DISS. ETH NO. 20314

Scaling Out Column Stores: Data, Queries, and Transactions

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

STEFAN HILDENBRAND

Master of Science in Computer Science, ETH Zurich

born March 8, 1982

citizen of Starrkirch-Wil, SO

accepted on the recommendation of

Prof. Dr. Donald A. Kossmann, examiner
Prof. Dr. Carsten Binnig, co-examiner
Prof. Dr. Thomas Neumann, co-examiner
Prof. Dr. Kenneth A. Ross, co-examiner

2012

Abstract

The amount of data available today is huge and keeps increasing steadily. Databases help to cope with huge amounts of data. Yet, traditional databases are not fast enough to answer the complex analytical queries that decision makers in big enterprises ask over large datasets. This is where column stores have their field of application. Tailored to this type of on-line analytical processing (OLAP), column stores enable informed decisions based on queries over huge amounts of data.

The idea of physically organizing relational data in columns instead of rows and applying powerful compression techniques helped a long way with OLAP workloads. Yet, the requirements become increasingly demanding based on the following three issues: a) the amount of data is increasing faster than ever before, b) responses to analytical queries based on a slightly out-of-date snapshot are not sufficient anymore but these queries have to be answered based on live, up-to-date data (so called operational business intelligence), and c) most workloads are not pure OLAP workloads but rather a mix of OLAP and OLTP (on-line transactional processing) workloads. These issues have two consequences for the development of column stores: 1) Column stores must be designed to *scale-out*, i.e., if more machines are available, the system should utilize these additional resources to achieve superior performance. Furthermore, the performance requirements can only be met if all data can be kept in main memory. However, with an increasing amount of data, the main memory of a single machine may not be sufficient for some workloads. In this case, only the aggregated amount of main memory of a number of machines will suffice, hence the need for distribution of the data. 2) The limited support for transaction handling in column stores does no longer suffice for mixed OLAP/OLTP workloads. In column stores, transaction handling is usually implemented as an exception rather than the normal (optimized) case, and only simple, centralized solutions are implemented for distributed transaction processing. As these solutions are insufficient for the increasing demands, the design of column stores needs to focus more on the performance of transactional processing, especially in the distributed scenario.

Column stores apply powerful compression techniques. One compression technique in column stores is dictionary compression: instead of storing the actual data, only a reference to an entry in the dictionary is stored in the table. Most dictionary-compression techniques encode the data in an order-preserving way, i.e., if the code of a value x is smaller than the code of another value y, i.e. $code(x) < code(y)$, it implies that $x < y$ and vice versa. In a distributed scenario, data has to be partitioned. In this case, traditional column stores apply compression after partitioning, i.e., the compression is per column

and partition. However, many query operations process data stored in multiple columns or partitions. Since data from different columns or partitions is compressed using a different dictionary, the data has to be decompressed before processing. To improve the performance of such queries, this thesis presents the use of a global dictionary. The global dictionary encodes data from many columns and partitions in an order-preserving way. The requirements for such a global dictionary are different from traditional dictionaries since it has to store more data and receives more updates. This thesis makes two contributions towards a global dictionary: a) data structures to store the data of an order-preserving dictionary, and b) an order-preserving encoding scheme that supports lazy updates.

When data is partitioned, this can also be leveraged in other areas. If partitioning works well, many operations access data from one single partition. Today's data stores do not take advantage of this fact when it comes to transaction handling. If the system does support distributed transactions, the transactional properties are enforced on all transactions. This is implemented using (conceptually) centralized approaches. One important transactional property is *isolation*: a transaction should see the database as if it were alone in the system, i.e., it should not bother about concurrent transactions. Many data stores support *snapshot isolation* to achieve this property. With snapshot isolation, every transaction gets its own (virtual) copy of the database to work with, and conflicts are sorted out by the system on commit.

The third contribution of this thesis is to improve snapshot isolation in the context of distributed databases, here, transactions that only access data from a single partition should not have to access the central coordinator but proceed only with local interaction. This thesis gives a general definition of distributed snapshot isolation and provides protocol variants to implement it. The main contribution is a technique called *incremental*, which can be implemented transparently to the application. This technique improves performance significantly in certain distributed scenarios.

Kurzfassung

Die Datenmenge, die uns heute zur Verfügung steht, ist riesig und wächst ständig weiter. Datenbanken helfen dabei, mit riesigen Mengen an Daten umzugehen. Aber auch traditionelle Datenbanken können die komplexen analytischen Anfragen, welche Manager in grossen Unternehmen über grosse Datenmengen stellen, nicht mehr schnell genug beantworten. Solche Anfragen sind die Stärke von Column Stores. Zugeschnitten auf dieses sogenannte on-line Analytical Processing (OLAP), ermöglichen Column Stores fundierte Entscheidungen basierend auf Anfragen über grosse Mengen an Daten.

