Collision Free Intelligent Bloom Join Filters

Dr. Sunita M. Mahajan, Ms. Vaishali P. Jadhav
Principle, Mumbai Education Trust, Computer Science Dept, Bandra Reclamation, Mumbai, India
Research Scholar, NMIMS University, Vile-Parle, Mumbai

Abstract— In operation research, there is no single method available for solving all optimization problems. Hence a number of techniques have been developed for solving different types of optimization problems. Optimization is the act of obtaining the best result under given circumstances. The ultimate goal of optimization is either to minimize the efforts required or to maximize the desired benefit [5]. Query Optimization is one of the optimization problems in database management system. It is a process of determining the most efficient way to execute a given query by considering the possible query plans. The approach suggested in the paper is mainly focused on join operation of the query. Previous work done was based on semi-join approach for query optimization but a semi-join needs more local processing such as projection and higher data transmission. To improve the previous approach, the filter based approach is utilized. The evaluation of filter is done by considering the collisions occurred, using perfect hash function and using sets of filters. Proposed filters focuses on importance of optimization and Intelligent Bloom Join filter approach for data reduction in query optimization.

Keywords—Optimization; Query Optimization; Bloom Join; Bloom Filter.

I. INTRODUCTION

Different approaches are suggested for query optimization. These approaches mainly includes joins, semi-joins etc. Join algorithm has high complexity and also it lead to a high data transmission cost. Semi-join approaches are better than joins but they may entail more local processing cost and more calculative cost. Bloom join is the next approach suggested for minimization of query cost. The use of filters in bloom join greatly improves the performance. A bloom filter was first developed in 1970 by Burton H. Bloom. It is an array of bits which functions as a very compact representation of the values of a join attribute. Bloom join may give the same result as a semi-join but at a much lower cost. The Bloom filter algorithm has lower local processing cost and data transmission cost but it has a problem of collision i.e. two different attributes values may be hashed in to same bit address. As the collision increases, the data reduction decreases. So this collision problem is minimized by use of set of Bloom filters. The data reduction is also achieved by applying these filters at the same time to same relation [1-4].

Bloom filter is a data structure for representing an object in memory. It provides a probabilistic approach to represent a set, in order to evaluate membership of an element in a set. It is a simple, space efficient, randomized data structure. To boost the performance of query execution, the filter based approach is utilized. Proposed algorithm uses the sets of filters to reduce the number of rows from a table. Filter based approach has the problem of collisions. This problem is minimized by using a set of bloom filters and by using a set of filters. Proposed filters are called as Intelligent Bloom Join Filters as they keep on changing as there is a change in a reduced table or relation and at the same time these filters are applied to the original relation that constructs the filter first time for reduction. Because of intelligence of bloom filters and set of filters working at the same time on same relation, the relation is getting fully processed and reduced. The main objective of this approach is to minimize the number of rows in a table or relation so that local processing cost, transmission cost in distributed environment and collision is reduced. With this approach query execution will be fast and with less response time. The suggested approach removes the non-contributive rows from a table and reduce the effect of collisions occurred in filter approach. The percentile reduction of rows is provided in the paper.

For reducing the network cost, Bloom filters have found wide use in distributed databases. Also peer-to-peer applications use bloom filters to represent peer contents, to enable query routing in unstructured P2P networks. They are also used for optimizing collaboration protocols, such as collaborative caching, content reconciliation. Bloom filters are also used to represent confidential data, to enable join execution without revealing information [6-16] [18].

Various types of bloom filters are available. The counter bloom filters normally used with deletion operation of bloom filters. Distance-sensitive bloom filters uses locally sensitive hash functions. Space-code and spectral bloom filters are approximate representation of multi set. The compressed bloom filters improves the performance in terms of bandwidth saving when bloom filters are passed on as messages. A bloom filter with two hash functions applies hash functions to reduce the data [6-16] [17].

