An Efficient Method for Improving Query Efficiency in Data Warehouse

Zhiwei Ni[1,2]
1)School of Management, Hefei University of Technology, Hefei , Anhui, China
2)Key Lab. of Process Optimization and Intelligent Decision-making, Ministry of Education, Hefei , Anhui, China
zhwnelson@163.com

Junfeng Guo[1,2], Li Wang[1,2] and Yazhuo Gao[1,2]
1)School of Management, Hefei University of Technology, Hefei , Anhui, China
2)Key Lab. of Process Optimization and Intelligent Decision-making, Ministry of Education, Hefei , Anhui, China
alloy1129@yahoo.com.cn, wl820609@163.com, yazhuogao@163.com

***Abstract*—There are lots of performance bottlenecks for real-time queries in mass data. Many methods can only improve the efficiency for frequently used queries, but it is not advisable to neglect the non-frequently used queries. This paper proposes a new integrated index model called BBI and illustrates the application of this model. Based on the feature of data warehouse and OLAP queries, this index model is built with inverted index, aggregation table, bitmap index and b-tree. It greatly promotes not only the efficiency of frequently used queries, but also the performance of other queries. The analytical and experimental results demonstrate the utility of BBI.**

***Index Terms*—Aggregation Table, Inverted Index, Bitmap Index, B-Tree Index**

I. INTRODUCTION

Data warehouse (DW) is defined as a subject-oriented, integrated, steady and time varying data set which supports enterprises or organizations to make decisions. As the decision maker needs to query several values from one subject for real-time analysis processing, the multidimensional model of DW is usually implemented as star schemes to meet the requirements. This kind of hierarchical model is highly unnormalized and query-oriented. There are two kinds of table in star schemes. One is fact table which contains basic quantitative measurements of a business subject, the other is dimension table that describes the facts. If there are more than one fact table in a DW, it can be called galaxy model which is actually constituted of several star schemes.

Complex queries are always requested in DW. When users need to process multi-dimensional analysis, multi-table joins may be involved. Although data can be stored in a multi-dimensional database, DW usually stores data in the form of relational database. As the number of dimension and the overall size of data sets increase, the size of DW often grows to gigabytes or terabytes. When the complex queries are implemented on mass multi-dimensional data， the query efficiency is far beyond satisfaction. Fact tables in data warehouses which store business measures usually have millions of records or more. Such tables usually have more than 10 attribute dimensions. For example, to select records of which time between 01/01/2008 and 09/10/2009, it needs to join the tables and compare the time value of millions of each record with the requested value. So it is necessary to retrieve the data more efficiently. Some methods and technologies were proposed to improve queries efficiency, such as materialized view[1], feature selection[5], index technology[3], etc.

Materialized view is a kind of pre-computed structure, it materializes the calculated results ahead of using. The pre-computed values are often mean, sum, average, etc. Queries on materialized views are fast responded, because no join needs to be made on successive requests, and the records in views are less than the original tables. Materialized views can be applied to OLAP, but due to the limit of storage space, it is infeasible to store results of all queries. Some heuristic algorithms have been used to find an approximate optimal solution. For example, greedy and genetic algorithms that based on requirement and probability are applied to generate views. But once queries are made on the records which are not materialized, the efficiency can not be improved, and it is unacceptable for any delay when users need the results urgently. So there is limitation of materialized method.

Feature selection is a procedure to select a subset from the original feature set by eliminating redundancy and less informative features so that the subset contains only the most discriminative features [4]. Applied to dimension reduction, a set of attributes that best represents the overall data set is found out by feature selection. But feature selection has the same problem with materialized view that when the queries involve the dimensions which are not selected, the efficiency of this method deceases.

Index technology greatly cuts down the load of I/O, which is highly effective in real application. Compared with materialized view, the space cost of index is reduced. Many indexes are classified into data-partitioning indexes, such as B+ tree and R-tree family and other tree indexes. B+ tree indexes are often adopted in databases to retrieve rows of a table with specified values involving one or

doi:10.4304/jsw.6.5.857-865

more columns. Data are placed in some partitions by the sequence of key values which also need to be pre-computed. Using these indexes to answer a query, system should find the partitions which contain the related data by comparing the key value from the root node to the leaf nodes. Those all search paths that may be potentially matched must be explored. Data-partitioning indexes are effective for single keyword queries. As the dimension quantity of both the indexes and the queries increases, the efficiency deceases.

OLAP query includes point query and cube query. It takes less time to process point query whose result is a single value or one record. Cube query returns a list of values or multiple rows of data which are aggregated from the data set. It needs to traversal the whole data table to get the data set and is relatively long in the duration of query implementation. For example, if we need the total revenue which is sorted only by time and location, it is a cube query that needs to traversal the whole sale fact table to aggregate the data. As for OLAP query，the common index technology can not meet the requirements of query efficiency. Data cube plays an essential role in fast OLAP query, but high dimensional data cube requires massive memory and disk space, and the current algorithms are unable to materialize the full cube under such conditions. It is called curse of dimensionality. Besides, it is hard to build data cubes based on relational databases, so how to promote the query efficiency becomes the key problem. As for OLAP application，paper [17] used inverted index which is common technology of search engine to build shell cube. Though the new cubing approach reduces the space cost of data cube and promotes the efficiency of queries, many queries require to be computed at run time.

