Domain Partitioning: Approach to Computing Average Iceberg Queries

Pallam Ravi, D. Haritha

Abstract—Data analytics and data mining systems work on data which stored in files, the files are not store relationships among the data, from such kind of data we compute aggregate values over the set of required attributes for find insights of data, find attributes values which aggregation values greater than threshold such kind of queries called iceberg queries. Computing iceberg queries with average aggregate function is default, because limited memory available. Existing method suffers with re-computation of candidate. We proposed a New Domain partitioning approach, it avoid re-computation of candidate in during next scan of data set , it use bit vector and bitmap numbers for Domain Partitioning the data , our experiment reveals that our approach give high performance than exit methods.

Keywords—Domain Partitioning, anti-monotone, iceberg queries, bit vector and bitmap numbers

I. INTRODUCTION

Some data analytics need Aggregation values on set of attributes, which reveals more information about data ,i.e widely used in decision making systems. Data analytics handle large amount of data (many unique attributes values) , larger than available counter buckets in main memory. For this type of data we need to compute aggregate values over set of attributes to find aggregate values above user provided cut-off values these type of queries are called iceberg queries, the resultant set of iceberg queries reveal more information.

The iceberg queries returns small set of records (10% of unique attributes values) called as tip of iceberg queries. Due to this it requires huge computation over data, iceberg queries are used in data where housing, big data analytics ,market bucket analysis and clustering in data mining, it can be used in such as memory systems like embedded systems. iceberg queries have four characteristics ,it run on 1) large data 2) its domain size is greater than available counter buckets 3) computes aggregation values over set of targeted attributes 4) produces results which aggregate values satisfy user contains.

The iceberg query has AVG aggregation functions called as average iceberg query. The general form of average iceberg query

\[
\text{SELECT } T_1, T_2 \ldots T_n, \text{AVG(rest) FROM } R \\
\text{GROUP BY } T_1, T_2 \ldots T_n \\
\text{HAVING AVG(rest) > Threshold}
\]

Where R is data set which contain \(T_1, T_2 \ldots T_n\) rest attributes , Threshold is user provided value

For Answering average queries need to sort the data and compute aggregate values one by one ,the sorting perform many swapping and scans of data .the other method is hashing method maintain one counter bucket for each unique hash value ,the counter bucket needed are larger than available memory , so with existing buckets only computes the aggregation values, but its tasks many scan of data, if domain size of data is \(D\) (The no of unique attribute values), and \(C\) is no of bucket will maintain at a time, so we need \(D/C\) scans required. To reduce scans of data [1] we use partition methods namely BOP and POP.

The partition method in [3] suffers with re-computation of Target, and non targets, to avoid re-computation we use bit vector to track the computation. In this paper We proposed a method called Avoid Re-computation in POP (RPOP) it avoid re-computation of candidate by efficient use of available memory with use of bit vector and bitmap numbers.

The remaining paper is organized as section2 we discussed about related works, section 3 we explained about re-computation accrued in exiting method ,our proposed method discussed in section 4, section 5 discussed about data sets and results in section 6

II. RELATED WORK

The general methods sorting and hashing used to compute average queries, the first work[1] use coarse count and sampling methods to iceberg queries, it works only for anti
monotone iceberg queries .if we use for average iceberg queries(ICBQ) it give false negative like fig.2. 


ICBQ with non ant monotone [1] partitioning method used, in[10] proposed bitmap number to sort the targeted attributes.

The algorithm used for iceberg cubs[5][14] and database queries[4] are not used for ICBQ .because have its own goals, iceberg cubs algorithm optimize use of memory where as ICBQ algorithms are minimize computational time.

<table>
<thead>
<tr>
<th>T₃</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>B₁</td>
<td>5</td>
</tr>
<tr>
<td>B₂</td>
<td>1</td>
</tr>
<tr>
<td>B₃</td>
<td>4</td>
</tr>
<tr>
<td>B₁</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hash_Value</th>
<th>sum</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig .2. False Negative example

B₁ sum is 8, count is 2 ,so it above cut-off is 3.5, but it committed in candidate selection because of it is combined with B₃,so its sum becomes 9 ,count 3 ,so average value becomes below cut-off in bucket 1,this false negative was mention in[3][1] too

The improved algorithm mentioned in [3] is POP, it suffers with re-computation of targets and non targets, shows in fig 3

For first candidate selection scan B₁ sum is 12, count is 3 so it average was above cut-off ,for B₂ sum is 8 count is 2, B₁ and B₂ average is above threshold ,so selected for candidate computation

For candidate selection scan B₁,B₂ which is already compute exactly average of entire data but it leads to re-computation target(B2).To avoid this re-computation we domain partitioning the target value it will discussed in section 3.

