Column-Oriented Storage Optimization in Multi-Table Queries

Davud Mohammadpur¹, Asma Zeid-Abadi²
¹Faculty of Engineering, University of Zanjan, Zanjan, Iran. dmp@znu.ac.ir
²Department of Computer, Islamic Azad University of Zanjan, Zanjan, Iran. asma.zeidabadi@gmail.com

Abstract. In new generation of applications, using of NoSQL databases growth rapidly. Distributed storing and processing of petabytes of data are the main advantages of this databases. Hence converting current relational databases into NoSQL databases and transferring data has been proposed as a new challenge. In this paper we propose a novel graph traversing algorithm for improving transformation of relational databases into column-oriented NoSQL databases. Results show that this transformation optimizes column-oriented databases to improve retrieving speed in multi-table queries.

Keywords: NoSQL, Column-Oriented, Relational, Relationship, Composite Row-Key

1. INTRODUCTION
Relational model is a well-known and suitable model for storing of organized data. Relational databases provide the best combination of simplicity, query capability, vertical scalability and adaptability. However, problems such as hard horizontal scalability, non-runtime flexibility, non-linear query time, static schema and hard distributioncauses to draw attention from relational model to NoSQL approaches (Rabi, 2011).

Most of NoSQL databases share a common set of characteristics such as: horizontal scalability, using the BASE rules, lack of support for SQL and weak in filtering on non-key fields. In fact, they are the best option for both storing and processing of massive data (Christof, 2012).

In this situation data transformation from relational databases into NoSQL databases has been proposed as a new challenge. Such transformations should keep the strengths of the relational databases specially, retrieving data from combination of multi-tables and filtering on arbitrary fields. However most of NoSQL databases are poor in providing these features (Petter, 2012).

Most of NoSQL experts recommend using of various forms of data storing to handle retrieving needs but the lack of a clear strategy is confusing. Thus even providing a partial solution is very useful.

2. NoSQL DATABASES
There is no accepted categorization of NoSQL databases. References provide different categorizations. However, with due attention to on specifications of data models, in a more accepted categorization; NoSQL databases are presented in four families: key-value databases, document databases, column-oriented databases and graph databases (Adam, 2010).

2.1. Column-Oriented Databases
Column-oriented databases are the best-known type of non-relational databases (Olivier, 2011). A column-oriented database serializes all of the values of a column together, then the values of the next column, and so on. In this type of storage the column is the smallest part of data. It is a tuple containing a name, a value and a timestamp. Generated lists of pairs of names and values can be structured as a column family and with consideration of row-key that forms a row. Each row has a row-key which is similar to primary key in the relational model (Fay, 2013). It should be noted that all rows in a table do not have necessarily the same structure. As shown in the Figure 1, column-oriented databases can be shown in a similar scheme with relational databases. This type of database is very similar in appearance to relational database while in the concept are completely distinct (Fay, 2006).

Some of the typical column-oriented databases are Cassandra, HBase and HyperTable (Lars, 2011) (Olivier, 2011). The most prominent feature that distinguishes them is how to save data (Adam, 2010).

![Figure 1. Column-oriented structure (Kellabyte, 2013)](image-url)
In this paper we propose a novel graph traversing algorithm for improving transformation of relational databases into column-oriented databases.

3. RELATED WORKS
R.P. Padhy et.al reviewed transforming relational model to NoSQL model and introduced all types of NoSQL data structures including: document-oriented, column-oriented, graph-based and key-value models (Rabi,2011). They also described each of them briefly but didn’t introduce any mechanism for transformation.

Ch. Li et.al suggested a heuristic method to transform relational databases data into HBase(Chongxin,2010). The method consists of two phases. In the first phase, relational schema is converted to HBase schema using a relational data model. In this phase they present three recipes that are more useful for HBase application development.

In the second phase, they explain two schemas as a set of nested schema mapping that can be used to create a collection of queries and programs to automatically convert relational data into NoSQL schema (Chongxin,2010). The main contribution of their work is that they recognize the problem of data exchange between relational databases and NoSQL databases and introducing a heuristic approach. These works could be developed in nested mapping technique to match with HBase. Comparing with traditional databases that follow the natural forms of data, it may be classified or duplicated to increase reading speed.

