Two State Counter Bucket: Computation of Average Iceberg Queries

Pallam Ravi, D. Haritha,

Abstract: Abstract: in limited memory environment computing Aggregate values needs many scans of data, to avoid these scans use Apriori property for computing anti-monotone iceberg queries but with efficient use bucket counter reduce scans for computing non anti-monotone iceberg queries , till now all algorithms use single state Bucket counters, which suffers massive counter checking for candidates , we propose two-state Bucket counter which reduces counter checking, we conduct an experiment on POP algorithm

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

I. INTRODUCTION

In data analytics and data mining operation needs aggregate values, these are handled with a larger unique value called domain size, for computing aggregate value for each unique value need a large amount of memory need, for getting information from the aggregate value which satisfies threshold value, finding such kind of unique value called iceberg queries the small set of domain values are produced as resultant, it called the tip of iceberg queries(ICQ), equal to 10% of domain values ICQ handles a large amount of data, the domain size is greater than the available counter, aggregate values computation over attributes, small set records produces a resultant set, and apply user threshold on aggregate values, it needs huge computation and many data scans needed. ICQ are used in many application such as data mining, an embedded system which has limited memory, information retrieval The aggregate function like COUNT, MIN, SUM STDIV, MAX, and AVERAGE are used in iceberg queries(ICQ), these classified as anti-monotone and non-anti-monotone functions anti monotone function are MAX, MIN, SUM and COUNT, AVERAGE and STDIV are now anti monotone aggregation functions .use of anti-monotone property it reduces computation in candidate generation but not reduce in non anti monotone aggregation function, it needs many data scans to need. Challenge is to reduce data scans for computing non-anti-monotone ICQ average iceberg query is computing AVG aggregation. The general form of average iceberg query(ICQ)

SELECT A1, A2, ..., An AVG(rest) FROM D
Having AVG(rest) > Thres

Where D is a data set which contains A1, A2, ..., An rest attributes, Thres is a threshold value

General method to Answering Average iceberg queries is sorted the data with respective target attributes values, sorting takes many scans and swapping its in efficient, the other method is allocating one counter buckers for each unique target value, in this method no of counter bucket need equals, but ICQ computed with limited memory so memory is not available as required, the first work on average ICQ in [1] it use partition methods namely POP and BOP. Partition methods POP work as fill bucket with data and remove which those are not satisfied threshold value, it reparate until no more remove counter from the bucket, it suffers from many scans and checks of counter buckets it explains in section 2, we proposed two-state bucket to eliminate scans of counter buckets Related work on ICQ in section 2, in section 3 explain two-state bucket, in section 4 explain experiment and dataset used.

II. RELATED WORK

The first work average ICQ based on a partition-based algorithm called POP and BOP [1], ICQ proposed in [1] coarse count and sampling methods it gives false negative For anti monotone ICBQ are efficiently compute using Bitmap Indexes[2][n] in [6], in [6] use dynamic pruning to avoid mass empty BIT-WISE AND operations, the different author improves the performance is in [8] cache-based, in [9] checkpoint, in[11]look head pointer, in [13] work with distributed system and in [12]work on vertical datasets, 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 the use of memory whereas ICBQ algorithms are reduced computational time. Partition methods POP work as fill bucket with data and remove which those are not satisfied threshold value, it reparate until no more remove counter from the bucket, it suffers from many scans and checks of counter buckets it explains With example 1, we proposed a two-state bucket to eliminate scans of counter buckets with one state bucket it suffers massive counter checking for candidate selection and average computation it will explain with an example 1 Example 1: let R is a relation with target attributes A, B, and C, threshold values is 10 and Max counter in the bucket is 3

Relation R
Two State Counter Bucket: Computation of Average Iceberg Queries

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A1</td>
<td>B1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>A1</td>
<td>B2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>A1</td>
<td>B1</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>A2</td>
<td>B1</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>A2</td>
<td>B2</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>A1</td>
<td>B2</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>A1</td>
<td>B1</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>A2</td>
<td>B1</td>
<td>5</td>
</tr>
</tbody>
</table>

Counter allocate for each new unique record, bucket becomes full.

Table 1.a. Counter Buckets values

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1B1</td>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>A1B2</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>A2B1</td>
<td>13</td>
<td>1</td>
</tr>
</tbody>
</table>

The bucket is fill up third record scan then remove counter which average value is below a threshold value, so A1 B2 is removed, allocation counter for A2B2 then the bucket is full.

Table 1.b. Counter Buckets values

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1B1</td>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>A2B1</td>
<td>13</td>
<td>1</td>
</tr>
</tbody>
</table>

For remove counter need to read all three counter and calculate its average value i.e., to void all counter checking and calculating average value we proposed a two-stage counter bucket.

Above problem with solve with new way checking using theorem 1 average threshold that explains in example 2.