Introducing inverted-index in search engine technology, integrating join-index and materialized view, we propose a new index model in this paper based on user interest and query statistics, called BBI. The new index can not only greatly promote the efficiency of frequently used queries, but also improve the performance of other queries. With a good performance in storage size, it is suitable to be applied in OLAP and other complex queries.

In Section 2, the relevant knowledge of existing approaches and technology are presented; our approach is elaborated and the efficiency of the theory is analyzed in Section 3. BBI is further discussed in Section 4. Section 5 describes the performance evaluation of our approach. Section 6 is the conclusion.

II. RELATED WORK

There are some feasible methods with acceptable performance for the queries that require multi-table joins in high dimension data warehouses when the queries are based on the dimensions that are important or commonly used, but not all dimensions can be included with most methods while the space expense grows rapidly as the dimension grows. The common technologies are view materialization, feature selection, index technology, etc., they are not independent with each other, and many researches concentrate on integrating various technologies to improve the efficiency of DW.

A. Materialized View

Dynamic materialized view [7] selectively materializes only a subset of rows which are the most frequently accessed. Compared to conventional materialized view which maintains all rows of a view, the set of dynamic materialized views can be changed dynamically and the storage space is reduced. A method in [8] materializes the views in a data warehouse to reduce the query response time. Aiming at the insufficient consideration of the dynamic update, the method can wash out materialized views and add new materialized views in the set of current materialized views on the basis of the greedy algorithm. Paper [9] proposes a constrained evolutionary algorithm for materialized views. Constraints are incorporated into the algorithm through a stochastic ranking procedure [18]. The basic principle of this technology has been well described in [10] and [11]. But [7], [8], [9], [10], [11] all avoid the question of generality.

An algorithm is presented in [1] for building materialized sample views for database approximation. The core technique is called ACE (Appendability, Combinability, and Exponentiality) Tree, improved from B+-tree, that is suitable for organizing and indexing a sample view, but the algorithm is not useful when integrated views are needed.

B. Feature selection

Feature selection techniques are targeted at finding a set of attributes that best represent the overall data. [2], [4] and [19] are traditional techniques of feature selection which focus on maximizing data energy or classification accuracy for dimension reduction [19]. The algorithm in [4] groups the features into different clusters based on feature similarity and selects a representative feature from each cluster, so that the feature redundancy is reduced. As a result, selected features may have no overlap with queried attributes. In this case, to neglect any attribute may bring troubles when queries are based on the attributes that have not been selected.

C. Index

Patrick O'Neil and Dallan Quass presented a review of indexing technology in Paper [6] which included join index, bitmap index and B+-tree. Two indexing structures called Bit-Sliced indexes and Projection indexes were introduced as well. The core of the method was the combination of B+-tree and bitmap index, which was not complex but effective.

At present, B+-tree and bitmap index are two of the most important indexes in most database software. In fact, the new edition of ORACLE database has involved the improved indexing technology. The key indexes of ORACLE are shown in Fig. 1 and Fig. 2. Fig. 1 shows the application of B-tree index. Based on the rules of B-tree, relational table is divided by a numerical attribute, all rows of the table are distributed in the leaf nodes of a tree. In leaf nodes, the values of indexed attribute are considered as labels. Each label corresponds to a value called 'rowid' which is used to uniquely identify one row of data in ORACLE database and can be seen as physical addresses. Leaf nodes are connected by indicator so that it

is convenient to use range query. When queries are made on keywords, it is rapidly to find the 'rowid' in leaf nodes by b-tree rule and localize the rows of data in database by 'rowid'. B-tree index in ORACLE is suitable for high-cardinality columns and it is inexpensive to update on keys relatively. But it is inefficient for queries using 'or' predicates.

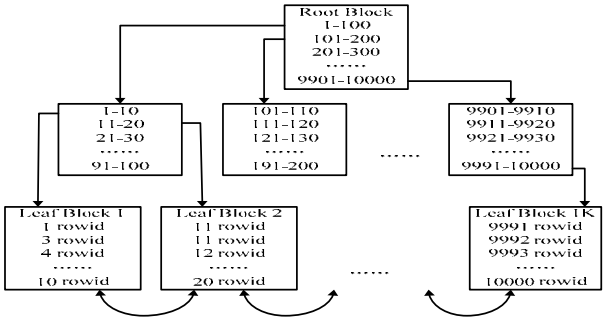

Fig. 1. B-tree index in ORACLE

Fig. 1. shows the structure of bitmap index which is built for gender in ORACLE. The word 'start' and 'end' represent a segment of storage space. The 'rowid' of '8.0.1' is the beginning address and '12.0.1' is end. Bitmap shows the distributing of all the gender keywords. '1' means the keyword appears in the row, '0' means the contrary. Bitmap index in ORACLE is suitable for low-cardinality columns and it is efficient for queries using OR predicates. But it is expensive to update key column. At the same time, each keyword needs a bitmap whose length is the same as the fact table. When the row number is large, it is difficult to use and store bitmap. ORACLE often uses the subsection structures to solve this problem which also shows in Fig. 2,but as the number of segment increases, performance decreases quickly, because it needs to checkall the bitmap segments to find answer.