Ch. Huang and W. Lion presented a mechanism to transform relational databases into column-oriented databases (Chin-Chao,2012). They used HBase as a column based platform. They work according to cardinality on the ER model and define some types of transitions: one to one translating, one to many translating and many to many translating. In their transforming the volume of generated data in comparison with the relational database is increased rapidly and number of tables is not decreased. Such translating cause to huge redundancy without any suggested extraction mechanism, therefore query execution performance reduced.

4. TRANSFORMATION MECHANISM
In this section we are going to introduce an approach that helps to improve the placement of massive relational data in column-oriented databases. Our aim is to optimize proposed column-oriented databases for data extraction in massive processing.

Our proposed approach performs in four subsequent steps and it follows two targets: First, providing a more suitable and more tangible schema for transforming from relational databases to column-oriented databases. And the second: providing a suitable arrangement for related table in column-oriented databases. By using the expressed principles in each step we can achieve these two targets.

In our approach we try to put together each table and its related tables in a non-relational structure. As a result, we can decrease the number of tables in the result schema. Also we can improve multi-table query execution speed dramatically.

4.1. Assumptions
It is assumed that the relational database is available and we can emit its image from the tables and relationships between them. The most comprehensive thinking is that we consider the supposed relational database as a graph. That is consisting of each of relational tables as graph nodes and relationships between them as edges. Edges have cardinalities of one to one and one to many.

To achieve a better understanding, the proposed algorithm will be implemented on the database presented in Figure 2, step by step.
4.2. Proposed approach

In our approach, we first select an appropriate starting point and then by introducing an algorithm that is similar to graph traversal, we get an appropriate number of tables in column-oriented databases. Our approach put the related tables in a column-oriented table. In each step by progressing in producing of physical schema, the internal structure of data storage will be completed.

Algorithm procedure is expressed in four steps as follow:

*Step 1: Creating of the waiting list:* In the first step we create a list named waiting list. The waiting list contains all nodes (tables) of the graph that must be navigated. In this list, order is not important and at the beginning of the work, the waiting list is full (with all tables). It is necessary to establish this list since it will prevent forming the cycling node traverse and vain repetition of data and data inconsistency in the next steps. As the relational model shown in Figure 2, the waiting list can be considered as (A, B, C, D, E, F, G, H, I).

*Step 2: Creating of the adjacency matrix:* In this step we create a matrix similar to the adjacency matrix of the graph. The matrix is created on neighborly relationships for each node (table). In this matrix the name of tables, forms the index of each dimensions and matrix elements is defined as:

Each element $a_{ij}$ of matrix will be equal to 1, if one of the two following conditions is satisfied:

1. There is a relationship from table $i$ to $j$ where table $i$ is in side of many.
2. There is a one to one relationship between table $i$ and $j$.

Figure 3 shows the matrix of our sample given in Figure 2.

*Step 3: Selecting the appropriate starting point:* Selecting an appropriate starting point is very important. So that starting from an unsuitable node can definitely cause an inappropriate outcome. Emphasize that the best choice for starting point is a node that corresponding row in the matrix has the minimum number of 1. If there are several nodes with these conditions, the second condition is also checked.

The Second condition is that among nodes which have minimum number of 1s in their rows, a node is chosen that its corresponding column holds the maximum number of 1s.
To select the most appropriate starting point we perform row and column counting of the matrix. Then the node that its corresponding row has the minimum number of 1s and its corresponding column holds the maximum number of 1s is the best starting point.

As it could be seen in our sample, the E, G, H nodes don’t have any 1 in their rows; we should consider the second condition. It specifies that the E is the best starting point.

Step 4: Creating column-oriented table and selecting appropriate completion path: After selecting the starting point, we delete starting point from the waiting list and create a column-oriented table with an arbitrary name and add corresponding table of starting point to column-oriented table as below:

For each table which added to a column-oriented table, we create a column family corresponding to it; each column of the table added as a column qualifier of the column-oriented table.

We define row-key in this way: for each column-oriented table, we consider a composite row-key consist of the primary key of all tables in the column-oriented table and the foreign keys of all tables in that. It's mentionable that all of them appear dash separated in the row-key.