The rest of the paper is organized as follows: section II gives the importance of optimization in
operation research [5]. Section III gives Bloom Join Filter approach. This section gives the flowchart and pseudo-code of collision free intelligent bloom join filter algorithm. Section IV gives experiments and results and finally section V concludes the work and outlines some future work.

II. OPTIMIZATION

Minimization of efforts or maximization of benefit in optimization can be expressed with the help of decision variables. Optimization can be defined as the process of finding the conditions that give the maximum or minimum value of a function. The following Fig.1 shows that if a point \( x* \) corresponds to the minimum value of function \( f(x) \), the same point also corresponds to the maximum value of the negative function, \(-f(x)\). Thus without loss of generality, optimization can be taken to mean minimization since the maximum of a function can be found by seeking the minimum of the negative of the same function [5].

\[
\text{Fig.1 Minimum of } f(x) \text{ is same as Maximum of } -f(x)
\]

Irrespective of optimization method used, three things always need to be specified: (i) representation of the solution which will determine the search space and its size. The size of search space depends on its representation. Representation varies with different optimization techniques used. (ii) Objective is a mathematical statement of a task to be achieved. It is not a function, but an expression. (iii) Evaluation function is mainly used to compare the quality of different solutions.

In query optimization, different optimization algorithms have different representations. For example, in semi-join algorithm, optimization need size of relations, number of relations, size of joining attributes, selectivity etc. and in bloom join, optimization need relations, adjacency list, inverted list, bit-arrays, filters etc. But objective is same for both the algorithm i.e., minimization of cost, response time and minimization of data transfer between different sites etc. Evaluation of different algorithms can be done by comparing the results obtained by them and finding which algorithm gives the minimum cost or response time.

III. BLOOM JOIN FILTER

Bloom filters are compact data structures for probabilistic representation of a set in order to support membership queries. The main design tradeoffs are the number of hash functions used, the size of the filter and the error (collision) rate.

Consider a set \( S = \{s_1, s_2, ..., s_n\} \) of \( n \) elements. Bloom filters describe membership information of \( S \) using a bit vector \( V \) of length \( m \) and with \( k \) hash functions, \( h_1, h_2, ..., h_k \).

The following function builds an \( m \) bits Bloom filter, corresponding to a set \( S \) and using \( h_1, h_2, ..., h_k \) hash functions [1-6][20]:

\[
\begin{align*}
\text{Function } & \text{BloomFilter(set } S, \text{hash_functions, integer } m) \text{returns Bloomfilter} \\
& \text{Bloomfilter} = \text{set } m \text{ bits Bloomfilter} \\
& \text{foreach } s \text{ in } S; \\
& \quad \text{foreach hash function } h_i: \\
& \quad \quad \text{Bloomfilter}[h_i(s)] = \text{True} \\
& \quad \text{end foreach} \\
& \text{end foreach} \\
& \text{return Bloomfilter}
\end{align*}
\]

Fig. 2 Construction of a Bloom Filter

Therefore, if \( s_i \) is member of a set \( S \), in the resulting Bloom filter \( V \) all bits obtained corresponding to the hashed values of \( s_i \) are set to True or 1. Testing for membership of an element \( elm \) given in fig.3 is equivalent to testing that all corresponding bits of \( V \) are set [1-6] [20]:

\[
\begin{align*}
\text{Function } & \text{ElementTest(elm, filter, hash_functions)} \text{returns yes/no} \\
& \text{foreach hash function } h_i: \\
& \quad \text{if filter}[h_i(elm)] == 1 \text{ return yes} \\
& \quad \text{else return no} \\
& \text{end foreach}
\end{align*}
\]

Fig. 3 Membership Testing of an element in a Set

A. Collision –free Bloom Filter Join Flowchart

The basic flowchart of bloom filter design is given below. The first flowchart Fig.4 shows the construction of data structures used in Bloom Filters and checking whether all relations are fully processed or not. Second flowchart Fig.5 shows the filter construction and reduction of original filter with newly changed filter [2-4][7].
B. Collision-free Bloom Filter Join Algorithm

