

Positional Delta Trees for Updating Column Stores

Sándor Héman, CWI, Amsterdam (S.Heman@cwi.nl)

November 5, 2007

1 Abstract

In this talk we introduce a datastructure, the *positional delta tree* (PDT), for maintaining differential updates against a read-optimized, column-oriented, relational DBMS, and present techniques for merging such updates into an ongoing table scan, to provide an up-to-date view of the data stored in the database.

Traditionally, transaction processing systems have been built on top of row-oriented physical layouts, or *row stores*, where each unit of I/O contains several *full* tuples, and updates can therefore be performed in-place, after performing the *random* I/O needed to retrieve the I/O unit to be updated. Such an architecture is optimized for write-intensive workloads.

Column stores, on the other hand, are known for their excellent performance on read- and data-intensive query workloads. The most important reason for this high read performance is that I/O units are always filled with data from a single attribute (column) solely. This allows a query to restrict its I/O accesses to the attributes it needs, and provides interesting opportunities for light-weight, data type and distribution specific compression techniques.

Several problems arise, however, if one needs to modify such a read-optimized layout. To insert a new tuple or remove an existing tuple, for example, we need to fetch an I/O unit for each attribute of the relation, resulting in I/O proportional to the number of attributes. The fact that column data is often stored contiguously, and potentially even in compressed form, complicates modifications even further, as data needs to be shifted and de-/re-compressed.

Given these observations about column stores, we propose to defer touching the read-optimized image, or *read-store* (RS), as long as possible, and instead buffer differential updates in a separate *write-store* (WS). The updates from this write-store are then merged into column scans, at run-time, to present the user with an up-to-date view of the database.

We introduce the *positional delta tree* (PDT) datastructure, which provides for efficient maintenance and mergeability of differential updates. The PDT maintains updates on basis of row index, and is always aware of a tuples current row index, which is subject to change under insertion and deletion of tuples. As PDTs work on basis of row index, they can be stacked, which opens doors for memory hierarchy optimizations. We propose to employ three layers of PDTs: one small PDT that fits the CPU cache and can easily be copied for snapshot isolation. Below this small PDT, we have a second, main-memory resident PDT, into which the cache PDT gets merged once it grows out of the cache. At the bottom layer, we have a disk resident PDT, which allows us to perform fast checkpoints by avoiding modifications to the read store section when system load is high.