A Survey on OLAP

K. Dhanasree  
Dept of CSE, DRKIST  
Hyderabad, Telangana, India  
bindusree77@gmail.com

C. Shobabindu  
Dept of CSE, JNTUA College of Engineering  
Anantapuramu, Andhra Pradesh, India  
shobabindu@gmail.com

Abstract—Online analytical processing is to-days major database technology that has completely changed the face of decision support systems. Many of the enterprise real-time analytical solutions are provided using most advanced OLAP methods. In this paper, we have presented the overview of the various OLAP technologies and their access paths. The focus of this paper is on OLAP in distributed scenario, where we pinned on the drawback of OLAPs natural indexing search. We designed a new translated lattice called the pchrome lattice, whose nodes are binary. We implemented the natural indexing on this translated lattice and showed a drastic reduce in indexing search space, search time and distributed communication cost.

Keywords- MOLAP, ROLAP, HOLAP, B-tree, Bitmap, R-trees, R*-trees, R-cube.

I. INTRODUCTION

In the past decades we have been using various database technologies to answer many of user queries either simple or complex. The prominent use of the database technology is seen in business enterprise where decision making is prior than transactions. Traditional database systems are transactional processing systems, which can access only few tuples for database reads and writes [1]. Their major drawback is they cannot handle the user decision making queries. This is because decision making is an instant comparison of past data and present data and traditional databases does not store any past data. To handle enormous past and present data and to support decision making queries many of the enterprises are using an extended database technology called data warehouse. Data warehouses differ very much from the traditional database applications. Data warehouses are mainly used by major business enterprises, to analysis their business trends and to track their business profits. Analysts use the data warehouse to extract the business information that enables better decision making. This type of interactive decision making process is provided by OLAP (On-line Analytical Processing) tools [2]. These OLAP applications mostly use only data reads for their decision making. Real time complex analytical queries are answered using OLAP.

The most commonly used OLAP technologies are Multidimensional On-line Analytical Processing (MOLAP), Relational on-line Analytical Processing (ROLAP) and hybrid on-line Analytical Processing (HOLAP) [3]. They are different in their data processing capabilities. They have their own supporting data accessing methodologies. Though they are opposing technologies they are widely recognized by many of the today’s decision making enterprises.

The remaining parts of the paper are organized as follows: In section 2 we briefly discuss the classification of OLAP technologies. In section 3 we discuss the data accessing methods. In section 4 we have discussed when and where to use these technologies. In section 5 we have discussed about OLAP in distributed scenario. Finally section 6 concludes the paper.

II. OLAP TECHNOLOGIES

Organizations huge data is a critical resource which is in need of powerful tools to fetch queried information. OLAP is one such powerful technology providing sophisticated tools for an enterprise to meet its competitive goal. Currently there are three dominant OLAP technologies:

- Multidimensional OLAP (MOLAP).
- Relational OLAP (ROLAP).
- Hybrid OLAP (HOLAP).

A. MOLAP

In MOLAP the preprocessed data is aggregated and uploaded periodically in a multidimensional array structure called Data cube [4]. Basing on the dimensional hierarchies the data cube is divided into sub-cubes. For a data cube with n dimensions without hierarchies there can be a total of 2^n sub cubes. With hierarchies defined the number of sub cubes increases. As the dimensions and dimensional hierarchies increase the cube becomes larger with many sub-cubes. As such a molap query for a user requested sub-cube has to spend time for an on fly analysis. To make this on fly analysis faster what followed by molap is pre-computation. Pre-computation is a generic support for short response times where some of the sub-cubes are materialized [5]. Materialization is way where some of the needed measures like sum, average are calculated pre hand and the values are stored in the sub-cubes. In molap all these measures are stored in arrays, referenced by dimensional names that are strings. Between the warehouse and the user end tools a Molap cube sits analyzing the user requested data. For a Molap cube with huge dimensional hierarchies many of the smaller granules of the cube will be left pre-computed. This is what is the dimensional cursity[6] of the data cube, where many sparse sub-cubes are generated. The main problem with sparsity is many of the olap methodologies will search through the sparse cube to identify whether the user requested sub-cube is materialized or not. This may increase the query waiting time. Research has provided with many methodologies on which sub-cubes to materialize [7]. To our knowledge there is less work done on
what sub-cubes are materialized. Molap has its own born advantage with its natural array structure, which is flexible for many of the olap accessing methods and analysis on present and past data can be easily done. The outer cube layer contains the present recent data and the inner sub-cubes contain the past data. MOLAP uses many operations to perform an on fly analysis. All the queries will be directly posed on the molap array based lattice structure shown in figure 5. Using a string matching technique the requested view can be fastly retrieved. Even the molap structure supports easy aggregation of data along multiple dimensions.

