Indexing a Suitable Approach for Data Warehouse Design

Kale Sarika Prakash  
Department of Computer Science and Engineering  
St. Peters University, Chennai, India  
sarika_k_99@yahoo.com

P.M. Joe Prathap  
Department of Information Technology,  
RMD College of Engineering, Chennai, India  
drjoeprathap@gmail.com

Abstract—Data warehouse is a collection of huge database which is subject oriented, integrated, time variant and non volatile. As it is a set of huge database, fast data access is the major performance parameter of any data warehouse. Generally the information retrieved from Data Warehouse is summarized or aggregated as it is required for some decision making process of organization. To retrieve such a information queries to be fired is of the nature aggregation function followed by having clause. Extracting information efficiently from data warehouse is the challenge in front of researchers. As it is a huge database time required to access information is more compare to normal databases. Due to this fact indexing on any table by using B-tree approach is useful for the databases where the frequent updates are required like On Line Transaction Processing system(OLTP). It is a time consuming approach for data Warehouse and On Line Analytical System(OLAP). Data warehouse is not frequently updated so Bitmap indexing is appropriate choice for the same. We have to create bitmap index on required vector at the start only. Once it is created on fixed database we can use it any time for any query. As per the requirement of query we have to select bitmap and execute query. The bitmap indexing is appropriate choice for Data warehouse only because of its feature like it is non volatile and huge data set.

Keywords—Data warehouse(DW), Iceberg(IB) queries, bitmap index(BI), Online analytical Processing(OLAP), Online Transaction Processing(OLTP), Iceberg query(IBQ)

I. INTRODUCTION

Data warehouse is a collection of huge database collected from various external data sources. It creates base for Decision Support System(DSS) with huge collection of information that can be extracted using Online Analytical Processing(OLAP) application. The way of accessing information from data warehouse is by writing OLAP queries. This huge data is historical so it is not frequently updated [1,2,3,4,5].

Data warehouse is a collection of datasets from various external data sources it is not possible to fit it in two dimensional relational database. It is represented using multidimensional databases. In multidimensional databases each dimension is nothing but one subject oriented table with attributes of relevant metrics[6]. Due to this queries written on DW are complex and includes join operations. This increases response time of query as it is working on huge database. The nature of query is aggregation function, summarization followed by having clause. This type of query is very complex and requires more time. To reduce this query execution time data analysts uses some techniques like data partition, summary of tables and indexes[6]. Data partitioning and summary of table concepts are not suitable for DW because DW is collection of number of dimensional tables. In such a situation selecting data partitioning and summary attribute is very difficult task. It require more time as well as more memory to store partition as well as summary tables. So the suitable choice for DW is indexing technique.

Different indexing techniques has been suggested by different database vendors such as B-tree [4],[6],[7],[9], projection [8], bitmap[6], Join bit map[10]. Researchers [13],[14],[15] suggest Bitmap index is a best choice for a system having data that are not frequently updated by different simultaneous as well as concurrent programs or process. They come to this conclusion only because of the fact that in case of bitmap indexing large amount of information related to row is stored in block of particular index structure. Meaning is that in bit map indexing locking of the values are done in block level. Any updates in database is affect the structure of bitmap created for that vectors. If there is any update in original table then new bitmap has to create. Whereas in case of B-tree all the leaf node blocks are at the same level and it does not requires rebalancing. Whenever original database is updated accordingly complete tree is balanced. Due to this fact B-tree is well suitable for the database which are frequently updated. Thus the performance of DW is depends upon the selection of appropriate indexing structure for its design.

The remaining paper includes following sections. In section II we have discussed about Bitmap Index(BI), section III about B-tree index, section IV Difference between B-tree and Bitmap index, Section V Performance Analysis of B-tree and Bitmap Indexing and section VI is Conclusion of the paper.

II. BITMAP INDEX(BI)

According to concepts described by [8,11,12,1] bitmap index is a matrix of 1 and 0 bits. The bitmap size depends upon the number of unique value attribute present in vector on which we are creating bitmap. Consider relation R, consist of number of columns (C1,C2,C3,.....,Cn) out of this we can select any column. The unique attribute from this columns are the bit map vectors. A bitmap index is a set of bit vectors, which are useful for a certain type of search on columns of a table. A bit vector is just a series of bits, e.g. 010101010001111. Table(I) shows Basic relation and table (II) represents the bitmap vector on position column.
In this way once the bitmap is created it will be useful for next all queries which work on same attribute. Bit map index support all logical operations like OR, XOR, AND and NOT. These operations directly perform on hardware so it is very fast. Due to this hardware compatibility of bit map index various researchers prove the applicability of bitmap index on Data warehouse.

In [22], researchers proposes Scattered bit map indexing strategy which is variant of Bitmap index. It requires less time to execute and less memory to store scattered bitmap. This approach is most suitable for queries which wants to find out equality and membership between the vectors.