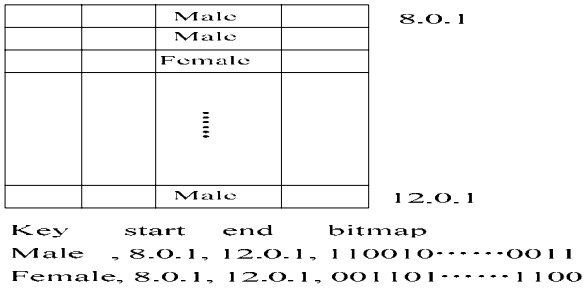

Fig. 2. Bitmap index in ORACLE

Several papers proposed designs for index recommendations based on optimization rules [14], [15], [16].Since the effectiveness of these indexes degrades when the query patterns change, Michael Gibas and his collaborators [5] introduced a technique to recommend indexes based on index types that were frequently used for high-dimensional data sets and to dynamically adjust indexes as the underlying query workload changes.

Paper [3] evaluated experimentally the dimension-join indexes using the TPC-H benchmark and showed a new index structure using bitmap and B+ tree that can dramatically improve the performance for some queries. Papers [12] proposed Hybrid Index which is suitable for given data by building with B+ tree and hash table, but it is not universality. Paper [13] introduced a new kind of multi-table join index and proved it was more effective than only using multi-table join by experiment. But this technique also only has the index on the frequently using data to advance the query performance; it upgrades nothing when queries are made on the data which has not been paid attention.

III. APPROACH

A. Inverted Index

Inverted index comes from the functional needs of attribute retrieval in practice. Each item in the index table has an attribute value and an ID which corresponds to the value. Inverted index tables identify ID by attribute, which is different from the common tables that identify attribute by ID, that is why it is called inverted index. Table 1 is a fact table which has three dimension columns and one numerical value column. Table 2 is the first order inverted index of table 1. The keyword 'male' appears in line 1, 2, 5 in table 1; so there is one line in table 2 which records 'male' and '1,2,5'. The rest can be deduced accordingly. It is easy to find that the storage space of inverted index smaller than that of the original table because it does not include the numerical value column.

Table 1. Fact Table

TID	sex	age	specialty	score
1	male	20	computer	90
2	male	20	computer	74
3	female	19	computer	83
4	female	20	computer	95
5	male	19	computer	81
6	female	20	computer	70

Table 2. First-order Inverted Index

Word	TID
male	1,2,5
female	3,4,6
19	3,5
20	1,2,4,6
computer	1,2,3,4,5,6

Second-order and high-order inverted index can also be built. Table 3 shows the second-order inverted index of sex and age. The TID of male and 19 is 5, it tells that male and 19 both appear in line 5. The advantage of inverted index is fast keyword retrieval.

Table 3. Second-order Inverted Index

Word	TID
male ,19	5
male ,20	1,2
female,19	3
female,20	4,6

Table 2 can be translated into Table 4 by using bitmap, which can further decrease the storage space and increase the query efficiency.

Table 4. Bitmap-based Inverted Index

Word	TID
male	110010
female	001101
19	001010
20	110101
computer	111111

B. Definitions

Definition 1 High-frequency Join

Queries which relate to the same dimensions are classified into the same category. If the ratio of the number of queries in the same category to the total number of all queries is over HFJ, the category is called high-frequency Join. HFJ is the threshold of high-frequency which is between 0 and 1. The lower the HFJ is, the more the high-frequency Join there are, which leads to the large space occupation. The setup of HFJ depends on the real cases.

Definition 2 Fact table can be formalized as Table (Dimx1, Dimx2,…,Dimxn, M1,…,Mm). Aggregation tables are derived from the original fact table, indicating to AgTable (Dimx1, Dimx2,…,Dimxi, f(M1),…,f(Mm)). Dimx1,Dimx2,…,Dimxi indicates the remaining dimensions after the aggregation. f ( ) is the aggregation function. There are (n-i) dimensions in AgTable, that is less than original fact table.

C. Approach Overview

The abstract of our approach is as follows: In condition of the limited storage space, inverted index is built for fact table in database while aggregation table is established for frequently used queries, aggregation table is applied for frequently used queries to get quick response and inverted index is adopted for other queries to reduce response time. So it improves the efficiency of both frequently used queries and the other queries.

This paper establishes aggregation based on the frequency of queries from users and adopts bitmap to optimize inverted index. A solution is proposed for the inefficient performance of bitmap when the number of records becomes too large. The approach has the advantages as quick response for frequently used queries, performance promotion for other queries and reasonable space cost.

D. Application of Our Approach

The establishing process of our approach can be mainly divided into three steps: getting high-frequency Joins; building inverted indexes; building aggregation tables and join indexes. If better space performance is required, then only the aggregation table for those high-frequency joins meet the threshold α will be built (α is HFJ which has been introduced before). The function of α is to reduce the inconsequence of building too many aggregation tables. For some high-frequency joins, the number of involved dimensions is slightly smaller than the total number, there is only tiny advancement by applying aggregation table for them. So this kind of high-frequency joins will be ignored while building aggregation tables.

