Relational Data Access for Business Data Analytics

Veit Köppen
Otto-von-Guericke-University Magdeburg, Germany

Andreas Lübcke
regiocom LLC, Magdeburg, Germany

INTRODUCTION

Data, information, and knowledge are dramatically increasing (Korth & Silberschatz, 1997; Naydenova & Kaloyanova, 2010). Although, Data Warehouses are a central access for business data since the 90s, new technologies have to be considered to achieve an efficient access and processing of these multidimensional data. Recently, new architectures have evolved that try to optimize data access for certain applications.

Database systems (DBS) are pervasively used for all business domains. Therefore, DBS have to manage a huge amount of different requirements for heterogeneous application domains. New data management approaches are continuously developed, e.g., new trends are NoSQL-DBMSs (Chang et al., 2006; DeCandia et al., 2007), MapReduce (Dean & Ghemawat, 2008), Cloud Computing (Armbrust et al., 2009; Foster, Zhao, Raicu, & Lu, 2009; Buyya, Yeo, & Venugopal, 2008), to make the growing amount of data manageable for new application domains. However, these approaches are developed for specific applications and need a high degree of expert knowledge.

From a technical point of view, there exist different opportunities to access, process, and analyze data in a more efficient way. On the one hand, the usage of hardware, due to decreasing cost is often a suitable way. On the other hand, this requires techniques that are developed or optimized for main memory usage in data warehousing. Another possibility is to use specialized storage models. Thus, it is possible to store data on all aggregation levels in multidimensional online analytical processing (MOLAP), e.g., the data cube, or only the most interesting data, e.g., iceberg cubes. Furthermore, the data access can be enhanced by considering the architecture. Since the 70s, relational database systems use row stores, that means, tuples (rows of a table) are stored sequentially. In contrast, column stores store data in such a way that attributes (columns of a table) are stored sequentially. This enables efficiency for data access in the domain of data warehousing due to a better access for aggregations. Another challenging optimization is the selection of a suitable index structure. Multi-dimensional analyses have to be supported. Dependent on domain, data, and application scenario different index structures can enhance data processing.

In this chapter, we provide an overview of architectural decision. We focus on the storage architecture for relational database systems.

Row and Column Stores Architecture

Relational database management systems (DBMSs) are developed to manage data of daily business and reduce paper trails of companies (e.g., financial institutions) (Astrahan et al., 1976). This approach dominates the way of data management that we know as online transaction processing (OLTP). Nowadays, fast and accurate forecasts for revenues and expenses are not enough. A new application domain evolves that focuses on analyses of data to support business decisions.

DOI: 10.4018/978-1-4666-5202-6.ch182

Codd, Codd and Salley (1993) define this type of data analysis as online analytical processing (OLAP). In line with others (Abadi, Madden, & Hachem, 2008; Zukowski, Nes, & Boncz, 2008), we state that two disjunctive application domains for relational data management exist with different scopes, impacts, and limitations.

Row store database systems store tuples sequentially. Thus, it is necessary to load or access all data attributes. In the context of OLTP, this is a suitable and often efficient type to store and access data. However, in data warehousing OLAP is in focus, where only some data attributes, often only one, have to be read. Thus, data access with row stores, which read all attributes of data sets, can be computational costly.

Nowadays, we have to decide between column or row stores in the OLAP domain. In fact, column stores are faster than row stores for OLAP workloads (Abadi et al., 2008; Stonebraker et al., 2005). In contrast to row stores, column stores partition data column-wise, i.e., values of a column are stored physically contiguous. This vertical partitioning is advantageous for aggregations on single columns that frequently occur in OLAP workloads. There is no gain without a loss, column stores perform worse on tuple and update operations because after vertical partitioning, column stores have to reconstruct tuples for these operations (Harizopoulos, Liang, Abadi, & Madden, 2006; Abadi, Myers, DeWitt, & Madden, 2007; Manegold, Boncz, Nes, & Kersten, 2004). Compared to column stores, row stores perform better on tuple operations without reconstruction of tuples which is not necessary for row stores. Although, this directly addresses the storage level, optimization is also achieved at query level. Due to the restriction to read access data in OLAP, I/O cost are not very important, instead CPU usage comes into focus.

To further reduce I/O cost, data can be compressed to transfer and process them. Compression reduces the data amount in such a way that an information loss is avoided. Due to the structure of column stores, e.g., same data types are stored together, they achieve better compression rates and thus reduce amount of data to be transferred as well as more data can be processed in main memory. Furthermore, each column can be processed with a different algorithm, which suites most for the corresponding analysis query. Typical compression techniques are run length encoding, bitmaps, null suppression, dictionary encoding, or more complex compression techniques such as gzip or Lempel-Ziv. Compression can increase efficiency of analysis by factor 10, however in practice an increase of factor 3 is more realistic, cf. (Abadi et al., 2008).

Another advantage of column stores is directly achieved in the context of OLAP queries. Due to access of all attributes in row stores and for the reason that OLAP queries only access a limited number of attributes, column stores not only outperform row stores in I/O cost but also for a reduced CPU usage, because in a row store architecture selected attributes have to be extracted in main memory. Column stores only read these attributes that are part of the query. It is also possible to read and process complete blocks of attribute values. This enables an array-like access. Modern hardware can optimize the access and parallelization is possible (loop pipelining). Besides, column stores need CPU for decompression and tuple reconstruction.

Another question for column stores is at which stage data attributes are connected to tuples. There exist two different approaches, early and late materialization. In early materialization, a tuple is already reconstructed at data access. This enables the usage of same operations as for row stores. However, it means that tuples are processed during query processing. Therefore, we use row store algorithms and benefit from reduced storage space and reduced transfer costs from physical memory, e.g., HDD. Hence, column stores can only outperform row stores if the query uses the late materialization strategy. That means, compressed data should be processed attribute-wise as much as possible and a concatenation of attributes should be done in a very late stage. Otherwise, row stores can outperform column stores even if they require a higher I/O.