According to the matrix in Figure 2, Table E is selected as a first starting point. Figure 4 shows generated column-oriented table for that.

To handle of related tables of starting point, all connected nodes (tables) to the starting point which the starting point is in the side one of them will be added to stack. These nodes will be deleted from the waiting list (Figure 5).

In order to complete the column-oriented table, nodes in the stack will be deleted one by one. As we explained above, the deleted node (table) should be added to the column-oriented table as a new column family. By deleting a node from the stack any connected nodes to it which the deleted node is in the side one of it will be added to stack. In addition, all added nodes to the stack should be deleted from the waiting list.

Figure 4. Generated column-oriented table by Table E

Now, if the stack is empty, the current column-oriented table considered as a part of the result. Then, the best starting point from the remained nodes in the waiting list will be selected and deleted. It will form a new column-oriented table and the step 4 will be repeated.

Creating to column-oriented table will be continuing until both the waiting list and the stack have been empty. At this point each node of the graph (tables) has a column family in one of the build column-oriented tables.

Figure 5 to Figure 8 show the proposed algorithm on the presented database in Figure 2.

Figure 5. First column-oriented table generation steps

Figure 6. Generated column-oriented table from Figure 5

As you see, all of nodes have traversed and in the end, the waiting list and the stack are empty. All of tables have appeared in their corresponding column families. Here are three column-oriented tables.
5. PERFORMANCE EVALUATION

In this section we execute a suitable comparison between the behaviors of the common structure and the proposed structure. For this purpose we undertake a series of appropriate queries on the HBase (as a column-oriented database) with the two mentioned different structures.

As a case study, we use the Iranian weather organization database (Figure 9). We follow our query results on data that were gained by the weather stations of Zanjan province which were registered in different time periods.

The desired query is computation of daily temperatures average of cities of Zanjan province. Eq. 1 is considered to calculate the average. Note that for every registered day, the temperature was measured and registered every hour.

\[
\text{Eq. 1. } \text{Average}_{\text{StationId}, \text{Date}} = \left( \frac{1}{N} \sum_{i=1}^{N} \text{Temperature}_{\text{StationId}, \text{Date}} \right)
\]

To execute the desired query on the common HBase structure, we create an HTable in HBase for every table in the relational database (common structure). Since there's no relation or foreign key between HTables, to extract required data for a city, we need multiple queries on HTables. Also since there isn't any suitable row-key, each data extraction needs traversing all rows of each HTable.

But in the second case, by applying the proposed structure (Figure 10), because of utilizing the appropriate structure for composite row-key and by applying limitations to tables, the required data will be extracted with minimal process and finally the desired queries will be feasible.
Figure 10. Proposed HBase structure of Iranian weather organization database

Figure 11 demonstrates number of fetched records in the common HBase structure and the proposed structure. For the common structure we have to scan all rows while in the proposed structure we don’t need to scan all rows since there is an effective extraction mechanism that uses an appropriate filtering on the composite row-key to extract rows directly.

Figure 11. Record counts comparison

Figure 12 compare our proposed structure with the common structure based on required times to computing the daily averages of cities. The comparison shows that the needed time in our proposed structure is less than the common structure in orders of magnitude. In addition by increasing the number of records the needed time of the common structure rises over the linear growth while in the proposed structure the increasing time is under the linear growth.

Figure 12. Runtimes comparison
6. CONCLUSION

In this paper by inspiration of graph traversing, we have proposed a method for transforming the relational structure to a column-oriented non-relational structure, which consists of four steps. This simultaneous structure which reached the best physical schema, presents the best internal storing structure of data and also it makes appropriate the data for multi-table queries.

By using this approach in comparison with common structure, we could decrease the number of tables in the presented non-relational structure. Also by using the table’s foreign key in the composite row-key, suitable mechanism provided to multi-table queries and speed of data processing improved.

References


Chin-Chao Huang, Wenching Liou. A Study on The Translation Mechanism From Relational-Based Database To Column-Based Database. Proceedings of The International Conference on Informatics and Applications (ICIA) – Malaysia, 2012, 480-486.


