Vectorized Text InputFormat Design

https://issues.apache.org/jira/browse/HIVE-6234

Eric N. Hanson

eric.n.hanson@microsoft.com

1/24/2014

Revision 01

Introduction

This JIRA is to implement support for vectorized scan input of text files (plain text with configurable record and field separators). This should work for CSV files, tab delimited files, etc.

The goal is to provide high-performance reading of these files using vectorized scans, and also to do it as an extension of existing Hive. Then, if vectorized query is enabled, existing tables based on text files will be able to benefit immediately without the need to use a different input format. After upgrading to new Hive bits that support this, faster, vectorized processing over existing text tables should just work, when vectorization is enabled.

Another goal is to go beyond a simple layering of vectorized row batch iterator over the top of the existing row iterator. It should be possible to, say, read a chunk of data into a byte buffer (several thousand or even million rows), and then read data from it into vectorized row batches directly. Object creations should be minimized to save allocation time and garbage collection (GC) overhead. If it is possible to save CPU for values like timestamps, dates and numbers by caching the translation from string to the final data type, then that should ideally be implemented.

I conjecture that it can be possible to get maybe a 2 or 3 times reduction in CPU time, end-to-end, for running a query in vectorized mode using this new vectorized text input format, compared to running a query in row mode with the standard text input format. This will of course need to be verified with experiments.

Controlling which text InputFormat is used

A new InputFormat that is built-in to Hive, VectorizedTextInputFormat, shall be implemented. This will support vectorized iteration over data. It can contain an instance of org.apache.hadoop.mapred.TextInputFormat. If hive.vectorized.execution.enabled is true, then if the InputFormat for a table is VectorizedTextInputFormat, query execution will be vectorized, provided that other requirements for vectorization are met.

If hive.vectorized.execution.enabled is false, then if the InputFormat is VectorizedTextInputFormat, it will create an internal instance of TextInputFormat, and delegate reading the data in row-at-a-time mode to it.

Another setting will also be introduced:

hive.vectorized.text.input.default = { true | false}

If this setting is true, then existing tables with TextInputFormat as their InputFormat shall automatically use VectorizedTextInputFormat instead if hive.vectorized.execution.enabled = true. This allows users with existing text tables to use vectorized input without altering their table.

Optimizing vectorized data reading

A sketch of the low-level implementation design is as follows. The nextBatch() method of VectorizedTextInputFormat will first check if a buffer of data has been filled. If not, it will read a good-sized chunk (say 4MB) into ByteBuffer buf. Then, a VectorizedRowReadingFrame will be initialized to point at the next one row worth of data in buf, with starts and lengths for each field. This initialization will not cause new objects to be created, in order to save CPU time for allocation and GC. Then, the row will be added to the VectorizedRowBatch being filled. The current vector element for each field of the current row will be directly filled from the source data.

For a number field, data will be converted directly from a byte[], start, and length to a number without creating a new object, and written to the target vector element.

For a string field, if data is UTF-8 in buf, setRef() will be called for the BytesColumnVector target to point to the data directly, without need to copy.

For possibly expensive translation, like converting from string to an long integer representing a timestamp, data may optionally be cached, so if the same input value is found again, computing the result will not be required. This will be an optimization that can be added after the end-to-end functionality is working, if it shows a reasonable performance benefit.

Only columns used by the query will be translated into vectors. Other columns in then source data will be skipped.

Alternative Design Option

An alternative design considered is to use a VectorizedBatchReadingFrame to initialize references into a full VectorizedRowBatch worth of text data. Then, fill the target vectors of the VectorizedRowBatch from this frame one full vector at a time.

I think this is more complicated than the other approach. It has better code locality, and better cache locality in the target vector. But it has worse locality in the source text data (because the data has to be looked at once to initialize the frame, and again to copy it into vectors). It is not clear it will outperform the row-at-a-time filling of the row batches. I don’t plan to pursue this option, although given enough time, both could be implemented and benchmarked against each other.