**B. ROLAP**

In rolap the warehouse data is stored in relational or extended-relational database. Rolap uses tables to store the past and the present data [8]. There is a greater scalability with rolap server for large data sets. Between the data warehouse and the client front end tools the rolap server is used which is a collection of multiple tables. The problem of sparsity does not arise here because tables can be joined to return the user query if needed with the multiple group bys. In rolap pre-computed data is not stored in advance. The aggregates from multiple tables are calculated on fly. Considering rolap, user requested aggregates may be in multiple tables. Here the rolap server follows a translation of user query to a multi-statement SQL (Structured query language) query posed on multiple tables. On fly analysis from multiple tables may take much time and this is the main drawback of rolap.

**C. HOLAP**

HOLAP combines the features of both MOLAP and ROLAP. It supports both of MOLAP’s multidimensional structure and ROLAP’s sql constructs. It uses the features of MOLAP to address fast processing of user queries and uses the features of ROLAP to address the processing of large data

Holap stores the most recent data in molap to enhance faster access and stores past data in rolap. HOLAP addresses a complex query by dividing it into sub-queries [9]. The sub-queries which span dense data sets are directed to MOLAP and the sub-queries which span sparse data sets are directed to ROLAP.

**D. OLAP OPERATIONS AND OPERATORS**

To perform the multidimensional analysis on fly and for faster query responses OLAP includes the following basic operations:

- **Roll-Up**: Otherwise called as aggregation where data from low levels to high levels is aggregated to provide a summarization at the high levels.
- **Drill-down**: Allows data navigation from higher level to lower level data.
- **Slicing**: Describes the selection of data along single dimension of which the view is a table.
- **Dicing**: Describes a selection of data along multiple dimensions whose view is again a sub cube.

Using the above operations olap will present user requested multidimensional analysis. Using Roll-up operations sub-totals can be aggregated to grand totals, using drill-down the application can navigate from grand total to sub-totals. Using dicing operation a sub-cube can be selected. Using slicing a cross section of the cube is selected i.e. a table can be selected.

To perform the above operations OLAP uses two types of operators:

- The group-by.
- The cube-by.

**TABLE 1. SALES DATA**

<table>
<thead>
<tr>
<th>Parts</th>
<th>Locations</th>
<th>Time</th>
<th>Sales quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>4</td>
</tr>
<tr>
<td>A1</td>
<td>B2</td>
<td>C1</td>
<td>2</td>
</tr>
<tr>
<td>A2</td>
<td>B3</td>
<td>C1</td>
<td>2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C2</td>
<td>4</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>1</td>
</tr>
</tbody>
</table>

**1. GROUP-BY OPERATOR**

The group-by is a usual relational operator. It typically operates by partitioning the relation into disjoint tuples and then aggregating along the given dimension. Group by operator does not allow a direct aggregation along a column i.e. rolling of sub-totals to a grand total or drilling down of grand total to sub-totals [10]. For instance consider the sales data as shown in table 1, where location B1 has dimensional hierarchy’s b1, b2.

A query of the type

SQL Query 1:

Select A,B,C, SUM(quantity)
Query 1 shows grouped data based on the dimensions B, A. It shows total quantity of items grouped by the dimension B with reference to the dimension A, but does not show sub-totals on single dimension B or on single dimension A i.e. it shows quantity of distinct values in B or A i.e. B1=4, B2=2, B3=7 individually but cannot drill-down to B1 and give sub-totals on hierarchies in B1 i.e. cannot give b1=2,b2=2. These types of sub-totals may fetch the decision makers that cannot be provided by group-by operator.