Algorithm 1 Creating Index and AgTable
Input: Fact Table, Query Set
Output: Inverted Index, AgTable(Dimx1,Dimx2 …Dimxi , f(M1),…,f(Mm))

```
begin
  (1)Scan Query Set → Get High Frequency Dim and
     High Frequency Join
  (2)Scan Database → Create First-order Inverted
     Index
        IF space is surplus
          For each High Frequency Dim
          Create High-order Inverted Index（X≥2）
        End
  (3)For each High Frequency Join
        IF i< α *N
        Create AgTable(Dimx1…Dimxi,
f(M1)…f(Mm))
        Create join index for AgTable;
        End
End
```

To apply this approach, first of all, make a judgment whether the aggregation tables in database can be used. Only if the aggregation table contains all the dimensions of the query, it can be to promote the queries efficiency (Reading join index to RAM and using join index to rapidly create aggregation tables). If there is no aggregation table can be used, we use inverted index (containing the high-order inverted index) to get results. It not only greatly promotes the efficiency of OLAP queries, but also improves the performance of multi-keyword queries.

Algorithm 2 Using index and AgTable
Input: Query Set
Output: Result

```
Begin
  Read join index and inverted index into RAM
    For each Query
      If  dimensions are all in AgTable
      Select from AgTable
    Else
      Use Inverted Index (first-order or high-order) to
get address
      Select from hard disk (Fact table) use address
  End
End
```

E. Analysis

Comparing to materialized view, our approach costs less space. The added space is cost by aggregation tables and inverted index, but bitmap greatly reduces the storage space. In the algorithm above, it reduces the space cost by restricting the ratio of joins' dimensions to total dimensions in aggregation tables below α. So a theorem can be derived when each attribute in different dimension tables fits together in fact table.

Theorem 1 The ratio of aggregation table to fact table is getting smaller rapidly with the increase of rows of any dimension table or the number of dimensions.

Theorem 2 Fact table and the structure of our approach have the same the space complexity.

Time complexity analysis is as follows: When there is no index, Cube queries of OLAP need to join the fact table with dimension table to build a temporary view. A summary query needs a traversal of the whole table, so the time complexity is O (Mn). When our approach is adopted, there are two situations:

(1)If the aggregation table can be used, it costs time to search for aggregation and query on it. As there are join index in RAM, it can quickly find out the aggregation, so the time is mainly spent on the second step. Query by using aggregation, the time complexity is O (Mn-i), 'i' is (1- α)*n which stands for the difference of dimension number of fact table and aggregation table. O (Mn-i) is much smaller than O (Mn), the more the dimensions in the dimension table are and the bigger the fact table is, the more efficient our approach is.

(2) If no aggregation table can be used, then inverted index can be adopted by OLAP query. The total time can be divided into three parts: the time of reading out the result id set from inverted index, the time of getting the intersection of result id set, the time of querying on the fact table with id. If inverted index is expressed by bitmap, time for intersection can be omitted, so the total time complexity is O (M*n). Otherwise, the intersection time should be considered. In fact, the time for calculating is much less than reading time. But for convenience, we suppose they are in same level. If we use dichotomy to sort one set, the time complexity is O (M $^{\log}$ M), so the time for sorting all sets is O (nM $^{\log}$ M). The time for getting common elements on N ordered set is O (M*n). At last the total time complexity is O (nM $^{\log}$ M+ M*n), equal to O (nM(1+$^{\log}$ M)). It is still much smaller than O (Mn).

IV. BBI

The application of bitmap in expressing inverted index decreases the space cost, as well as promotes the efficiency, but there are two problems for using bitmap on huge database. Firstly, every keyword needs a long bitmap with the same length of the whole table. If there are many records in a table, it is hard to store and use the bitmap. Suppose there are 10000000 records in the table, however one keyword needs 125000 bytes for bitmap, getting the intersection of two 125000 bytes bitmap is also difficult. Secondly, high-cardinality columns like time dimension have many keywords and it is common that most keywords appear rarely, so it wastes much space by using bitmap inverted index to express each keyword. For the above two problems, we made an improvement on our approach. We use bitmap, b-tree and inverted index and apply two different methods to high-cardinality columns and low-cardinality columns, called BBI.

A. BBI index for low-cardinality columns

For low-cardinality columns' keywords, the number is not large and the frequency is high. For these columns we use the BBI index structure as shown in Fig. 3. It divides the fact table into some segments which still can be divided into many blocks. Index blocks are established on each table block by using inverted index and bitmap, containing the initial and ending address of the block. One bit of the bitmap refers to one row and stands for whether a keyword appears in that row. After establishing all the index blocks, it establishes index segment for the index blocks. Index segments has the similar structure as index block, but one bit of the bitmap refers to one block and stands for whether a keyword appears in that block. It establishes chief index on the total index segments and also uses a bit to stand for whether a keyword appears in that segment. BBI is an open index, using the same method, it can build higher order index on the table, but not only third order.

