Classification of Sparsity Patterns and Performance Evaluation in OLAP Systems

Juyoung Kang∗+, Hwanseung Yong*, and Yoshifumi Masunaga+
Department of Computer Science and Engineering, EIST
Ewha Womans University*, Korea
Department of Information Science, Faculty of Science
Ochanomizu University+, Japan

Abstract

In fact, input data of OLAP is usually very sparse and it causes data explosion phenomenon in precomputation of summary tables. With very sparse data, space for all possible cells defined as combinations of dimension members in multi-dimensional cube has to be assigned although they have no actual data. Therefore, this takes a huge amount of space, regardless of the storage efficiency or the database technology used and decreases response time of multi-dimensional queries [7,8]. Several approaches have been presented for handling sparsity of OLAP like chunk-based array structure [9,10], composite dimension method [11] and sparse-dense split method [12]. Although these techniques can handle the sparse data with a fixed pattern, they have no general solution to the mixed type of sparsity patterns. However, actual data in applications of OLAP represents complex sparsity patterns so more generalized approaches have to be considered.

OLAP performs the applications that interact with users and require fast response time over large amounts of data of data warehouse. Because these applications need a lot of multi-dimensional aggregation, it can take many hours or even days to run if executed directly on the raw data. Therefore, various kinds of precomputation techniques are proposed to handle this performance problem. However, sparse data of OLAP causes the data explosion problem in precomputation process and decreases the performance of OLAP. In this paper, we discuss the sparsity of OLAP data and the performance bottleneck problem. As a fundamental research on sparsity handling problem, we define the sparsity patterns and develop an automated data generation program according to the sparsity patterns. We evaluate the performance of two OLAP products, MS SQL Server 2000 Analysis Services and Pilot DSS (Decision Support Suite) with the generated data. Focusing on the sparsity handling ability, evaluation results also are presented.

1. Introduction

OLAP (On-Line Analytical Processing) provides analytical insight into enterprise data to knowledge workers through fast, consistent, interactive access to a wide variety of possible views of information [1,2]. In general, these OLAP systems need to support the complex analysis requirements of decision-makers, analyze the data from a number of different perspectives (business dimensions), and support complex analysis against large input data sets. Because of these requirements, OLAP queries are complex than those of OLTP and can take many hours or even days to run if executed directly on the raw data. This performance bottleneck comes from the fact that these queries need a lot of multi-dimensional aggregation [3,4]. The most common and powerful method of reducing execution time is to precompute some of the queries into summary tables and then to build indices on these summary tables [5,6]. However, precomputation of aggregates in the presence can result in extremely large increase for storage required by the database.

2. Related Works

Generally, the actual data of OLAP is very sparse and this sparsity is the critical factor of performance bottleneck of OLAP systems. Several approaches to this sparsity problem have been proposed.

In case of ROLAP (Relational OLAP), only cells that have actual data, i.e. non-sparse cells, are stored in database as tuples. However, in case of MOLAP (Multidimensional OLAP) data is stored in the multi-dimensional array, so sparse cells should be handled in efficient way [9]. As a approach to handling the sparse data based on multi-dimensional array structure, array can be split up into “chunks” as
suggested by Sarawagi [13]. When multi-dimensional arrays are stored as chunks, each chunk can be either stored as a dense or a sparse structure. Sparse storage by chunk compression has been used in [9]. Chunks are compressed if the data density is below a certain threshold otherwise they are stored using dense array representation. Several sparse data structures, like index-value pairs, offset-value pairs or compressed sparse dimensions extended from CSR (Compressed Sparse Row), can be applied to storing sparse chunks as shown in figure 1.