Pseudo-code

```plaintext
Build basic data structures such as adjacency matrix, adjacency list etc.
Build AdjacencyListCount which includes the count of each relation’s in-degree.
Sort AdjacencyListCount and find the relation with lowest in-degree
Let RiCount is Lowest and so start with Relation Ri.

if(AdjacencyListCount != empty) // Repeat until all relations are fully processed
{
  //Designing filter for first column of Ri i.e. RiList[0]
  // Filter Check of that column in filter list
  i(FilterList != empty)
  {
    foreach Filter in FilterList
    {
      foreach column in Ri
      {
        if( Filter exists in FilterList for that column )
        {
          Assign Column to CommonCol between Ri and FilterList
          Apply existing Filter to reduce the relation
        }
        else
        {
          Design a new filter for that column
        }
      }
    }
  }
  Applying existing filter to reduce the relation:
  if(CommonCol != empty)
  {
    if(CommonCol = “Join Attribute1”) 
    {
      Assign BloomFilter1 to Join attribute1
      foreach element in BloomFilter1
      {
        select * from Ri where Join Attribute1=element }
    }
  }
}

Designing a new Filter for column:
if(CommonCol does not exists)
{
  Read RiList[0] // Reading First column of Ri
  // Design Filter for RiList[0] // Use perfect hash function
  Set the BloomFilter with the exact contents of that column
  Add RiList[0] to FilterList
}
```

Fig. 4 Flowchart of Intelligent Bloom Join Filter Algorithm

**Fig.5 Flowchart of Intelligent Bloom Join Filter Algorithm**
// decreasing the in-degree of relation for which filter is designed
if(RiList[0] exists in any other relation)
{
Remove that column from respective relation adjacency List
Decrease the in-degree of respective relation once the filter for
RiList[0] is designed.
}

if(In degree of Ri = 0)
{
Display “Relation Ri is fully processed”
}
// Repeat from first loop until all relations are fully processed

Explanation

Algorithm checks the in-degree of each participating relation and
starts with lowest in-degree relation. The algorithm runs until all
relations are fully reduced or processed. The selected relation
designs the new filter for first column of that relation. Perfect
hash function and Bloom array is used which maps column
values exactly with index values of bloom array. After designing
a filter remove that column from adjacency list and the in-degree
of relations which includes that column is reduced by one.
Designed filter is added to filterlist. If all the joining columns of
any of the relations are fully designed, then that relation is called
fully processed relation. Algorithm designs new filters until all
relations are fully processed.

If the filter exists in the filterlist for the column in the relation,
then apply existing filter to that column. Reduce the relation and
save changes in temporary table. If temporary table is not same
as original relation and if the original filter changes after
applying to another relation, then apply new filter to the original
relation which constructs that filter. Construct the new filters for
columns from reduced relations. Remove original and apply new
filters wherever required. Update the original filterlist. After
applying filters to all joining attributes of a relation, the relation
is reduced fully. Check the filterlist until all relations are fully
processed or reduced.

IV. EXPERIMENTS AND RESULTS

This chapter describes the experimental system and experiments carried out using single set of
filters, two sets of filters. The evaluation of algorithm is done by testing the bloom join filter
algorithm with various queries [2-4][7][17-19].

The bloom join filter algorithm is implemented in C#. We constructed 120 queries based on eight
different databases. The database includes five AdventureWorks databases, NorthWind database,
Pubs database and our own designed query optimization database. The queries and relations vary in the number of relations, sizes of relations,
number of joining attributes, number of columns in each relation etc.

The relations are reduced by Intelligent Bloom Join filters. Experiments find how close an algorithm achieves the data reduction with real
queries. Perfect hash function is used for testing of algorithm with set of filters. Each row shown in
table is average of 20 queries. Query Type specifies number of relations with number of join attributes.
For ex. 5-#2 denotes 5 relations with 2 joining
attributes. Bloom join reduction of each relation is shown in percentages.