Such as Fig. 3, it establishes index block for each 256 rows, and builds index segment for each 64 index blocks; at last built a chief index for 64 index segments.

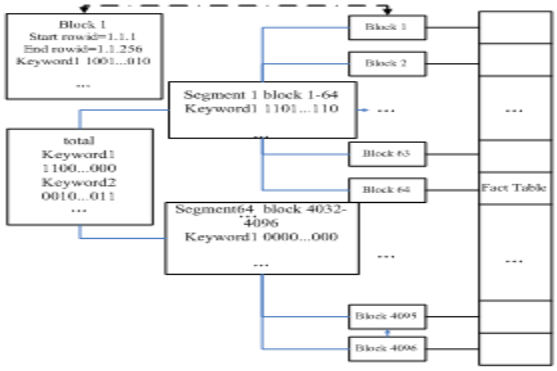

Fig. 3. BBI index for low-cardinality columns

The main space cost is the storage space of index blocks, because the space of index segments and chief index is only a small portion of the index blocks' . We can see from Fig. 3 that establishing BBI for one column costs the same space as building bitmap index of ORACLE. Suppose there are N rows in the fact table, the total number of keywords of all the low-cardinality columns is M, K bytes bitmap is used for one keyword in each block, so it comes that:

$$Space = \frac{N}{8K} * M * K = \frac{NM}{8} \quad (1)$$

It can be derived from (1) that the space of BBI is only related to the number of rows and the number of keywords. Suppose there are one million rows and one thousand keywords, the space cost of BBI for all the keywords is 125M. Compared to the space of table with one million rows, it is much smaller.

B. BBI index for high-cardinality columns

For high-cardinality columns' keywords, the number is large but the frequency is low. For these columns we use the BBI index structure like Fig. 2. It also divides the fact table into many blocks. Index is established by using

B-tree, inverted index and bitmap. Index is organized by b-tree and the inverted indexes are stored in the leaf nodes. According to the property of high-cardinality columns, it will waste too much space to establish whole inverted index for all the keywords in all the rows, so it only stores the bitmap for the block in which the keywords appear, containing the tab of block. All the leaf nodes are linked by indicator for range query. Such as Fig. 4, there are 3000 keywords in one column, it only find out the id of the keywords in dimension table. The keyword '1' exists in some rows of block 5 and block 121, so there are two bitmaps in leaf node 1 for keyword '1'. Each leaf node stores the bitmaps of 10 keywords at most, sorting by ID.

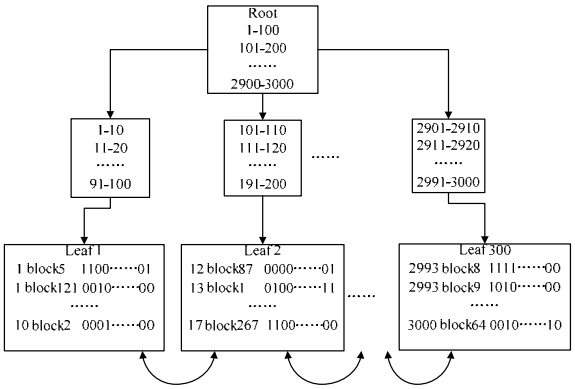

Fig. 4. BBI index for high-cardinality columns

The main space cost is the storage space of index leaf nodes. Suppose there are N rows in the fact table, the total number of keywords of all the low-cardinality columns is M, each keyword may appear in L blocks on average (it means each keyword needs L rows of bitmap in the leaf node on average), it uses K bytes for each row of bitmap, so it comes that:

$$\text{Space} = M * L * K \quad (2)$$

It can be seen from (2) the space of BBI for high-cardinality columns is related to size of one block, the distribution of keywords and the number of keywords. Suppose there are 100000 keywords, each keyword may exist in 10 blocks on average and each bitmap is expressed by 32 bytes, the space cost of BBI is 30M.

Two extreme conditions are considered. The first condition is that every keyword exists in all blocks. It is obvious that the space cost can be calculated by (1). The second extreme condition is that for one column, the keyword in each row is different from each other. So the number of keywords is the same as row number. If the row number is N and the dimension number is D, we can derive that:

$$\text{Space} = N * D * K \quad (3)$$

Suppose there are one million rows and one hundred high-cardinality columns with the second extreme condition, it uses 32 bytes for one bitmap; the BBI space cost of these a hundred million keywords is 3.2G. Comparing to the space of table with one hundred high-cardinality columns, it is much smaller.

V. PERFORMANCE EVALUATION

In this section the performance evaluation of BBI is illustrated. In experiments, we used both real data and simulated data to verify the space and time performance with the variation of row number and dimension number. The experiments are performed on fedora core 10.0 with Inter(R) Core(TM)2 2.83G CPU and 2GB RAM, data is stored on a local disk which is 7,200-rpm. The space of simulated data is 1622.8M which contains 5 million rows and 22 dimensions, including 11 high-cardinality columns and 11 low-cardinality columns, it is randomly generated. The real data set contains 7.79 million rows and 12 columns, and the space is 1080M.