Die Idee, relationale Daten in Spalten statt in Zeilen abzuspeichern, und die umfassende Anwendung von Kompressionstechniken haben die Leistung für OLAP Workloads verbessert. Trotzdem steigen die Leistungsanforderungen ständig weiter. Drei Herausforderungen zeichnen sich ab: a) die Menge an Daten steigt heute schneller als je zuvor; b) es reicht nicht mehr, Anfragen basierend auf minim veralteten Daten zu beantworten, die Antworten auf analytische Anfragen müssen auf aktuellen Daten basieren (sogenannte operational business intelligence); und c) die meisten Anwendungen erfordern nicht ausschliesslich OLAP sondern stellen eine Mischung von OLAP und OLTP (online Transactional Processing) Anforderungen an das System. Diese Herausforderungen haben zwei Konsequenzen für die Weiterentwicklung von Column Stores: 1) Column Stores müssen ihre Leistung steigern können, wenn dem System mehr Maschinen zur Verfügung gestellt werden. Das Schlüsselwort lautet *scale out*. Der Grund dafür ist, dass die Anforderungen nur erfüllt werden können, wenn die Daten im Hauptspeicher gehalten werden. Dies ist aber infolge der zunehmenden Datenmengen nicht mehr immer möglich mit einer einzelnen Maschine. In manchen Fällen ist nur der Hauptspeicher von mehreren Maschinen zusammengezählt ausreichend, um die Anforderungen erfüllen zu können. 2) Die begrenzte Unterstützung von Transaktionen in Column Stores ist nicht mehr ausreichend für die Verarbeitung von gemischen OLAP und OLTP Anfragen. In aktuellen Column Stores ist die Unterstützung von Transaktionen meist eher als Ausnahmefall denn als optimierter Regelfall implementiert. Zudem existieren meist nur einfache, zentralisierte Mechanismen für das verteilte Szenario. Diese Mechanismen sind nicht mehr ausreichend um mit den erhöhten Anforderungen mithalten zu können, daher müssen Column Stores die Verarbeitungsgeschwindigkeit von Transaktionen – besonders im verteilten Szenario – verbessern.

Column Stores benutzen mächtige Kompressionstechniken. Eine verteilte Datenbank erfordert, dass die Daten partitioniert werden. Eine eingesetzte Kompressions-Technik ist die sogenannte Dictionary Compression oder Wörterbuch-Kompression: an Stelle

der eigentlichen Daten wird nur ein Verweis auf den entsprechenden Eintrag in einem Wörterbuch gespeichert. Darüber hinaus kodieren die meisten Dictionary Compression-Verfahren die Daten in einer ordnungs-erhaltenden Weise, d.h., wenn der Code eines Wertes x kleiner als der Code eines anderen Wertes y ist ($code(x) < code(y)$), folgt daraus, dass $x < y$. Im verteilten Szenario wenden die meisten Column Stores Kompression nach der Partitionierung an. Das bedeutet, die Kompression findet pro Spalte und Partition statt. Allerdings verarbeiten viele Abfrage-Operationen Daten aus mehreren Spalten oder Partitionen. Da die Daten aus verschiedenen Spalten oder Partitionen mit unterschiedlichen Wörterbüchern komprimiert wurden, müssen die Daten vor der Verarbeitung dekomprimiert werden. Um die Geschwindigkeit von solchen Anfragen zu verbessern, präsentiert diese Arbeit die Verwendung eines globalen Wörterbuches. Das globale Wörterbuch kodiert die Daten von vielen Spalten und Partitionen in einer ordnungs-erhaltenden Weise. Die Anforderungen an ein solches globales Wörterbuch unterscheiden sich von traditionellen Wörterbüchern, da es mehr Daten speichern und mehr Updates verarbeiten muss. Diese Arbeit leistet zwei Beiträge für ein globales Wörterbuch: a) Datenstrukturen um die Daten eines ordnungs-erhaltenden Wörterbuch zu speichern und b) ein ordnungs-erhaltendes Codierverfahren, das es erlaubt, Updates erst zu verarbeiten, wenn die Zeit günstig ist.

Wenn die Daten partitioniert sind, kann das auch in anderen Bereichen Vorteile bringen. Wenn die Partitionierung gut funktioniert, müssen viele Operationen nur auf Daten aus einer einzigen Partition zugreifen. Aktuelle Datenbanken nutzen diese Tatsache bei der Abwicklung von Transaktionen allerdings nicht aus. Falls das System überhaupt verteilte Transaktionen unterstützt, werden die Transaktions-Eigenschaften für auf alle Transaktionen gleichermassen erzwungen. Hierzu werden meist (konzeptuell) zentralisierte Ansätzen benutzt. Eine wichtige Eigenschaft von Transaktionen ist *Isolation*: Eine Transaktion sieht die Datenbank als ob sie alleine im System wäre. Parallele Transaktionen stören sich also nicht gegenseitig. Viele Datenbanken setzen *Snapshot Isolation* ein, um dies zu erreichen. Mit Snapshot Isolation erhält jede Transaktion eine eigene (virtuelle) Kopie der Datenbank mit der sie arbeiten kann. Konflikte werden durch das System am Ende jeder Transaktion erkannt.

Als dritter Beitrag dieser Arbeit wird Snapshot Isolation im Zusammenhang mit verteilten Datenbanken verbessert: Transaktionen, die nur auf Daten aus einer einzigen Partition zugreifen, sollen nicht über den zentralen Koordinator laufen, sondern nur mit lokaler Interaktion abgewickelt werden. Diese Arbeit präsentiert erst eine generelle Definition von verteilter Snapshot Isolation und zeigt dann Varianten auf, um diese Definition zu implementieren. Das wichtigste Element dabei ist eine neue Technik, genannt *incremental*, die für die Anwendung transparent implementiert werden kann. Diese Technik verbessert die Leistung erheblich, wenn die Partitionierung gut funktioniert (d.h., wenn viele Transaktionen lokal sind).