2. The problem with group-by operator

The main problem with group-by operator is it will not provide multidimensional group-bys, even it cannot analyze the aggregates of dimensional hierarchies. The group-by operator cannot support drill down and rollup operations which are meant for multidimensional analysis. Queries of the type: Display the total sales from the sub location b1 cannot be answered. In order to provide a multidimensional aggregative analysis the application has to perform the union of multiple SQL statements as follows:

```
SQL Query 2:
Select A,B,C, Sum(quantity) from sales
Group-by A,B,C;
UNION
Select A,B,C, Sum(quantity) from sales
Group-by A,B;
UNION
Select A,B,C, Sum(quantity) from sales
Group-by A;
UNION
Select A,B,C, Sum(quantity) from sales
Group-by B;
UNION
Select A,B,C, Sum(quantity) from sales
Group-by C;
UNION
Select A,B,C, Sum(quantity) from sales
Group-by ();
```

Even such type of union of SQL group-bys cannot address analysis on dimensional hierarchies and this is the failure case of group-by operator.

3. The cube operator

The cube operator is designed to address the drawback of SQL group-by. The union of multi SQL statements given in SQL query 2 can be replaced with a single cube-by operator as follows:

```
SQL query 3:
Select A,B,C, Sum(quantity) from sales
Cube -by A, B, C;
```

The above cube-by query calculates aggregates along all possible dimensional combinations. Since there are 3 dimensions the cube-by operator calculates $2^3=8$ possible aggregates ABC, AB,AC,BC,A,B,C ,ALL and thus addressing the failed case of group-by operator. The cube-by operator along with where conditions can provide aggregates on dimensional hierarchies.

4. Drawback of Cube-by operator

The major drawback with cube-by operator is it more expensive [11]. More expensive in the sense the cost of usage is high. For roll-up or drill-down operations the cube by operator has to carry with huge lots of string cube by dimensions. This may increase the operator file size. In SQL query 3 the cube by operator has to carry with all 3 dimensions A,B,C that are characters and thereby the cube operator occupies three bytes of search space. But in reality these dimensions are strings which when included with cube-by operator may considerably increase the search space. Carrying all these huge string data from outer layer of the olap structure to inner layer may still increase the search space. This has made the cube operator to be less choosable though it is advantageous over group-by operator. In the context of this survey we state a research problem:

**Problem 1 description:** Can there be a better transformation of the cube-by string dimensions to somewhat like binaries which exactly identifies the requested views and thus can reduce the search space of the cube-by operator.

III. OLAP ACCESSING METHODS

To increase the performance of olap, research has provided many accessing methods. In this paper we present our survey on two major accessing methodologies:

- OLAP Pre-computation.
- OLAP Indexing.

A. OLAP Pre-computation

Pre-computation is a way where parts of the whole cube are materialized to provide fast query responses. The advent of warehouse technology has led to many sophisticated pre-computing methods. Parts of data cube that are pre-computed can also be termed as materialized views [12]. With pre-computation the question is: which views to be materialized, and how to optimally materialize. Choosing which views to materialize is based on three important factors: the query cost, view materialization cost and storage space. Sub-cubes requested by common queries are given priority for materialization. Sub-cubes most frequently requested for are pre-computed.
There are many approaches on how pre-computation is performed:

1. Multi-way
Multi-way array aggregation discussed in [13] pre-computes the aggregates using array as its basic structure and is a full cube computation method. It makes use of chunk concept where the entire cube memory is partitioned into chunks. These chunks are then simultaneously aggregated across multiple dimensions to pre-compute various sub-cubes. The multi-way array aggregation is faster as it is done on the molap structure using a direct array addressing.

Figure 4 shows the multi-way array aggregation where ABCD is a base cuboid. Memory chunking is done to fit ABCD and from ABCD cuboids ABC,AB,A etc., can be calculated allowing multiple aggregations across various dimensions.

![Figure 4. Multi-way](image)

For instance with an initial sort of ABCD, the prefix group-bys ABC,AB,A can be pre-computed without actually sorting them, thus reducing additional sorts.

4. Hashing
The hash based method is based on optimizations of cache results and scans [16]. Usual pre-computation methods incur multiple scans of the dimensional attributes which is costly. For instance in one scan the aggregate ABC is pre-computed. To compute AB, again we have to scan AB once, thus taking two scans of the same attributes.

Instead the hash based method caches the result to further reduce the scans. For example the hash based method maintains hash tables in memory where AB and AC can fit. Now in one scan of ABC both AB, AC can be precomputed.

5. H-Cubing
A better cube computation is offered by H-cubing discussed in [17]. H-cubing computes on a tree like data structure called the H-tree. From the lattice structure shown in Figure 5 H-cubing constructs H-tree from which it computes the multidimensional aggregates. The advantage of H-cubing is being in one level the method calculates the possible aggregates within the same level before proceeding to the next higher level.