![Sparse data structure](image)

Figure 1: Storage for sparse data based on chunk structure

Another approach has been proposed in [11] and this is used in a MOLAP product, Essbase of Hyperion. In this method, distribution characteristic of cells is considered, and then dimensions are split into dense and sparse dimensions. The dense dimensions are stored as a multi-dimensional array blocks. The sparse dimensions act as an index into the dense blocks. Two methods, block pointer array structure and binary tree index structure, are applied to store the sparse dimensions. In case of the former array structure, each valid combination of the sparse dimension has a pointer to its data in the dense array, while each node of a binary tree has pointers to the corresponding dense block in case of the latter.

As an alternate sparse data structure, composite dimension method proposed in [12] is used in Oracle Express. They assume that sparsity occurs from relationship between specific set of items in one dimension with other dimensions in practical applications. An example of this case is illustrated in Figure 2(a). In this method, a composite dimension that is made up of two or more base dimensions is composed. The basic idea of a composite dimension is that it gives a way to group most of the dimensions in a variable’s definition in order to manage sparsity. Therefore, with this storage structure, storing only related data can save storage space.

In [10], bit-encoded sparse structure (BESS), which maintains indices for a mapping of each dimension in bit encodings, is represented. By using chunking and bit-encoded index structure, the storage requirement is improved noticeably.

Although these techniques can handle the sparse data well, they have assumptions on the sparsity type of data. Therefore, more generalized sparsity control method should be considered for controlling actual data of OLAP that shows more complex sparsity type.

3. Sparsity Patterns of OLAP Data

Sparsity refers to the degree to which cells contain invalid values instead of data [2,14]. For example, if a variable is 25 percent sparse, that means that 25 percent of that variable's cells contain invalid value and 75 percent of that variable's cells contain data. Sparsity in multi-dimensional model never be existed independently but it occurs involved with relationship between dimensions. For example, sales data have to be treated as sparse when some products are not sold in some stores. In this case, there are no sales of specified product in stores, which do not handle that product from the first. Actually, because of a nature of business related with OLAP application, these cases are very often [2,7].

We considered how these sparsity characteristics can be presented more formally and defined the patterns of sparsity in OLAP data in 2 and 3 multi-dimensional models. Suppose that sales analysis model includes 2-dimensions named Product and Store as shown in Figure 3(a). We can have sales by product and store like Figure 3(b) when invalid cells are scattered throughout a variable, usually because some combinations of dimension values never have any data. This type of sparsity can be called ‘random’ pattern. In other case, when store s2 and s4 are closed on that day or when no household electronic appliances are dealt with at those stores, sales data will show the sparsity pattern as illustrated in Figure 3(c). This is ‘stripe’ type of sparsity pattern. As shown in Figure 3(d), ‘cluster’ pattern can be possible. Like this, sparsity patterns in 2-dimensional
model can be defined as three types, which are random, stripe and cluster.

Next, consider 3-dimensional model, in which we have the dimensions Product, Store, and Time as shown in Figure 4(a). Similar to the 2-dimensional model, we can classify the sparsity patterns into four types, random, stripe, cluster and slice as illustrated in Figure 4(b), 4(c), 4(d) and 4(e). In addition to the same three types of 2-dimensional model, ‘Slice’ pattern can be possible in the 3-dimensional model when, for example, all the products were sold steadily from January to April at store S1 while there have been no sale at other stores.

These sparsity patterns have interrelationship in the aspect that one sparsity pattern can be converted into another one by coordinating the dimension members. Moreover, this characteristic is important in that existing methods, which handle only a restrictive pattern, can offer a basic concept for controlling another pattern of sparsity.

For evaluating OLAP systems with defined sparsity patterns, we designed an example multi-dimensional model and implemented automated test data generator based on the designed data model. Although dimension in OLAP applications is generally higher than five, our tasks are concentrated on 2 and 3-dimensional patterns mentioned above. 2-dimensional model is composed with two dimensions, Store and Time, and two measure values, which are Unit Sold and Sales as illustrated in Fig 5(a). Store dimension has a hierarchy which has two levels: Store and Retailer, and Time dimension has 4-levels hierarchy: Year, Quarter, Month and Date. Since the space for fully aggregated data can extremely increase, we set up the number of members to 9600 for store dimension and 730 for time dimension each. Detail information about the 2-dimensional model is described in Table 1.

