. Yan and P.-A. Larson, Data Reduction through Early Grouping, Proc. Conf. Centre for Advanced Studies on Collaborative Research (CASCON), p. 74, 1994

[26] B.He,H-I.Hsia,Z.Liu,Y.Huang and Y.Chen Efficent computing Iceberg queries using compresed bitmap index IEEE TRANSACTION ON KNOWLEDGE AND DATA ENGINEERING,2012‘

[27] Y.Cui, W.Perrizo ”Aggregate Function Computation and Iceberg Query- ing in Vertical Database”,Computers and Their Applications,2006