6. Star Cubing
Star cubing discussed in [18] combines the features of multi-way, BUC and H-cubing. It combines both top-down and bottom-up computation approaches. From the lattice structure shown in Figure 5 it constructs a star tree and identifies the star nodes as the nodes not satisfying iceberg conditions and prunes them. The advantage of star cubing is being in one level the method even computes the aggregates of both lower and higher next levels using shared dimensions and simultaneously prunes the aggregates not satisfying iceberg conditions.

B. OLAP Indexing
To support fast accessing to multidimensional aggregates the olap systems follow indexing. Many existing indexing methods are followed by both molap and rolap. We examine each in the context of our survey.

1. Natural Indexing
Natural indexing also called array based indexing is supported by MOLAP. The array structure of the MOLAP itself forms the natural indexing. Natural indexing is the only indexing method which is done on the storage layer of the data cube [19]. As the indexing is done on the storage layer it is faster and the requested views are retrieved in no time. The lattice structure of the cube in Figure 5 presents arrays as layered sub cubes at various levels. The end points of each layer represents one possible group-by of the dimensions which are usually strings. In natural indexing all these string nodes are stored in string arrays. Whenever user queries for
an aggregate ABC, these aggregates are directly mapped on to the end points of the layers using natural indexing and the corresponding sub cube which is highlighted in Figure 5 is retrieved. Thus MOLAP’s natural indexing offers improved performance by directly indexing on the structure of MOLAP. But many enterprises prefer other indexing techniques as it doesn’t suit for large data sets. The major drawbacks with data cube natural indexing are:

- When number of dimensions is more the cube becomes sparser [20], that means several cells that represent particular attribute combinations will not contain any aggregated data. There by the natural indexing search for which sub-cubes are pre-computed becomes time consuming.
- The natural indexing search directly uses the user group-by dimensions, which is string data. This type of search using string data even increases storage space for large dimensions.

For instance consider the sql query

SQL query 4:
Select * from sales
Cube by A,B,C;

Here the molap indexing directly indexes the outer view ABC which is directly fetched from the string array using string comparison technique and the view highlighted in figure 5 is retrieved. This type of string comparison may take huge index file size and high comparison time and even the retrieval time is bit increased.

To reduce the index file size and high string comparison time the MOLAP based model IBM Cognos8 uses a transformer module where user queries are transformed to BEx queries (Business Explorer queries) [21]. In the BEx queries the user requested string group-bys are represented using BEx variables that are also long strings. For example in the above query the group-by ABC can be assigned a BEx variable of the type SAPBWOODPP2. Then using the cognos locate option the BEx variables are matched.

In the context of our survey here we project the drawback of BEx queries: BEx variables are also long strings and using string comparisons the search index file size may be large and there by the search time too long. We want to go with a method where in the transformed query the user requested string group-bys are transformed to unique binaries; there by the search index file size and search time is reduced.

2. Tree Based Indexing And Variations

Both MOLAP and ROLAP supports tree based indexing methods. One of the traditional tree based indexing is the B-tree [22]. A B–tree indexing includes sub trees corresponding to each dimension of the data cube. As the values of the cube dimensions are unique, B-tree uses these dimensions as index pointers that point to the sub trees. By tracing the pointers, data can be easily retrieved. For an 8 bytes column the B-tree index file size is 326 MB and the construction time is 1580s. To build index on a large column B-tree is expensive in terms of space and construction time. The main drawback with B-tree indexes are rebalancing the tree is needed with updates. Other popular tree based indexing structures that are supported by both MOLAP and ROLAP technologies are R-trees [23], aR-trees [24]. The R-tree indexing supports complex range queries to some extent. Much research was done on R*-trees to extend them into structures like Ra’-trees [25], Hilbert R-trees. All of these uses more sophisticated update algorithms; they can answer complex range queries; they can dynamically rebalance the tree structure whenever updates are performed. The major drawbacks of tree based indexing are:

- Huge storage.
- Supports only few dimensions.