In 2008[23] introduces concept of adaptive bitmap indexes. This includes multi resolution optimal bitmap index and group of auxiliary projection index. These are used while removing false positive from current query result. It has been kept in LRU(Least recently used) cache in memory or disk which will be useful for subsequent queries.

In 2008[24] introduces dynamic encoded bitmap indexing strategy. This is based on query history. They kept lookup table of query attributes, next time for same query the answer is given on the basis of entry in look up table data.

In 2009[19] suggest that the BI is well suited for DW queries because BI has ability to perform operations on bit level rather retrieving original table. They prove that it is suitable for COUNT based as well as statistical queries. It is work on bitmap not on huge table IO access time is reduced and the efficiency of query increases.

It has been concluded in [25] bitmap index has three advantages for using it in DW are: It avoids complete table scan and saves disk access. It saves computational time as it work on bitwise operation and BI make use of antimonotone property of IBQ and develop different index pruning strategy. In this paper they make use of compressed bitmap index concept which save the memory and shows the effectiveness of BI for IBQ evaluation.

In [26] researchers prove that BI is time efficient for answering IBQ which we are firing on DW. It performs so as it works on index level rather on original table.

### TABLE I BASIC RELATION

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Position</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sudha</td>
<td>First</td>
<td>15000.00</td>
</tr>
<tr>
<td>2</td>
<td>Subodh</td>
<td>First</td>
<td>20000.00</td>
</tr>
<tr>
<td>3</td>
<td>Radha</td>
<td>Third</td>
<td>10000.00</td>
</tr>
<tr>
<td>4</td>
<td>Megha</td>
<td>Second</td>
<td>12000.00</td>
</tr>
<tr>
<td>5</td>
<td>Priya</td>
<td>Third</td>
<td>10000.00</td>
</tr>
<tr>
<td>6</td>
<td>Anu</td>
<td>Second</td>
<td>11000.00</td>
</tr>
<tr>
<td>7</td>
<td>Ram</td>
<td>First</td>
<td>17000.00</td>
</tr>
<tr>
<td>8</td>
<td>Raj</td>
<td>Third</td>
<td>11000.00</td>
</tr>
</tbody>
</table>

### TABLE II BITMAP INDEX ON POSITION COLUMN

<table>
<thead>
<tr>
<th>Id</th>
<th>First</th>
<th>Second</th>
<th>Third</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

B-Tree index uses a balanced tree structure for efficient record retrieval. B-tree indexes store key data in ascending or descending order. A B-Tree index stores the index value and the physical row id of the row. The index values are arranged in the form of leaves. A B-Tree index is used most when the cardinality is high. It stores the index pointers and values to other index nodes by using recursive tree structure. As pointers are placed on each node so information is retrieved using tracing on the pointers. The top level of index is root of the tree. The lowest level of tree is leaf node. All other levels in between the tree are branches of the tree or internal nodes of the tree. Root and internal nodes contain pointers which points to next level in the index. Leaf node consist of index key and pointer to physical location of the respective data.[1]

#### B-TREE INDEXING

Due to such a structure using searching, update and delete becomes very easy in case of B Tree indexing. It maintains a level wise structure and height is fixed according to size of index. When we want to insert values into table the space is created as per the request in respective level it increases index but level/height of tree remains as it is. It doesn’t affect other attributes index value and not huge modification is required in database design. Similarly for delete operation respective index value is deleted and previous pointer is allocated to next node. It is just replacement of pointer, no additional structure as well as format is needed. Due to this working nature of B-tree it is suitable for databases where frequent changes are expected like OLTP systems. For each B-tree operation, the number of disk accesses raise with the height of the B-tree, which is kept low by the B-tree operations B-tree is widely used in a relational database environment but it cannot handle efficiently on a large amount of data which causes memory overhead in complex and interactive queries.

Based on researchers studies [7,8,9], B-tree index has features that make it a well suitable on columns with high cardinality values like OLTP system.

#### IV. DIFFERENCE BETWEEN B-TREE INDEX AND BI

According to the study of researchers[11,12,16,17,21,20], in this section we are showing the difference between B-tree Index and Bit map Index.
TABLE III COMPARISION TABLE