<table>
<thead>
<tr>
<th>T₃</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>B₁</td>
<td>5</td>
</tr>
<tr>
<td>B₂</td>
<td>1</td>
</tr>
<tr>
<td>B₃</td>
<td>4</td>
</tr>
<tr>
<td>B₁</td>
<td>3</td>
</tr>
</tbody>
</table>

(b) Counter buckets at First candidate generation

<table>
<thead>
<tr>
<th>bucket</th>
<th>sum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>B₁</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>B₂</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

(c) Counter buckets at second candidate generation

(a) Data Set D

![Fig.3. Re-computation of candidate Example](image)

We assign Bitmap Numbers [10] to records, use this number we identify the records which have same Domain value. Bitmap Numbers

BitMapNumber is assigns a number to record the function of target attribute values

BitMapNumber = f (target attribute values) (1)

Each target value record is assigned with an integer value from 0 to n values, n is cardinality of that attribute and same no of bits allocation for all attributes ,explain with example.1.

**Example 1:** A and B are two attributes with 3 and 4 cardinality, A₁,B₂ target attribute records bitmap number i.e. \(1*2^3+2*2^2=18\)

### III. DOMAin PARTITIONING

Computing AVG ICBQ with use of POP[3] Algorithm, it computes complete AVG values of candidate target which satisfy the threshold value in a single partition of dataset

Example 2: Apply POP algorithm on data set in fig3(a) with Bucket count 2 and threshold 3.5. For one scan of data set it computes total average values of B₁ and B₂,and for second scan to compute total average values of B₃ and B₂, in these two scans B₂ is computed twice. We Proposed Domain partition approach, it avoids re- computation of candidate target set .

**Domain Partition:**

Let \( D = \{D₁, D₂, ... Dₙ\} \)

\( Dᵢ \cap Dⱼ = \emptyset \) for all \( i ≠ j \) and \( t ∈ Dᵢ \)

D satisfies two conditions:

1) \( t > \text{threshold in } pᵢ \)
2) \( \text{find complete average of } Dᵢ \)

*single record partition*

First condition, The divide Domain value are set target values which threshold above the threshold up to that portioning that use partition theorem in [3] ,for with record tracking second condition is accommodated, it use Bitset, Bitset is sequence of 1’s and 0’s bits ,bit position represent record number and a bit value 1 indicates whether that record is used for total computing Average ICBQ ,where 0 indicate that record not used. In example 1 first its Bitset values are 00000000 ,it becomes after computing the Domain value of B₁ and B₂ it result as 11111010.In second scan of dataset it computes Average values of domain B₃ only and avoid re-computation of B₃ in second scan of data set.

Let unsorted Dataset D ,disjoint Record order based partition \( P₁,P₂,.. Pₙ\) of D ,means \( Pᵢ ∩ Pⱼ = \emptyset \) for all \( i ≠ j \), Threshold value T and t target records

**Table:**

<table>
<thead>
<tr>
<th>bucket</th>
<th>sum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>B₃</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>B₂</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>
3.1 Record Tracking Algorithm (RTA)

It uses Bit vectors, which are sequences of 1,0 (true or false), where it uses for tracking whether the record can globally compute average value or not. It computed that record line number is used as index. In bit vector set 1 at this index position, when we allocate new buckets for records, if that record line number indexed bit vector is 0, if bit vector is 1, new bucket is not allocated, then this is modification of POP algorithm, remaining same as POP, we call modified as record tracking algorithm (RTA).

Algorithm: Record Tracking Algorithm Inputs: Relation

1. Begin:
   2. bucket B;
   3. Second_scan=0;
   4. BitSet BS;
   5. Line_no=0;
   6. For each record r in R {
      7. If r in bucket B {
         8. B(r).count++;
         9. B(r).value+=r.value;
         10. Line_no++;
      } Else {
         11. If Bucket_count<size B] && BS(Line_no)=0; {
            12. Insert r in B with B(r).count=1; B(r).value=r.value;
            13. Bucket_count++;
            14. Line_no++;
         } else {
            15. for each B(t) {
               16. if B(t).value/ B(t).count < T {
                  17. B(t).remove
                  18. } If Bucket_count<|B| {
                     19. Second_scan=1;
                     20. all t reset B(t).count =0
                  } }
               21. } If Second_scan=1 {
                  22. For each record sr in R {
                     23. B(sr).count++;
                     24. B(sr).value+=r.value;
                     25. sLine_no++;
                     26. BS(sLine_no)=1;
                  } Print all t if B(t).value/ B(t).count >T
                  27. Second_scan=0;
               } 
   8. }
9. 