3. Bitmap Indexing And Variations

Bitmap indexing was introduced to enhance the performance on various query types [26]. For each attribute of the table one bitmap index is associated. Each row of the bitmap vector is given a row-id starting from 0. Rows will have distinct attribute values. The basic idea behind bitmap indexes is to use a string of binary numbers to indicate whether the indexed attribute in a table is equal to a specific value or not. If the bit is set to 1, it indicates that the row with the corresponding row-id contains the key value; otherwise the bit is set to 0. Complex queries on one or more dimensions can be answered by intersecting the bit maps over multiple dimensions and also by using AND/OR operations. The major advantages of bitmaps are:

- Overcomes the storage limitation of B-trees.
- Bitmaps are retrieval efficient for low cardinalities.
- Sparse data can be efficiently handled.
- More CPU efficient because of their simple representation.

The major disadvantages of bitmaps are:

- Efficiency decreases for high cardinalities.
- AND/OR operations are expensive.
• As the dimensions increases more bitmap vectors are needed; results in overhead of storage space.
• Cannot support huge reads(updates).

To address this storage overhead encoded bit maps [27], hybrid bitmap methods are introduced [28]. The encoded bitmap indexing can be used for large cardinalities. The basic idea of encoded bitmap indexing is to encode the attribute domain. There by we can reduce the number of bit vectors and thus reduce the storage space.

Other variations of bitmaps: projection bitmap, bit-sliced indexes are discussed in [29].

4. OLAP Join Indexes

Most popular olap index is the join index [30]. All the traditional indexing methods discussed above indexes by mapping a column value to a group of rows having that value with in a same relation. Further every relation need to be indexed separately and if needed with tuples from two are more tables traditional indexing methods increase the cost of joins. This may increase the index size with increased relations. In contrast the join index provides a grouped index on two or more tables. They contain indexed records that contain joinable rows of relations. Thus making it easy to identify the joinable tuples and without going for further costly joins.

IV. CHOICE BETWEEN OLAP TECHNOLOGIES

Not all real word enterprises are using either strictly MOLAP or strictly ROLAP or HOLAP. Their choice is varying according to the potential benefits of OLAP like which improves decision making, which provides accurate analysis, provides all user required information, that improves working efficiency, and that increases user productivity.

A. Choice Between MOLAP / ROLAP / HOLAP

• MOLAP best suits non sophisticated users as it uses user friendly graphical visualization techniques; whereas use ROLAP for sophisticated users [31].
• If the users are needed with consistent information for a period of time MOLAP is preferred. If the requirement changes frequently ROLAP should be used because of its flexible query capability.
• For decision making on past data MOLAP should be adopted. Whereas for decision making on current data ROLAP is adopted.
• Because of the easiness of MOLAP, it is recommended at the beginning of an enterprise. After considerable decision making experience, a ROLAP system is preferred because of its flexibility and ability to handle complex queries.

Table 2 shows various OLAP technologies their features and the enterprises implementing them.

<table>
<thead>
<tr>
<th>OLAP Technology</th>
<th>Features</th>
<th>Adopted by</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOLAP</td>
<td>High performance, less scalable for huge dimensions</td>
<td>Microsoft SQL server 2005, Essbase server from hyperion</td>
</tr>
<tr>
<td>ROLAP</td>
<td>Low performance, More scalable for huge dimensions</td>
<td>Microsoft SQL server 2005, Micro strategy’s Dss server, Informix meta cube</td>
</tr>
<tr>
<td>HOLAP</td>
<td>High performance and scalable for huge dimensions</td>
<td>SAS server</td>
</tr>
</tbody>
</table>

B. Choice between OLAP Indexes

The main problem with OLAP indexing is still today there is no definite guideline for an analyst to choose best suited indexing method:

• For a DW system with very few dimensions a MOLAP natural indexing is advisable. For an enterprise which is at it’s begin set a MOLAP natural indexing can be adopted.
• For a DW system which is frequently updated tree indexing is advisable because tree indexing does not need rebalancing. For an enterprise using only few dimensions B-trees are advisable. To answer complex range queries the variation of tree indexing, R*-trees are advisable [32].
• Bitmap index is best suited for columns having less number of distinct values. Bit map indexing supports more number of dimensions than B-trees. For a DW system which is not frequently updated a bitmap indexing is advisable.

Table 3 shows various OLAP indexing techniques their features and the enterprises implementing them.

