Native ORC Reader/Writer

Owen O'Malley

October 2014

1 Requirements

Although the current Java ORC file reader and writer are convenient for much of the Hadoop ecosystem, the ability to read and write ORC files without using a JVM would help adoption of the format. Our goal is to enable reading and writing ORC files to or from local files, HDFS, or third party object stores.

1.1 Platform

The native ORC reader/writer must at least support the following platforms:

Linux CentOS 6 (gcc 4.7)

Windows Windows Server 2012 (Visual Studio 2013)

Mac OS X 10.9 (clang 6.0)

1.2 Features

C++ API The native ORC reader/writer will provide a C++ (x11) API to read and write ORC files.

Source Agnosticism The native ORC reader/writer must be able to read or write from local files, native HDFS client, or third part object stores.

Read Compatibility The native ORC reader/writer must be able to read any file produced by the Java ORC writer from Hive 0.11 to Hive 0.14.

Write Compatibility The native ORC reader/writer must default to writing files that are readable by Hive 0.14's Java ORC reader. There must be an option of creating files that are readable by Hive 0.11's Java ORC reader forward.

Full type coverage All of the types from the Hive's Java ORC writer must be supported.

Input Splitting The native ORC reader must allow readers to limit row readers to a part of the file. This enables multiple tasks to independently process part of the file and ensure that each row is processed exactly once.

Vectorized Reading/Writing All reading/writing will be done using row batches consisting of up to 1000 rows.

Column Projection The native ORC reader must be able to select a subset of the columns for reading. Columns that are not selected should not be read or decompressed.

Asynchronous Input/Output To support asynchronous input/output, the native ORC reader/writer will use libuv, which is supported on Linux, Windows, and Mac OS.

1.3 Vectorized Row Batch

The native ORC reader/writer will only provide a reader and writer that take a vectorized row batch instead of a row by row approach. Our experience in Hive is that for high performance applications that reading, processing, and output should be done in batches of roughly 1000 rows. Each primitive column within that batch will be represented as an array of a corresponding primitive type. Each batch should be read or written using a single call to the native ORC reader/writer.

The mapping for primitive types will be (hive type ⇒ array type):

- boolean ⇒ long array
- tinyint ⇒ long array
- smallint ⇒ long array
- int ⇒ long array
- bigint ⇒ long array
- float ⇒ double array
- double ⇒ double array
- string ⇒ void* (start) array and long (length) array
- binary ⇒ void* (start) array and long (length) array
- timestamp ⇒ long array
- decimal ⇒ int128 array
- varchar ⇒ void* (start) array and long (length) array
- char ⇒ void* (start) array and long (length) array

Null values will be represented using an isNull byte array. If none of the values within the batch are null, a flag will denote that.

For complex Hive types, the mapping will be:

- struct ⇒ a list of subcolumns
- union ⇒ a long array selecting the variant (0 to N-1) and a list of subcolumns.
- list ⇒ a long array of the number of elements in each list and a subcolumn for the list elements.
- map ⇒ a long array of the number of elements in each map and subcolumns for the keys and values.

1.4 Finding an Apache Home

It isn't clear yet where to put the Native ORC reader/writer in Apache. One option is to put it in Hive, although Hive does not currently have any native code. Another option would be to create a new ORC project. A final option would be to put it into Tez or LLAP. For now, we can use the original ORC github repo (http://github.com/hortonworks/orc).

2 Plan

Here's a rough sketch of the current plan. With just me, I'd pretty much go through them in order.

1. Phase 1 - Reader (76 days)
   (a) Write complete specification of the current ORC file format (5 days)
   (b) Implement integer rle version1 (5 days)
   (c) Implement integer rle version2 (10 days)
   (d) Implement file footer reader (2 days)
   (e) Implement the integer types reader (5 days)
   (f) Implement the float types reader (3 days)
   (g) Implement the string reader (3 days)
   (h) Implement the date and timestamp reader (3 days)
   (i) Implement the decimal reader (3 days)
   (j) Implement the char and varchar reader (2 days)
   (k) Implement the complex types (list, map, struct, union) reader (10 days)
   (l) Integrate with Native HDFS client (10 days)
   (m) System tests - randomized (5 days)
   (n) System tests - performance testing (10 days)
2. Phase 2 - Writer (38 days)
   (a) Implement file footer writer (2 days)
   (b) Implement the integer types writer (5 days)
   (c) Implement the float types write (3 days)
   (d) Implement the string writer (3 days)
   (e) Implement the date and timestamp writer (2 days)
   (f) Implement the decimal writer (3 days)
   (g) Implement the char and varchar writer (2 days)
   (h) Implement the complex types (list, map, struct, union) writer (3 days)
   (i) System tests - randomized (5 days)
   (j) System tests - performance testing (10 days)
3. Phase 3 - Predicate pushdown (28 days)
   (a) Implement seek to row number (10 days)
   (b) Implement SearchArgument (sarg) interfaces (3 days)
   (c) Integrate sarg predicate pushdown into reader (15 days)
4. Phase 4 - ACID format support (33 days)
   (a) implement acid utils (3 days)
   (b) implement merger (10 days)
   (c) implement interface to Hive transaction manager (10 days)
   (d) system testing (10 days)