IndexDev Bitmap

= Bitmap Indexing =

Introduction

This document explains the proposed design for adding a bitmap index handler (https://issues.apache.org/jira/browse/HIVE-1803). Bitmap indexing (http://en.wikipedia.org/wiki/Bitmap_index) is a standard technique for indexing columns with few distinct values, such as gender.

Approach

We want to develop a bitmap index that can reuse as much of the existing Compact Index code as possible.

Proposal

First implementation

This implementation confers some of the benefits of bitmap indexing and should be easy to implement given the already existing compact index, but it does few of the optimizations such as compression that a really good bitmap index should do.

Like the complex index, this implementation uses an index table. The index table on a column "key" has four or more columns: first, the columns that are being indexed, then _bucketname, _offset, and _bitmaps. _bucketname is a string pointing to the hadoop file that is storing this block in the table, _offset is the block offset of a block, and _bitmaps is an uncompressed bitmap encoding (an Array of bytes) of the bitmap for this column value, bucketname, and row offset. Each bit in the bitmap corresponds to one row in the block. The bit is 1 if that row has the value of the values in the columns being indexed, and a 0 if not. If a key value does not appear in a block at all, the value is not stored in the map.

When querying this index, if there are boolean AND or OR operations done on the predicates with bitmap indexes, we can use bitwise operations to try to eliminate blocks as well. We can then eliminate blocks that do not contain the value combinations we are interested in. We can use this data to generate the filename, array of block offsets format that the compact index handler uses and reuse that in the bitmap index query.

Second iteration

The basic implementation's only compression is eliminating blocks where all rows are 0s. This is unlikely to happen for larger blocks, so we need a better compression format. What we can do is do byte-aligned bitmap compression, where the bitmap is an array of bytes, and a byte of all 1s or all 0s implies one or more bytes where every value is 0 or 1. Then, we would just need to add another column in the bitmap index table that is an array of Ints that describe how long the gaps are and logic to expand the compression.

Example

Suppose we have a bitmap index on a key where, on the first block, value "a" appears in rows 5, 12, and 64, and value "b" appears in rows 7, 8, and 9. Then, for the preliminary implementation, the first entry in the index table will be:

`https://issues.apache.org/jira/secure/attachment/12460083/bitmap_index_1.png`

The values in the array represent the bitmap for each block, where each 32-bit BigInt value stores 32 rows.

For the second iteration, the first entry will be:

`https://issues.apache.org/jira/secure/attachment/12460124/bitmap_index_2.png`

This one uses 1-byte array entries, so each value in the array stores 8 rows. If an entry is 0x00 or 0xFF, it represents 1 or more consecutive bytes of zeros, (in this case 5 and 4, respectively)