A. IO performance of BBI

For retrieval speed, one of the most important influences is I/O cost. On the one hand, if it is not consecutively stored in physical address, the efficiency of disk I/O will decline. On the other hand, it costs huge I/O for querying on huge data, so using BBI to decrease I/O cost can increase the retrieval efficiency.

Fig. 5 shows the change of I/O cost when establishing BBI in stages. As the number of columns on which BBI established grows, 100 queries are repeatedly executed on the simulated data.

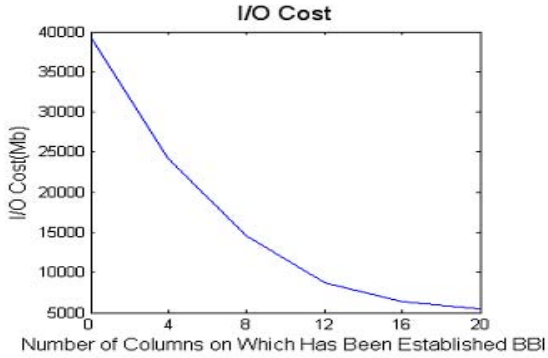

Fig.5. I/O Cost

In fact, if all the data is read into RAM at a time, it doesn't need disk I/O any more if the following queries are executed on the same data. But it is hard to read all the huge data to RAM, so only the essential data is read at a time in the experiment, and released after query. When there is no BBI, the I/O cost is close to 40000M. When there are 4 columns with BBI, the I/O cost decreased to 24100M. When there were 20 columns with BBI, the I/O cost is only 5400M.

B. Space size of aggregation table

We use the simulated data and simulated queries in this experiment. We vary HFJ to see the change of high-frequency join number. Fig. 6 shows change tendency of ratio of high-frequency joins to all joins when HFJ ranges from 1% to 5% and the columns number are 6,10,14,18,22 respectively. There are 10 high-frequency joins when there are 22 columns and HFJ equals to 0.5%, but there are only 3 when HFJ equals to 2%. No high-frequency joins exist when HJF increases to 4%. It shows the

common discipline by using simulated data and simulated queries which are all randomly generated.

Fig. 7 shows the storage size of aggregation table, α =2/3. The storage size of aggregation table decreases rapidly with the increase of HFJ, but the tendency is not regular. Through analysis we can see that high-cardinality columns often affect more than low-cardinality columns, because high-cardinality columns contain more keywords. So if a high-cardinality column is deducted, the storage size of aggregation will drastically decrease. In the condition of 22 columns, when the HFJ increases from 1.5% to 2%, the number of high-frequency joins decreases 1, but the aggregation table size decreases 200M. At the same time, when HFJ increases from 2.5% to 3.0%, the number of high-frequency joins also decreases 1, but the aggregation table size only decreases 7.1M.

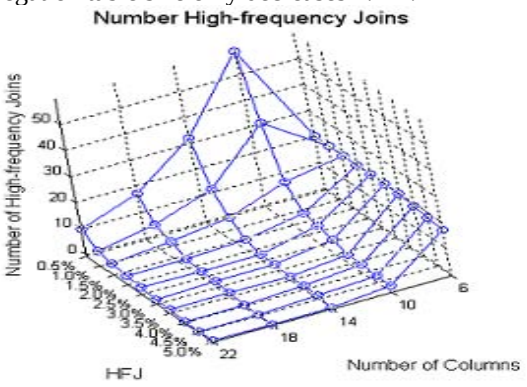

Fig.6. Number of High-frequency Joins

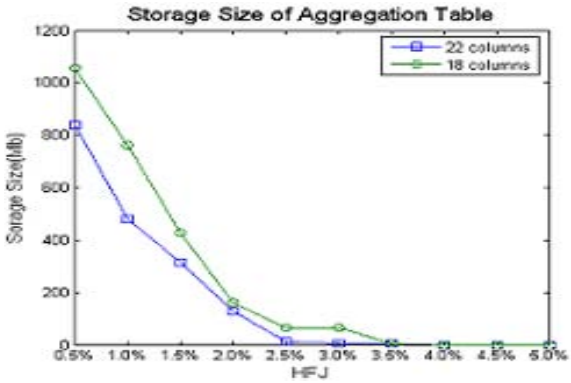

Fig.7. Storage Size of Aggregation table

C. Performance of our approach

In this section, the overall test is illustrated. Our approach has two functions, one is only to establish BBI as common index, and apply it to promote the efficiency for querying on huge data, the other is to use BBI and aggregation table to promote the efficiency for OLAP query. Based on real data, we first establish BBI, B-Tree index, traditional bitmap index and compare their efficiency for kinds of queries, then verify the OLAP performance of our approach.

There are 7.79 million rows in the fact table. We choose 6 low-cardinality columns to establish BBI, B-Tree and traditional bitmap index, then 6 high-cardinality columns. Fig.8 shows the comparison of space costs. First two low-cardinality columns only have 9 keywords, both storage size of BBI and traditional bitmap are 12M, but B-Tree is over 100M. When two high- cardinality columns are added, keyword number is over 31000, the traditional bitmap already can not be built, its storage size is predicted to be over 38000M. At this time, though BBI also used bitmap, its size only increases to 550M. At last, the number of all keywords is over 8.2 million, the traditional bitmap is strongly unsuitable, but the storage size of BBI and B-Tree are 767M and 593M, both of which show the good space performance for huge number of keywords. We can see that BBI has good space performance no matter how many the columns are.