<table>
<thead>
<tr>
<th>OLAP Indexing</th>
<th>Features</th>
<th>Adopted by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural indexing</td>
<td>Easy and fast retrieval, supports few dimensions</td>
<td>Microsoft corporation, IBMs Cognos</td>
</tr>
<tr>
<td>Tree indexing</td>
<td>Supports huge dimensions, huge storage space, and more retrieval time when compared to natural and bit map.</td>
<td>IBM SPSS, Oracle, Red Brick</td>
</tr>
<tr>
<td>Bit Map indexing</td>
<td>Easy, supports huge dimensions, less storage when compared to tree index</td>
<td>Inter system Corporation, Oracle, Red Brick, DB2</td>
</tr>
</tbody>
</table>
C. Indexing Performance Study

Here we study the performance of various olap indexing methods by comparing the index file size and search time on varied dimensions. For example consider a warehouse sales on three dimensions A,B,C. suppose each dimension is included with 10000 tuples: consider the query SQL query 5:

```sql
Select count(*) from sales
Where A='X'
Group by A,B,C;
```

The index file size in units of memory bytes, the index construction time and the query retrieval time in units of seconds is compared as shown in Table 4.

### TABLE 4. INDEX COMPARISON

<table>
<thead>
<tr>
<th>Index Type</th>
<th>File Size (bytes)</th>
<th>Construction Time(µs)</th>
<th>Retrieval Time(µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>10</td>
<td>1200</td>
<td>10</td>
</tr>
<tr>
<td>Btree</td>
<td>10</td>
<td>1000</td>
<td>40</td>
</tr>
<tr>
<td>Bitmap</td>
<td>6</td>
<td>800</td>
<td>20</td>
</tr>
</tbody>
</table>

Bitmap index requires less storage space than B-tree as shown in Table 4. As the number of dimensions increases the B-tree indexing fails to scale to the increased file size. Figure 6 shows the search time comparisons of various indexing methods. The natural indexing presents fast search for less number of dimensions, but as the dimensions increase the search time also increases. This is because the natural indexing search is carried out with huge dimensional groupbys that are string data and searching with string data takes comparatively more time. The B-tree and Bitmap search scales well even for high dimensions.

V. OLAP IN DISTRIBUTED SCENARIO

Usually data warehouses contain real world’s huge amounts of data that must be analyzed. Most of to-days OLAP applications work on data warehouse with a centralized structure in which a single database contains huge amounts of data. As data warehouse tend to be extremely large, the centralized data warehouse is very expensive. This has led to the distributed scenario where the large data warehouse can be partitioned and distributed across various locations. All the OLAP technologies scale considerably well in the distributed scenario.

A. Distributed MOLAP

Many current distributed OLAP systems use MOLAP approach. The main reason is its fast execution of OLAP queries. But MOLAP does not scale well in case of more number of dimensions. But many of the distributed enterprises are following a vertical fragmentation of the dimensions, which allows few dimensions to be at each location and then going for a join of these fragments for user queries. The scenario even supports central access methods like natural indexing, B-tree indexing and bitmap indexing.

B. Distributed ROLAP

Though many vendors are sacrificing scalability for performance; to support current huge data warehouse enterprises are adopting scalable ROLAP. Indexing by R-trees and R* trees are still supported. Many of the to-days ROLAP enterprises are with a novel distributed indexing called the RCUBE indexing [33]. RCUBE indexing is fast and is a combination of packed R-trees with distributed stripping and Hilbert curve based data ordering.

The supporting features of RCUBE are:
- Low communication volume.
- Scalable in terms of data sizes and dimensions.

C. Distributed HOLAP

Many of current distributed systems use HOLAP. The distributed approach of MOLAP stores the frequently requested sub-cubes in MDDB (Multi dimensional data base) and less popular parts on a remote RDB (Relational Database) [34]. In addition to its flexibility to larger data sets, distributed HOLAP provides other features like:
- Caching - The distributed HOLAP saves the results of a query so that it can be reused later.
- Logging- The distributed HOLAP creates a log file where the information about each query is stored.

D. Distributed OLAP Querying

The distributed nature of OLAP architecture results in the following costs [35]:
- Query processing cost.
- Communication cost.
E. Query Processing Cost

Early olaps are criticized for being inefficient in handling complex queries with huge operations. The efficiency of distributed olap is measured in the way the query is optimized. Query optimization is a way in which complex queries are transformed to include less cost operations. Many of the distributed olaps follow query optimization to reduce the processing cost by using transformation mechanisms in the search space. A desirable optimization is the one which incurs less cost by reducing the search space and minimize the response time.

