Review Report-1

By Harsha Salim (USC ID: [redacted])

Paper: P.-A. Larson, et al., SQL Server Column Store Indexes, in SIGMOD 2011

Summary

This paper introduces a new database index type called a column store index, which was added to the SQL Server 11.0(Denali). It also talks about a new set of query operators that processes batches of rows to better utilize the column indexes. This novel feature has been integrated fully with the database system, including query processing and optimization. Prior to its release, SQL Server traditionally used a row-based storage system, which worked well with transaction processing, but ill-suited for data warehousing applications. For the latter, queries focused on a specific set of columns in the table, spanning over a lot of records. A column-based storage would minimize the data processed to only the relevant columns, and this feature combined with the new batch processing operators greatly improved query performance. While column-based storage had been discussed and proposed much earlier, this was the first time a pure column store was fully incorporated into an established database system. To minimize the storage and I/O times, the data is encoded, reordered optimally, and compressed, in a way so that some operations can directly be done on this data. These compressed segments make use of existing SQL Server storage mechanisms and catalog implementation, which automatically makes existing features available to them. This includes locking, logging, recovery, partitioning, etc. The encoding can be a dictionary-based encoding or a value-based encoding, depending on the nature of the data. This is followed by reordering the rows within the segments to ensure maximum compression. Run-length encoding (RLE) compression or bit packing is done on the modified data, which is finally stored as segments. The introduction of batch processing operators is done in an effective manner so that it extended the query optimizer to choose row or batch processing or a mix of both, based on the nature of the queries. Bitmap filters and certain access methods further enhance the performance. There were two types of experiments done to validate the proposed features. The first identifies the benefits of compression introduced by the column store index on six databases. It showed that it was 1.8 times more effective than the highest compression offered by SQL Server row store structures. The second experiment compared the elapsed and CPU times of four different queries on SQL Server restricted to row stores only, to that with both row and column stores, on a warm and cold buffer pool. The results of these experiments clearly showed a 10 times improvement in elapsed time, and a significant improvement in CPU time, highlighting the efficiency of the new system.

Paper Strengths

The paper clearly mentions the need for column-based indexes, and how the existing row-based storage would not work well for data warehousing applications. It mentions that column-based storage was proposed much earlier, as far as the early 1970's, but this was the first time it was fully incorporated into an established database system, and that too, in a manner that it enhanced the existing capabilities rather than inhibiting it. It effectively utilized the existing blob storage mechanism and catalog implementation of SQL Server, so it was able to automatically attain the existing features available for row-based storage indexes – which include locking, logging, recovery, partitioning, mirroring, replication and more.

While adding such an index would introduce more storage needs, the paper goes into detail regarding the methods it employs to minimize this impact. It first encodes the data, then reorders it, and finally compresses it. This is done in a manner such that some operations can be done directly on this compressed data, which saves time. There were also cache improvements for column storage, wherein the cache was able to handle larger segments, and the data was stored consecutively in the cache, not scattered across it. To improve I/O performance, read-ahead was applied both within and among segments, and to data dictionaries. When column segments were written to disk, additional compression could be applied to it. All these measures reduced the effect of the overhead of adding on column storage.

The paper also mentions the advantages of extending the existing SQL Server engine with the column-based indexes over creating a new engine specific to data warehousing applications. Customers did not have to invest in a new engine, and transfer data between the two. This would be very useful for customers who had varied queries. This reduced implementation costs, as a lot of the existing functionality was also available to the new indexes as mentioned earlier. Query plans were able to mix both row and column-based operators, thereby choosing the most efficient operations. The queries can also dynamically switch at runtime from using the column-based batch operators to using row operators as necessary, which shows the flexibility of the system. Feature orthogonality is an additional benefit, and it does not affect the existing operators and features and works well with them.

Two types of experiments were conducted, the first one examining the benefits of compression on six different databases which contained real-life data. It showed the size of the uncompressed tables, their corresponding column store index size, and the compression ratio. It mentioned the effectiveness of column store compression to be 1.8 times more than the SQL Server PAGE compression, which was the highest form of compression implemented for row-based storage. The second experiment compared the performance of queries on row based and a combination of row and column-based storage. There were four different types of queries, and the experiments were conducted on a cold and warm buffer pool. Three of the queries shows a significant improvement in both CPU and elapsed time when the systems were allowed to use column-based storage along with row-based storage. The only query that did not show an improvement was one that

had a restrictive predicate making it unsuitable to be queried by a column-based index. The paper goes into depth explaining the queries and their query plans to further improve the reader's understanding of the new feature, while clearly showcasing the benefits it adds.

Paper Weaknesses

Column store indexes do not efficiently support point queries and range scans, and the paper acknowledges that. It also does not offer any sort order, and hence does not offer statistics. The paper also mentions the problems related to CPU utilization that can increase because of adding a column index, but it also goes into detail regarding system modifications to reduce its impact.

While using a mix of row operators and batch operators may be efficient in terms of execution, there is an added overhead of converting the data from one format to the other. To mitigate this, the initial implementation tries to limit the number of conversions. Some operators can output data in row or batch form, so transitioning between the two formats can also naturally happen when these operators are used.

Some features, like direct update and load of tables with column store indexes are not supported, but they are acknowledged and mentioned under the work in progress section of the paper. The present options to load data include flushing the data and replacing it, or range partitioning. An additional restriction mentioned is that using a column store index as the primary organization for data is not supported. Hence a user who is primarily interested in data warehousing may still be better suited to choose a column-based store.

Finally, the first experiment regarding the effects of compression on six different databases does not show a clear picture regarding its effectiveness. While the size of the column-based store was mentioned for each data set, the size of the row-based stores after compression was not provided. This does not allow the reader to compare the effectiveness of compressions implemented in row-based and column-based stores. It is mentioned, however, that column stores are 1.8 times more effective than their row-based counterparts.

Opinion

Despite the minor weaknesses, I feel that the paper presents a strong case for column-based stores and its effectiveness for data warehousing queries. It clearly outlines how it was seamlessly integrated into an existing database system, and how it can be paired well with the existing features to provide a more efficient query processing capability. Hence, I accept this paper.