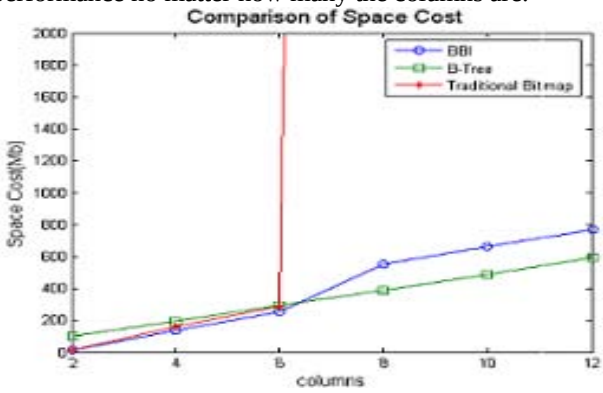

Fig.8. Comparison of Space Costs

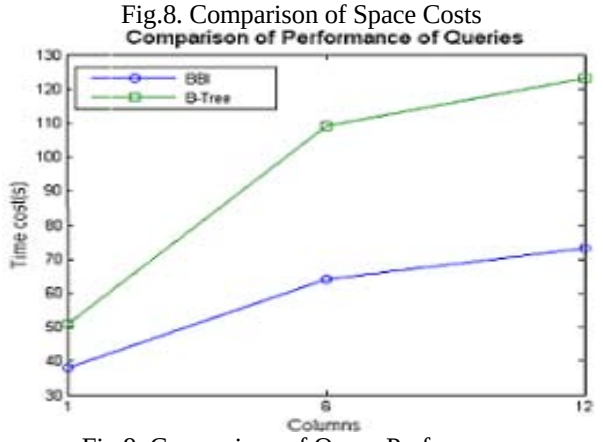

Fig.9. Comparison of Query Performance

As proved above, BBI has good query performance. Suppose there are three conditions, the queries involve 1,6,12 columns. We simulate queries for each condition. Fig. 9 shows the comparison of query performance of BBI and B-Tree. BBI always surpasses B-tree in the three conditions especially when the columns increase. BBI inherits the advantages of bitmap and b-tree, it could easily answer reply the range query, multi keywords query, ‘or’ query and etc.

At last, we verify the comprehensive performance (containing cube query for OLAP) of our approach. We establish the aggregation table on real data with the condition that α =2/3, HFJ=3%. For comparison, we also use greedy algorithm to establish materialized views which has the same space cost of our approach (aggregation table and BBI). Fig.10 shows the comparison of comprehensive performance. As cube queries increase, it turns up that some queries could not use materialized views and aggregation tables, but BBI is suitable for all the queries which could use inverted index to established data cube for OLAP (paper [17]).

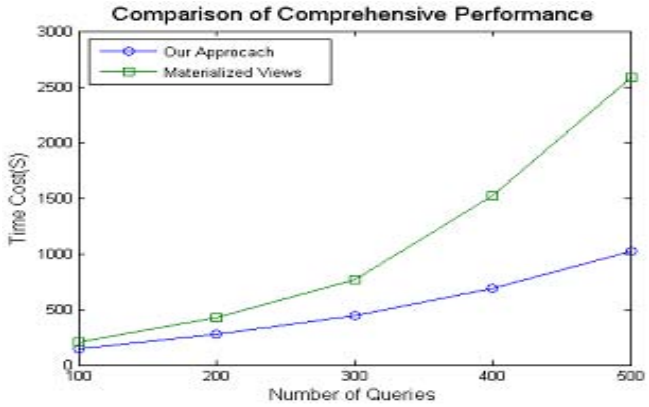

Fig.10. Comparison of Comprehensive performance

VI. CONCLUSIONS

A new approach is introduced in this paper, which can be used to decrease the time cost for frequently used queries, including cube queries. The core of the approach is BBI which is an integrative index, inheriting the advantage of bitmap index, b-tree index and inverted index. Inverted index represented by bitmap is adopted to get result quickly by intersection operations. Meanwhile, tree structure is used to accelerate speed for range queries. BBI can be built on both high-cardinality and low-cardinality columns and it is suitable to all data types. BBI established on different columns can cooperate with each other to promote the efficiency. Combining with aggregation which is based on user-driven, the approach can promote the efficiency for of cube queries as well. It does not only greatly promote the efficiency of frequently used queries, but also improve the performance of other queries. This paper discusses the space and time complexity of BBI, and the experiment results show good performance on space and time of this approach. Future work is planned to be focused on live update strategy.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant NO. 70871033 and the National High-Tech Research and Development Plan of China under Grant NO. 2007AA04Z116.