4. Sparse Data Generator

We developed automated test data generator for
Table 2: Detail information of the 3-dimensional model

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Store, Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of members of each dimension</td>
<td>Store: 500, Time: 365, Product: 120</td>
</tr>
<tr>
<td>Hierarchy of each dimension</td>
<td>Store: Store ⇒ Retail, Time: Year ⇒ Quarter ⇒ Month ⇒ Date, Product: Division ⇒ Group ⇒ Product Code</td>
</tr>
<tr>
<td>Number of data</td>
<td>21900000 (120×500×365)</td>
</tr>
<tr>
<td>Sparsity</td>
<td>5%</td>
</tr>
</tbody>
</table>

evaluation of current OLAP systems. This program generates sparse data based on multi-dimensional model and sparsity patterns defined in previous section. Users can choose the format of data in either in text file format or MS Access data format. Fact table and dimension table are generated into separated data file for the convenience of loading to OLAP systems. Generated data size is about 9.5MB in flat text file format in case of the 2-dimensional data and about 33MB in case of the 3-dimensional model. Interface of developed program is shown in Figure 6.

To confirming the patterns of generated data, we used 3D Cube Explore of DBMiner 2.0. This function makes it possible for users to visualize the multi-dimensional data in 3D cube shape and to perform the OLAP queries like drill-down or roll-up easily. Visualized shapes of 3-dimensional data is shown in Figure 7. As you see in each sub-figure, generated data show their sparsity characteristics well.

5. Experimental Results

This section describes evaluation programs for two commercial OLAP systems, MS SQL Server 2000 Analysis Services and Pilot DSS. We design an example query model for performance evaluation. The queries are based on the general seven queries of OLAP applications. The basis of OLAP query is to define the part which user is interested in. Most of OLAP queries perform the operations that summarize or consolidate data or give a view of specific part of data. The queries fall under the following categories [2,10,15].

1. Exact Match
   Retrieve one cell that is a combination of specific members from each relevant dimension.
2. Range
   Retrieve the values within specified ranges in any \( m \) of all \( k \) dimensions. \((m \leq k)\)
3. Slice
   When a value of member in one dimension is specified, retrieve all values over all unspecified dimensions.
4. Dice
   Retrieve the values for a sub-cube consisted with the combinations of specified members in one or more dimensions.
5. Pivot
   Rotate the cube to fit to the new point of view.
6. Drill Down
   Retrieve values, which are more specific, along the descending hierarchy in one or more dimensions.
7. Roll Up
   Aggregate along the ascending hierarchy to get values at higher level in the hierarchy.

The visualization of the operations of these queries illustrated in Figure 8 can give us explanation that is more intuitive. Figure 8 shows 7 OLAP queries in the 3-dimensional model.

In this paper, we develop performance evaluation program for two commercial OLAP systems as mentioned earlier. Using the test data generated by the data generator also mentioned earlier, we evaluate
the performance by measuring the execution time of seven OLAP queries.

The program for MS SQL Server 2000 Analysis Services, DSO (Decision Support Object) library [16], ADO MD (ActiveX Data Objects Multidimensional) API and MDX (Multidimensional Expression) [17] were used to access the data cube and execute the multi-dimensional queries. DSO supports COM object model to control all the metadata completely and ADO MD API includes objects like CubeDef and Cellset for handling multi-dimensional model [16]. For the integration and uniformity of data generator and evaluation program, Visual Basic 6.0 is used. Visual Basic 6.0 is adequate to the extension for sharing same multi-dimensional model using Microsoft Repository [18] and makes it possible to integrate the evaluation programs of different OLAP systems based on different API using Microsoft OLE DB for OLAP. The program does not include the cube generation process in case of Pilot DSS while cube generation is performed as well in case of MS SQL Server 2000 Analysis Services. Cube generation module performs six sub-processes as follows: Connection to server, creation of database, creation of data source, creation of dimension, creation of cube and cube process. Dimensionality (two or three) and a target cube (seven cubes with defined sparsity types: Three types in the 2-dimensional model and four types in 3-dimensional model) can be chosen by users. An example of result from execution of Drill Down query in a 2-dimensional cube with Random sparsity pattern is illustrated in Figure 10. The table of bottom part shows the retrieved values by the given query.