<table>
<thead>
<tr>
<th>Sr.No</th>
<th>B-Tree Index</th>
<th>Bit map Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>B-Tree index uses a balanced tree structure for efficient record retrieval.</td>
<td>Bit map uses a matrix structure with Zero and One bit value.</td>
</tr>
<tr>
<td>2.</td>
<td>It is not possible to index NULL value in B-Tree Index.</td>
<td>Bit map index can index NULL value.</td>
</tr>
<tr>
<td>3.</td>
<td>Compare to bitmap index it is slow as it is not directly work on logical operations.</td>
<td>Bit map indexing is fast as it operate on logical operations. Logical operations are directly compatable with hardware system</td>
</tr>
<tr>
<td>4.</td>
<td>Update, delete and insert operation does not completely affect the structure of B-Tree</td>
<td>Any modification in table completely affect the structure of Bitmap index. We have to create again the bitmap index.</td>
</tr>
<tr>
<td>5.</td>
<td>B-tree index updates on key values has relatively inexpensive</td>
<td>Bitmap index update/delete has more expensive</td>
</tr>
<tr>
<td>6.</td>
<td>B-tree Index has High cardinality values(usually when we have repeated columns),</td>
<td>Bitmap Index has Low Cardinality values(usual when column have few values like true/false).</td>
</tr>
<tr>
<td>7.</td>
<td>B-tree Index is useful for OLTP</td>
<td>Bitmap Index is useful for Dataware Housing</td>
</tr>
<tr>
<td>8.</td>
<td>Performance of B-Tree is degrades for nested queries.</td>
<td>Bitmap indexes provide better performance than b tree indexes when queries often use a combination of multiple &quot;where&quot; conditions involving the any logical operators like AND, OR and NOT.</td>
</tr>
</tbody>
</table>

V. PERFORMANCE ANALYSIS OF BI AND B-TREE INDEX

All the databases uses B-Tree as a standard indexing technique in database design[1,21,18]. In these system if we create index on any column it by default is B tree. As we have studied that B-tree is suitable for OLTP system not for DW system. In this section we will show the performance difference of B-Tree and Bitmap indexing by assuming example query and its result. This sections proves that the bitmap index is the perfect choice for data warehouse design. To check this we have done following experiment.

Step 1: We create a table empindex

CREATE TABLE empindex
( Empid int(50),
Empcode char(10),
Type int(10),
EmpName char (10));

Created a table with 100000 records and have a Empid column that has 100000 distinct values and a TYPE column with just 4 distinct values.

Step 2: Execute query on empindex table.
Select * from empindex;
In this case record count is 100000.

Step 3: Created B-Tree index on the Type column and check the size of the index.

CREATE INDEX empindex_type on empindex(Type);

Step 4: We check the size of B-Tree index using following query.
Select index_name,index_type,distinct_key,blevel,leaf_blocks from dba_indexes where index_name =”empindex_type “;

Also we have checked the status of table by executing following query:

Show Index from empindex\G;
The output of this query is:

<table>
<thead>
<tr>
<th>INDEX_ NAME</th>
<th>INDEX_TYPE</th>
<th>DISTINCT_KEYS</th>
<th>BLEVEL</th>
<th>LEAF_BLOC K</th>
</tr>
</thead>
<tbody>
<tr>
<td>empindex _type</td>
<td>normal</td>
<td>4</td>
<td>1</td>
<td>176</td>
</tr>
</tbody>
</table>

Step 5: In this step first up all we dropped the B-Tree index which we have created in last step. We created bitmap index using following commands.
Create bitmap index empindex_type_bmindex on empindex(Type);

Step 6: For checking the size of bitmap index we execute following query.

Select index_name,index_type,distinct_key,blevel,leaf_blocks from dba_indexes where index_name =”empindex_type_bmiindex “;
The output of this query is:

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>INDEX_TYPE</th>
<th>DISTINCT_KEYS</th>
<th>BLEVEL</th>
<th>LEAF_BLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>empindex_type_</td>
<td>bitmap</td>
<td>4</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>bmindex</td>
<td>bitmap</td>
<td>4</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

From the output of both above query we can notice that the block size get reduced from 176 to 10 in case of Bit map indexing. This is very huge difference in block size. In this case Type column has low cardinality i.e it has only 4 distinct values. Similarly we tried for Empid column which has high cardinality with 100000 distinct column in this case the B-Tree indexing performance is better than Bitmap indexing (BI).

VI. CONCLUSION

By considering the results of above query we come to the conclusion that bitmap indexes are more suitable for tables which has thousands of rows means a huge database like Data warehouse system. It is more suitable for DW due to property of data warehouse that it is non volatile i.e updates are not frequent. Because of this in DW if Bitmap is created on some column it remains as it is and will be used again when required by subsequent queries. There is no need to create the bitmap again and again which reduces query execution time. Another aspect of bitmap indexing is it create the bitmap again and again which reduces query when required by subsequent queries. There is no need to on some column it remains as it is and will be used again.

REFERENCE

5. C. DELLAQUILA and E. LEFONS and F. TANGORRA, Design and Implementation of a National Data Warehouse. Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, February 15-17, 2006 pp. 342-347
8. R. Strohm, Oracle Database Concepts 11g,Oracle, Redwood City,CA 94065, 2007.
18. Oracle @Database Administration’s Guide,”Managing Indexes”,http://docs.oracle.com