ORC Support Solution

Overview

(note: This doc is applied only for the first patch)

Just like Parquet file format, ORC (Optimized Row Columnar) is another columnar file format widely used in Hadoop ecosystem. In Hulu, most of the Hive tables are stored in ORC format. However, Impala currently cannot read from ORC tables. We have to transform tables we needed in analytics into parquet format, which is redundant and inefficient. Finally, we decided to support reading from ORC format tables in our branch of Impala.

Our philosophies include:

1. Don't reinvent the wheel
2. Make things simple

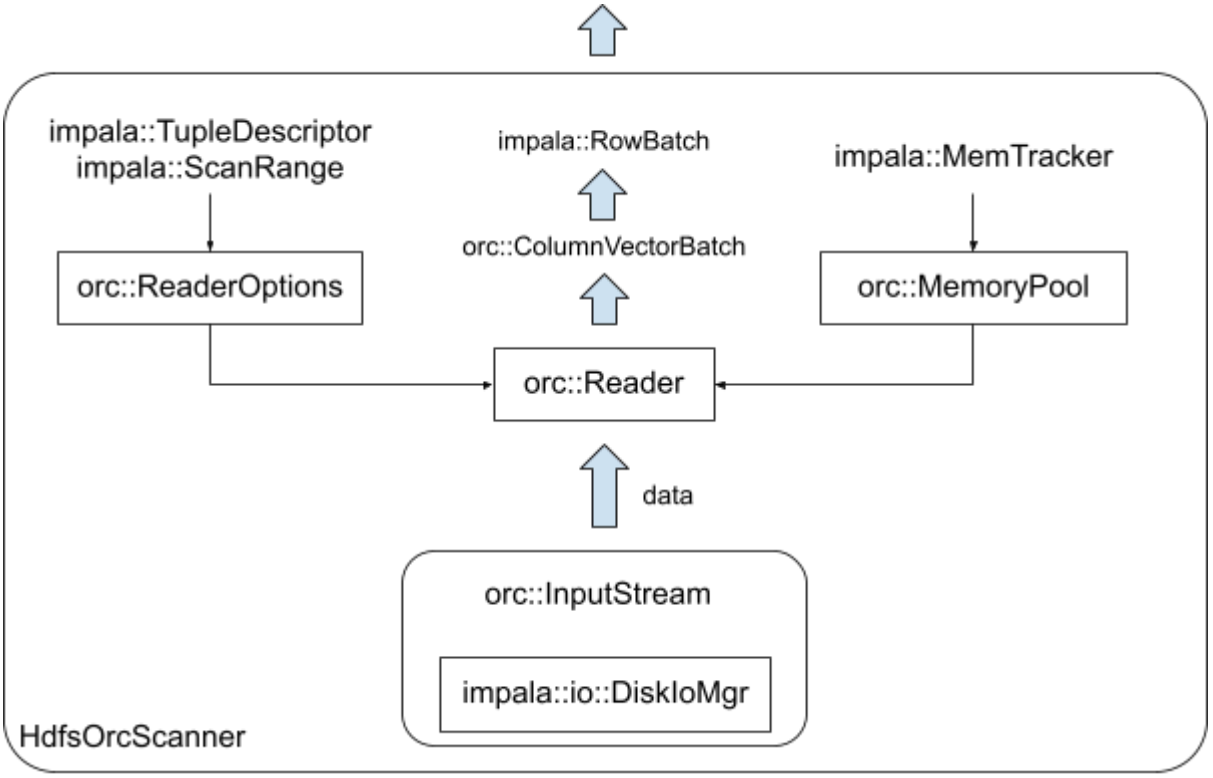
There's a mature C++ orc-reader: https://github.com/apache/orc/tree/master/c%2B%2B. Thus, we decides to integrate it into Impala, instead of building an ORC scanner from scratch.

We implement an HdfsOrcScanner which performs like a middle layer between the orc-reader and Impala. It supplies input needed from the orc-reader, tracks memory consumption of the reader and transfers the reader's output into Impala's RowBatch.

Instead of linking the orc-reader as a third party library, we integrate it in the code level, leaving chances for further optimization, e.g. Predicate Pushdown, Code Generation. Currently, we haven't changed any codes of the orc-reader.

As a first step, we just support reading primitive types. We haven't handle complex types (i.e. struct, array, map) yet. Besides, writing into ORC tables via Impala has not been supported neither.

Architecture



OrcReader

The original reader reads ORC files from local disk. It's able to deal with schema conversions and many corner cases. When encounter invalidate data, it'll throw an orc::ParseError.

What's the input of the orc-reader?

To integrate it as an hdfs scanner in Impala, we need to supply

1. **orc::MemoryPool**, without which the orc-reader will use std::malloc and std::free directly, and we can't track the memory it used.
2. **orc::InputStream**: we implement this interface so the orc-reader can read data from impala::io::DiskIoMgr
3. **ReaderOptions**: describing the scan ranges and selected columns

What's the output of the orc-reader?

An orc::ColumnVectorBatch object, which is a batch of tuples containing all the selected columns. We use it as the scratch batch and copy its data into the impala::RowBatch.

The lifecycle of an orc-reader

We create an orc-reader for each stripe (i.e. row group in ORC). So we can release resources at the end of each stripe.
The orc-reader will keep reading scratch batch until encounter the end of stripe, parse error or cancellation.

Functions

Here is the call trace and comments of functions in the scanner.

- HdfsOrcScanner::IssueInitialRanges // same as the parquet scanner
- HdfsOrcScanner::Open
    - ProcessFailTail
        - ParsePostscript // then we know footer length, compression
          // type, compression block size, etc.
        - Deal with footer length larger than original split size
        - ParseFooter
        - Get file meta, e.g. schema, stripes meta, num of rows, etc.
    - UpdateReaderOptions
        - Translate impala::TupleDescriptor into orc::ReaderOptions
        - Deal with table schema and file schema not match
        - Supply orc::MemoryPool implementation
        - Supply serialized FileTail so orc-readers don't need to parse it again
- HdfsOrcScanner::ProcessSplit // same as other scanners
- GetNextInternal
    1. Transfer remaining tuples in scratch_batch_ (orc::ColumnVectorBatch) to dst_batch (impala::RowBatch)
    2. NextStripe
        a. target the next valid stripe
        b. create an orc::reader for it
    3. AssembleRows
        a. read a new scratch_batch_ (orc::ColumnVectorBatch)
        b. TransferScratchTuples
            i. ReadRow // translate data into impala tuple layout
        c. Loop until step "a" cannot read out more data
- HdfsOrcScanner::Close // Clear resources

Error Handling

How to interrupt the orc-reader?
We use exception to interrupt the orc-reader for errors like cancellation, memory limit exceeded. For example, when we cannot read from DiskIoMgr, we throw an std::runtime_error inside the ScanRangeInputStream::read. The orc-reader will throw it out

as well. Then we catch the exception in HdfsOrcScanner::AssembleRows and cancel the while loop.

Notes

Some difference between ORC and Parquet:

1. FileTail structures are different.
2. Terminology of Stripe v.s. RowGroup
3. If using compression, the ORC file is split into chunks and each chunk is compressed individually. If the compressed size of a chunk is larger than original size, it won't be compressed. https://orc.apache.org/docs/compression.html

Future Work

- Support reading complex types
- More optimizations
    - Predicate Pushdown (min-max, dictionary)
    - Code Generation
- Support writing into ORC tables