Theorem 1: Checking average values of $a_1, a_2, ..., a$ with $T$

$$F = \sum_{i=0}^{n}(a_i - T)$$

$F > 0$ average values of $a_1, a_2, ..., a$ are above $T$

$F < 0$ average values of $a_1, a_2, ..., a$ are below $T$

Example 2: With the use of example 1 problem, checking the average value of A1B1 its allocate bucket with A1B1 2 (12-10) for record 1 for record 3 update it as A1B1 2+1 (11-10), it becomes 3, finally check it is above zero are not, so avoid maintain a counter.

With the use of theorem 1 bucket becomes full, it eliminates A2B1 and A2B2 without check the A1B1.

Table 2. two-state Counter Buckets

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1B1</td>
<td>5</td>
</tr>
<tr>
<td>A2B1</td>
<td>-2</td>
</tr>
<tr>
<td>A2B2</td>
<td>-1</td>
</tr>
</tbody>
</table>

III. TWO STATE COUNTER BUCKETS

In POP algorithm the buckets operations are New counter, Update and remove that explains in fig 1. In this remove operation did with respective all counter in bucket to avoid this two-state bucket counter proposed, in state one has new counter, remove update and change state, the new operation change state is added and differ in remove operation, the remove operation removes all counter which average value below threshold and update the counter which value greater than threshold with value/count - threshold.

In state 2 new counter, remove and update operation are differ with state 1 and one state bucket operation, in new counter bucket allocate with single values only if value < threshold, whereas in state 1 without checking value with threshold, it creates with two values i.e < value, 1>, Update operation adds value with value-threshold and remove operation done with respective single counter if that counter has less than zero then remove it.

State 1:

New counter:

if no Counter in Bucketer for a Record then

Allocate New Counter with <value, 1>

Remove:

if |counter buckets| <= Max_counter then

for all counter update its value with its value/count - threshold

for all counter its value < Threshold

Remove counter from bucket

Update: if a record have counter in bucket

Add value of counter with record value and increment count

Change state: After remove, change state to state 2

State 2:

New counter:

if no Counter in Bucketer for a Record and its value > Threshold then Allocate New Counter with <value - threshold>

Remove: if counter value < 0 then

Remove counter from bucket

Update: if a record have counter in bucket

Add value of counter with record value-threshold

Change state: After bucket is full, change state to state 1

In state 1 &2 in bucket operation flow represent in fig 5 & 6 respectively, in state 1 first its check for new counter allocation, if already have a counter for it update it for a record not able to create new counter perform remove operation then change state to 2.
In state 2, first check for new bucket allocation if it have counter for it update it and perform remove operation, change state operation done when no counter for a record

Algorithm: state 1 Bucket operation

\[
\text{If}(!\text{New counter})
\]

\[
\text{If}(!\text{Update})
\]

\[
\text{If}(!\text{Remove})
\]

\[
\text{Change state 2}
\]

\[
\
\]

Algorithm: state 2 Bucket operation

\[
\text{If}(!\text{New counter})
\]

\[
\text{If}(\text{Update})
\]

\[
\text{If}(\text{Remove})
\]

\[
\
\]

\[
\text{else}
\]

\[
\text{Change state 1}
\]

One State Counter Buckets:

New counter: if no Counter in Buckets for a Record then

<table>
<thead>
<tr>
<th>Domain</th>
<th>0.9</th>
<th>0.8</th>
<th>0.7</th>
<th>0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td>118</td>
<td>2571</td>
<td>1900</td>
<td>2000</td>
</tr>
<tr>
<td>Normal</td>
<td>3</td>
<td>42</td>
<td>325</td>
<td>755</td>
</tr>
</tbody>
</table>

Table 3: No of counter Buckets Scans

IV. EXPERIMENT

We conduct experiment on 4GB RAM and windows 7 operation system, we use two synthesized data set with different distribution namely normal and uniform distribution of target attributes 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 statesPOP, we did an experiment with respective of execution time. In this experiment keep threshold value constant with changing of domain ratio, threshold value of 700,000 for data set 1 and 14000 for dataset 2. The experiments reveals the statesPOP gives better performance shows in fig 1 &2. Due to reduce scans of counter buckets shown in table 1.

Allocate New Counter with <value, 1>

Remove:

\[
\text{if}(|\text{counter buckets}| >=\text{Max \_ counters} \text{ then})
\]

\[
\text{for all counter its value < Threshold}
\]

Remove counter from bucket

Update: if a record have counter in bucket

Add value of counter with record value and increment count

In POP algorithm have two scans namely first scan and Second Scan, our two state bucket used in first scan only, in second scan differs the record update, it did as Update operation in state 2, in first scan bucket in state 1. New Counter and Update operation are not differ only differ in remove operations.

Algorithm: Two-State Bucket POP algorithm (statesPOP)

First Scan: start with Bucket in state 1

In State 2: change state start the second scan

Second scan: perform State 2: Update

Print counters in Bucket which > 0

V. CONCLUSION

Our theorem for efficient checking of average value with threshold used in our two state counter buckets for eliminate rescans of entire counter buckets for remove counter which are below threshold value due this reduce computation time for average iceberg queries.
REFERENCES


AUTHORS PROFILE

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

Professor in Dept. of Computer Science & Engineering is a Commited 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.Tech in 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