**Theorem 1:** there existence of some t, it becomes a candidates in one or more disjoint Record order based partitions

\[ P = \{ t : t = AVG (t, P_i) \geq T, i = 1,2,..n \} \text{ then } |P| \geq 1 \ni \exists t (1) \]

Let \( P_t \) and \( TP \) are

\[ P_i = \{ t : t = AVG (t, P_i) \geq T \} \quad (2) \]

\[ TP = \{ t : t = AVG (t, P_i) \geq T, i = 1,2,3.. n \} \quad (3) \]

\[ Re - computation = \sum_{i=1}^{n} n(P_i) - |TP| \quad (4) \]

denotes, representing with (2), According (4) re-computations happens, so use domain partition approach we avoid re-computations it equals to shows in (4).

We write an algorithm based on Domain partitioning call as Record Tracking Algorithm (RTA) discussed in section 3.1

**Example 1:** take relation R in fig 4, the max counter bucket is 2, cut-off=3. Initially bit vector is set as 00000000 number bits equals number of records so 8 0’s, first scan bucket contain B1, B2 up to 2nd record the counter bucket becomes full, start second scans the 1,2,3,4,5,7,8 records are used, so Bit vector becomes 11110111, produce the results as B1, then counter bucket removed continue new bucket allocation start from 3rd record, only 6 the record are not used so allocated bucket for it that B3, compute for cut-off then product for results, so it avoid re-computation.

The bit vector is operation take less amount of memory, it is to take the bits equal to number of records in data, it very easy to track the computation and easy to check indexed position values.

To reduced memory required for bit vectors by using the bitmap numbers [10] to track the weather the record is used or not, can generate bit map number for each record, then it used as index as record line number.

**Example 2:** Assign bit map numbers to records as in [5], B1, B2 and B3 bit map number are 1, 2 and 3 respectively.

So we require Bit vector with size of three, after first bucket is full, for second scan only 110 are set in Bit vector,

So with bit map number we can reduced the memory required for track computation by using Bit vector

\[ T=20 \times (10100) = V_1, V_2, V_3, V_4, V_5 \]

IV. EXPERIMENT RESULTS

In windows 7 32bit with 2GB RAM system is used this for experiment. Record Tracking Algorithm was implemented in java programming language and used wake library. Two synthesized data set are generated for this experiment with 100,000,000 records, about 2.1 GB. The distribution of target attributes values, domain size and max average value as follow.

Dataset 1: Target attributes in normal distribution and domain size 220000, min and max values are 0 and 999000 respectively.

Dataset 2: Target attributes are in uniform distribution and domain size 1,000,000, min and max values are -19000 and 21000 respectively.

Domain Ratio: It is the ratio of domain size and no of counter buckets, Domain ratio >=1.0 indicates sufficient counter bucket available for all possible target values (domain size), if it is <1.0 indicates insufficient counter buckets.

To evaluate Performance of RTA, we did an experiment with respective of execution time, no of candidate computation and no of data scans need, In this experiment keep threshold value constant with changing of domain ratio, threshold value of 700,000 for data set 1 and 14000 for data set 2, the experiments reveals the RTA gives better performance in Fig 4 (a)&(b) domain ratio from 0.5 to 0.1 RTA execution time was very less compared to POP, because to it computes less number of candidates shows in Fig 5 (a)&(b) and it performs less no of scans in Fig 6 (a)&(b), this will happen avoid re-computation of candidate by tracking records.
In Fig 6 (a)&(b), RTA has less on of scans with Record Tracking records ,also the partition size will increase from scan to scan. No of reduced candidate Re-computation is indicating distance between curves in Fig 5 (a)&(b) These result are without use of bitmap number, if it used only reduced memory requirement of Bit vector

V. CONCLUSION

According our theory candidate set selection happens in more than one partition, it leads to re-computation, Domain partition approach record tracking algorithm is avoid re-computation of candidates it track the records which are used for computing in oral computation of average aggregate value that record is not used for further candidate selection in further partition, we use Bitset for tracking records, and we reduced memory required for Bitset by applying bitmap number.

Our experiments reveals RTA is giving better performance than existing algorithm. At domain ratio 0.1 to 0.5 it compute less no of candidates and with use lesser scans lesser POP, the maximum no of data scans need \( \frac{D}{C} \) only

REFERENCES


Pallam Ravi, Research scholar in KL university, Assistant professor in Dept. of Computer Science & Engineering Anurag Group of institutions, he reived B.Tech in Computer Science & Information Technology from JNTU Hyderabad, M.Tech in Software Engineering from JNTU Hyderabad University, he present research on data mining

Dr. D. Haritha, Professor in Dept. of Computer Science & Engineering is a Committed Academician and Active Researcher having 8 years of teaching experience at KL University. She Received B.Tech in Computer Science & Engineering from JNTU Hyderabad, M.Techin Computer Science & Engineering from Acharya Nagarjuna University, Ph.D from Acharya Nagarjuna University in the area of Software Reliability. She has 14 research publications in various referred International Journals. She is a life member of CSI. Currently 4 research scholars are pursuing Ph.D. degree under her guidance. Her areas of interest are Software Reliability, Data Analytics and Text Mining.