The distributed olap optimizations include many transformation techniques where the original query is translated to some sort of algebraic expressions which represent the original query. Evaluation of user query using these algebraic expressions is of less cost when compared to evaluation of the original query. Many of the distributed query evaluation techniques are successful in minimizing the search time but failed to reduce the search space.

The SKALLA system discussed in [36] uses Multidimensional Join (MDJ) and Generalized MDJ (GMDJ) operators for expressing olap queries. The GMDJ operator optimizes the complex olap queries by separating the aggregate functions, the definitions and the dimensions from the complex query into operator notations that are of less cost. While separating the query into GMDJ expressions the SKALLA system still includes the string dimensions of the cube by query. These types of GMDJ expressions with string dimensions may decrease the query response time but the search space which includes the string dimensions may still be large.

For instance let A, B be two table, f1,f2…fn be the list of aggregate functions, a1,a2…an and b1,b2…..bn be the dimensional attributes of A,B. The GMDJ expression is also a relation of the type (f1Aa1, f2Aa2….,fnAan, f1Bb1….fnBbn). Since the search is done using these GMDJ expressions that include the dimensions and dimensional hierarchies Aa1, Aa2, ..Bb1, Bb2…. that are strings, increases the search space.

F. Communication Cost

A decreased communication cost increases the efficiency of the OLAP technology.

For example consider the SQL query:

SQL query 6:

Select A, B, C, sum(s)
Cube by A, B, C,D;

The query is a request for the group-by with 4 attributes A,B,C,D that are strings. This group-by represents a materialized view of the whole MOLAP cube shown in figure 5. In a distributed scenario if this view is not present at a node, then the query has to be redirected to other nodes. We have surveyed that in distributed scenario OLAP redirects the query in a translated form which includes the group by attributes that are strings. Communicating string group-bys to more than one node may increase the communication cost.

The GMDJ relations discussed above are used to count the number of query redirects to various distributed nodes. While redirecting these GMDJ relations, the SKALLA system includes reduced base table of the original query thus reducing the communication cost. But the reduced base relations still includes with the string dimensions which may increase the communication cost. We are now with a problem of:

Problem 2 description: Can there be a better translation of group-bys from string data to somewhat like binary, there by communicating binaries instead of strings decrease the communication cost. We started our work by combining these two problems and planning to publish as our future extension to distributed OLAP technologies. Using this translation of cube-by dimensions to binaries we can address the problem of cube-by operator and can make molap less costly thus can make the molap technology to be adopted by all.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we discussed about prominent OLAP technologies and their accessing methods. Though MOLAP and ROLAP are different in features both are considerably extending their services to real time decision making. At the beginning many enterprises are adopting MOLAP then after a better acquaintance with the usage they are switching to ROLAP. We provided a survey on various standard olap accessing methods and their disadvantages. In the context of our survey we projected on MOLAP’s advantage of fast search and retrieval. Because of molaps dimensional curiosity problem many enterprises are moving to rolap Even the distributed olap suffered from increasing the communicating cost. Research in data cube technology is still arriving with new indexing methods. Some of these methods are still under our study and we will project them in our future work.

We are working towards the problem of making MOLAP technology to be used in such a way to reduce sparsity and render fast search; and also communicating the group-bys in the distributed molap cube architecture so as to reduce the communication cost.

As a future enhancement we want to map molap lattice structure to a compressed lattice structure whose nodes are binaries rather than strings. We want to go with a query transformation mechanism which translates the string cube-by dimensions to the binaries which exactly represent the lattice nodes. Thus the search can be carried on the compressed lattice with binaries there by reducing the query retrieval time as well as search space. In the distributed olap architecture if the cube-by view is not present at a location then instead of communicating the string cube-by dimensions our method communicates binaries that are empirically same as the requested view and thereby reducing the communication cost.
REFERENCES


[21] IBM Cookbook for IBM Cognos 8.4 for use with SAP Netweaver business warehouse.


ABOUT THE AUTHORS

Dr. K. Dhanaasree received her Ph.D degree in Computer Science and Engineering from JNTU, Anantapur. Her research interests include Data Mining, Database Security, and Network Security.

Dr. C .Shoba Bindu, received her Ph.D degree in Computer Science and Engineering from JNTU, Anantapur. She is now extending her services as Professor, Dept. of CSE, JNTUA. She has guided many external and internal projects and has good contributions in many of reputed journals. Her research interests are Mobile and Adhoc Networks, Network Security, Data Mining and Cloud Computing.