<table>
<thead>
<tr>
<th>Query Type</th>
<th>No of Intelligent Bloom Join Filters applied</th>
<th>Database Used</th>
<th>Intelligent Bloom Join Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-#6</td>
<td>6 Query Optimization</td>
<td>R1 6 R2 7 R3 6 R4 6 R5 34</td>
<td></td>
</tr>
<tr>
<td>3-#2</td>
<td>2 AdventureWorksLT2 008R2</td>
<td>R1 8 R2 5 R3 1 R4 0 R5 -</td>
<td></td>
</tr>
<tr>
<td>2-#1</td>
<td>1 AdventureWorksLT</td>
<td>R1 8 R2 0 R3 - R4 - R5 -</td>
<td></td>
</tr>
<tr>
<td>1-#4</td>
<td>4 Query Optimization</td>
<td>R1 8 R2 - R3 - R4 - R5 -</td>
<td></td>
</tr>
<tr>
<td>3-#3</td>
<td>3 NorthWind</td>
<td>R1 0 R2 6 R3 1 R4 - R5 -</td>
<td></td>
</tr>
<tr>
<td>3-#5</td>
<td>5 Query Optimization</td>
<td>R1 5 R2 0 R3 4 R4 0 R5 25</td>
<td></td>
</tr>
</tbody>
</table>

The above results shows that as the number of filters increases, the percentage of reduction also
increases depending upon the query used. With a single filter used, the percentage of data reduction is
very small. When number of filters used increased to 5 or 6, data reduction of all relation increases. In
some cases, there may not be any data reduction with relations.
The Fig. 6 shows the bar chart of data reduction (%) vs. query type. All the relations are not applicable to each query type. So bar chart shows only those relations which are participating in data reduction. For example, in query type 5 - #6, all five relations shows the reduction while in 3 - #5, only three relations which participated in data reduction shows the output. In 3 - #3, even if three relations are participated in reduction process, only one is reduced with 61%, so it is involved in bar chart. The reduction of a relation is purely depends upon the query. It is not necessary that if query involves 3 relations, all 3 should be processed fully.

The experiment shows the effect of different filters on different types of queries. As the number of filters increases, the data reduction rate also increases. The tuples not required for the final answer of the query are eliminated from the relation using multiple filters.

For experimental evaluation, the maximum relations considered are 5 and maximum joining attributes considered are 6. Different types of queries are considered from eight different databases. Reduction of each relation used in a query is given in percentage.

Finally among the different join algorithms suggested earlier, Collision – free Bloom Join Filter algorithm works better than any other algorithm. It requires less number of data structures and number of phases required for data reduction is also less in collision free bloom join algorithm. It mainly saves the overhead of data transmission in case of distributed databases.

Results are based on 120 queries. Each query type result is average of 20 queries. Data reduction is more if numbers of filters involved in relation are more. Percentage of data reduction is depends upon the size of relation and number of filters applied to it for reduction.

V. CONCLUSION

For query optimization problem, to find the optimal sequence of database operations to process the query is a NP hard problem. So here paper suggested the Intelligent Bloom Join filter which minimizes the collision and increases the data reduction rate. Bloom Filter uses the perfect hash function and each joining attribute uses separate Bloom filter array. A set of filters applying at the same time on same relation avoids the non-contributive tuples from a table.

The experiment shows the effect of different filters on different types of queries. As the number of filters increases, the data reduction rate also increases. The tuples not required for the final answer of the query are eliminated from the relation using multiple filters.

For experimental evaluation, the maximum relations considered are 5 and maximum joining attributes considered are 6. Different types of queries are considered from eight different databases. Reduction of each relation used in a query is given in percentage.

Finally among the different join algorithms

REFERENCES