The evaluation program for Pilot DSS is implemented using Pilot Desktop [19] and Pilot Designer [20]. Pilot Desktop supplies the business analysis environment for single user supporting multi-dimensional queries like drill-down and pivoting. We set up the multi-dimensional cube with Model builder function provided in Pilot Desktop. Users can organize a multi-dimensional model from ODBC-based data sources. Pilot Designer supports a comprehensive object-oriented application development tool for developing visual decision support applications. Figure 10 shows the result of Drill Down query over 3-dimensional cube of Slice type sparsity pattern. You can see the retrieved values of the given query at the table of upper left side.

For the analysis of the performance results, the execution time of a multi-dimensional query is measured in a unit of millisecond. We execute each query 5 times repeatedly and average the observed values. Figures 11 to 13 show the execution time of MS SQL Server 2000 Analysis Services to perform the queries Exact Match, Drill Down and Slice. In fact, another four queries except for the above are also executed, but the observed execution time in those cases is almost same. Since the difference in results of those four queries can be ignorable (actually the difference by the average time is less than 20% of average value), we discuss those three cases. As illustrated in Figure 11, Exact Match query shows best performance in Cluster sparsity pattern. The performance of Drill Down query decreases in case of the 3-dimensional Random sparsity pattern as shown in Figure 12. We note that the execution time
Generally, execution results of four queries, which are Exact Match, Range, Drill Down and Roll up, are faster in case of MS product. We can think that this performance difference comes from that, although two systems use MOLAP storage, the types of two systems are different in detail structure. MS product uses block multicube while Pilot uses series multicube style. Block multicubes use orthogonal dimensions, so there are no special dimensions at the data level, moreover, a cube may consist of any number of dimensions and measures. Series multicubes treat each measure as a separate cube, with its own set of other dimensions. Therefore Pilot DSS treats each measure as a separate cube, this system has to look up both two cubes to get the result of the given queries, which need two measure values,
Sales and Unit Sold [21]. We note that, except for some cases, generally the two systems show good performance in Cluster sparsity pattern. Especially it takes much more time to perform Slice query in case of MS product while Pilot product shows an even performance among all queries. And the performance of Drill Down and Roll Up queries decrease in Cluster sparsity pattern.

6. Conclusion
OLAP applications need to perform a lot of multi-dimensional aggregation. To support the fast and interactive response to users, most OLAP systems precompute the results of some queries and build the summary tables. However, since the actual OLAP data is very sparse, these aggregation processes cause the data explosion problem, which is an important cause of performance bottleneck.

In this paper, we present patterns of sparsity of OLAP data, especially in 2 and 3-dimensional model. We develop test data generator program according to the defined sparsity pattern. Two OLAP systems, MS SQL Server 2000 Analysis Services and Pilot DSS, are used for performance evaluation. We design seven multi-dimensional queries, which are most general in OLAP applications, and implement performance evaluation programs of two OLAP systems.

We find that generalization of sparsity patterns to higher dimensions can be an area of future work. Moreover, based on our research more generalized sparsity handling approach can be considered. By evaluate and analyze the performance of Oracle Express or Essbase, which have their own sparsity control method, we can consider more generalized method for controlling the complex types of sparsity.

References
[9] Y. Zhao, P.M. Deshpande, and J.F. Naughton, An Array-Based Algorithm for Simultaneous Multidimensional Aggregates, in Proc. of ACM SIGMOD ’97, pages 159-170, Tucson, 1997