REFERENCES

[1] Joshi.S, Jermaine.C, “Materialized Sample Views for Database,” [J] IEEE Transactions on Knowledge and Data Engineering, Volume 20, Issue 3, pp: 337 – 351, March 2008

[2] Guangrong.Li, Xiaohua.Hu and etc, “A Novel Unsupervised Feature Selection Method for Bioinformatics Data Sets through Feature Clustering” Proc. Granular Computing, 2008. IEEE International Conference on 26-28 Aug. 2008(GrC 2008.), pp: 41 - 47

[3] Dimension-Join: A New Index for Data Warehouses http://www4.wiwiss.fu-berlin.de/dblp/resource/record/conf/sbbd/BizarroM01

[4] M.E Morita, R. Sabourin, F. Bortolozzi, and C.Y. Suen, “Unsupervised Feature Selection Using Multi-Objective Genetic Algorithm for Handwritten Word Recognition”, in the 7th International Conference on Document Analysis and Recognition, Edinburgh, Scotland, 2003, pp.666-670.

[5] Gibas.M, Canahuate.G, Ferhatosmanoglu.H, “On row Index Recommendations for High-Dimensional Databases Using Query Workloads” IEEE Transactions on Knowledge and Data Engineering, Volume 20, Issue 2, Feb. 2008 Page(s):246 - 260

[6] P·O’Neil, D·Quass. Improved query performance with variant indexes [EB/OL]. http://www.cs.duke.edu/~junyang /courses/cps216-2003-spring/papers/oneil-quass-1997.pdf,1997-05.

[7] Jingren.Zho, Larson.P.A, Goldstein.J, Luping.Ding, “Dynamic Materialized Views”, Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on 15-20 April 2007 Page(s):526 - 535

[8] Yin.GS ,Yu.X, Lin.LD, ” Strategy of Selecting Materialized Views Based on Cache updating”, IEEE International Conference on Integration Technology Shenzhen, CHINA, MAR 20-24, 2007 pp:789-792

[9] Jeffrey.Xu.Yu, Xin.Yao, ChiHon.Choi, Gang.Gou. “Materialized View Selection as Constrained Evolutionary Optimization”, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Volume 33, Issue 4, Nov. 2003 Page(s):458 - 467

[10] H. Gupta and I. S. Mumick, “Selection of views to materialize under a maintenance cost constraint,” in Proc. 7th Int. Conf. Database Theory,1999, pp. 453–470.

[11] A. Shukla, P. Deshpande, and J. F. Naughton, Materialized view selection for multidimensional datasets,in Proc. 24th Int. Conf. Very Large Data Bases, 1998, pp. 488–499.

[12] Byeong-Seob You, Dong-Wook Lee, et al. Hybrid Index for Spatio-temporal OLAP Operations[A] //International Conference on Advances in Information Systems(ADVIS 2006). Germany:Springer,2006:110-118.

[13] Wen Juan, Xue Yongshen, et al. An Efficient Method for Multi-Table Joining in Data Warehouse[J]. Journal of Computer Research and Development, 2005, 44(11): 2010~2017(in Chinese).

[14] M. Frank, E. Omiecinski, and S. Navathe, “Adaptive and Automated Index Selection in RDBMS,” Proc. Third Int’l Conf. Extending Database Technology (EDBT ’92), 1992.

[15] S. Choenni, H. Blanken, and T. Chang, “On the Selection of Secondary Indexes in Relational Databases,” Data and Knowledge Eng., 1993.

[16] A. Capara, M. Fischetti, and D. Maio, “Exact and Approximate Algorithms for the Index Selection Problem in Physical Database Design”, Knowledge and Data Engineering, IEEE Transactions on Volume 7, Issue 6, Dec. 1995 Page(s):955 - 967

[17] Xiaolei Li, Jiawei Han, Hector Gonzalez.High-dimensional OLAP:a minimal cubing approach[A]. NASCIMENTO M A,OZSU M T,KOSSMANN D,et al. International Conference on Very Large Data Bases(VLDB 2004).San Fransisco:Morgan Kaufmann,2004:528-539.

[18] Yan.Jun; Liu, Ning; Yan, Shuicheng; Yang, Qiang; Chen, Zheng;, “Synthesizing Novel Dimension Reduction Algorithms in Matrix Trace Oriented Optimization Framework”, Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on 6-9 Dec. 2009 Page(s):598 - 606

[19] Smalter.A, Huan.Jun, Lushington.G, “Feature Selection in the Tensor Product Feature Space”, in the ICDM '09. 2009 Page(s):1004-1009.

Zhiwei Ni (Tongcheng City, Anhui Province,1963), Professor, Doctoral supervisor.

He received his master degree from the Department of Computer Science and Engineering, Anhui University, Hefei, China, 1991 and a PhD degree from the Department of Computer Science and Technology, Hefei, China, in 2002, all in computer science.

He is currently a full Professor in the School of Management and also the Director for the Institute of Intelligent Management in Hefei University of Technology, Hefei, China. His major research interests include Artificial Intelligence, Machine Learning, Intelligent Management and Intelligent Decision Technique.

Junfeng Guo (Hefei City, Anhui Province, 1983), graduate student. He focuses on the research of business intelligence, data warehouse and OLAP (online analytical processing).

Li Wang (Fuyang City, Anhui Province, 1982), PhD student. His research field includes business intelligence, data mining and cloud computing.

Yazhuo Gao (Zibo City, Shandong Province, 1984), PhD student. Her research field includes business intelligence, data mining and OLAP.