Design Considerations

Kevin Liew

May 30, 2016

1 RPC

How are ResultSets transferred? Batches contained in Thrift objects? Contiguous batches of TRowSet.

```
ie.
ResultSet{
    TRowSet batch1{
        TColumn1{row1, row2},
        TColumn2{row1, row2},
        columnCount = 2
    }
    TRowSet batch2{
        TColumn1{row3, row4},
        TColumn2{row3, row4},
        columnCount = 2
    }
}
```

except that the columns are in 'binaryColumns' rather than a list of TColumn after HIVE-12049

2 Interface

Should we give priority to the server or client preference for compressors? ie. the client prefers snappy first, gzip second, but the server prefers gzip first, snappy second. Should we use gzip or snappy? The server's preference should have priority.

3 Compressor-Decompressor

How can we ensure that result sets are compressed for cases where the query does not invoke any map-reduce tasks (ie. select * from

table)? Our SerDe is not called so if we write our compressor within the SerDe, then some queries will not be compressed.

How do we handle the case where the compressor throws an exception while handling a batch? If an exception is thrown while processing a column, it won't be compressed for that batch. The client will receive it as an uncompressed column and will know that it does not need to be decompressed. The server will still try to compress that column for all other batches and those that succeed will be compressed.

Where does decompression occur in the client? Will we operate on ColumnBasedSet or directly on TRowSet? Which files should I look at? In ColumnBasedSet, we deserialize TColumn or TEnColumn from 'binaryColumn' of a TRowSet.

hive.server2.thrift.resultset.max.fetch.size: "Max number of rows sent in one Fetch RPC call by the server to the client." If the client receives batch-by-batch, then we don't need to store the batch size in TEnColumn because the client will receive and decompress each batch separately instead of receiving one huge blob with all batches? So does the client receive the batches separately or in one blob? Batch-by-batch so we do not need a list of batch sizes.

May 24 2016

Decompress needs the type information so that a CompDe implementation can delegate to other compressors based on data-type. Pass in type information as input? Or a TEnColumn, which has type info? Or encode the type information as part of the binary? Encoding type-info into the binary: places a lot of requirements on implementations of the CompDe. TEnColumn has all the necessary information without imposing requirements that the plugin encode certain data into the output binary, so lets use that.

ODBC drivers will not implement a Java interface so we could have a compressor interface instead of a compressor-decompressor interface. The same applies to SerDes and yet SerDe includes 'deserialize' as part of the interface.

Decompressor should return something that is language-agnostic. Thrift object, TColumn? If we return Object, we can go from binaryColumns to TEnColumn to ColumnBuffer, but Object is specific to Java. If we return TColumn, we'd go from binaryColumns

to TEnColumn to TColumn to ColumnBuffer, adding an extra step. How can we return something for both Java and C++ but not add an additional step to the deserialization process? 'decompress' will only be called from a Java client so we should optimize the function output for Java. ODBC drivers will have their own implementation so the output of 'decompress' does not need to be language-agnostic.

If the user disables serialization in the task nodes (hive.server2 .thrift.resultset.serialize.in.tasks), is our SerDe called at all? No

If the user disables serialization in the task nodes but wants compression? Our SerDe is not called and so compression will not occur. enColumns no longer has a use and can be removed.

May 26 2016

Compressor-Decompressor

- give more responsibility to the compressor
  - compressor should handle exceptions
  - compressor should be given the entire set of columns so that it can group similar columns for more efficient compression
  - provide a way for the client to pass parameters to the plugin
    - pass byte array to an init function
- make compress-decompress symmetric

Configuration

- the entry class must be provided to use custom compressors
- should be able to use different versions of the same algorithm so that plugins can have backward compatibility. ie. Snappy v2 vs v3
  - version can be sent as part of the parameter byte array
- how to allow client to configure the byte array that will be sent to the server for each plugin?

Next

Does the compressor have access to information about the server state? ie. for the server-side plugin, will the 'init' function be able to accept/reject the compressor request based on server-state?