HIVE-13680: Provide a way to compress ResultSets

Kevin Liew

May 31, 2016

Abstract

Hive data pipelines invariably involve JDBC/ODBC drivers which access Hive through its Thrift interface. Consequently, any enhancement to the Thrift vector will benefit all users.

Prior to HIVE-12049, HiveServer2 would read a full ResultSet from HDFS before deserializing and re-serializing into Thrift objects for RPC transfer. Following our enhancement, task nodes serialize Thrift objects from their own block of a ResultSet file. HiveServer2 reads the Thrift output and transfers it to the remote client. This parallel serialization strategy reduced latency in the data pipeline.

However, network capacity is often the most scarce resource in a system. As a further enhancement, network load can be eased by having task nodes compress their own block of a ResultSet as part of the serialization process.

1 Introduction

The changes proposed herein draw from Rohit Dholakia's design document and patches for HIVE-10438, which implemented compression on HiveServer2. Now that HIVE-12049 has been committed, compression can take place in parallel on the task nodes.

Our goals for this enhancement are to:

- improve performance out-of-the-box for new clients
- maintain compatibility with old clients
- provide flexibility yet security
- confer a simple interface

2 Design Overview

2.1 Compressor-Decompressor Interface

We define a compressor-decompressor (CompDe) interface which must be implemented by plugins. A default CompDe will process all data-types

using the Snappy algorithm. Type-specific compression-decompression can be achieved by implementing a CompDe that delegates to other CompDes based on a case-switch block. CompDes may also pass specific data-types through unprocessed.

org.apache.hive.service.cli.CompDe;CompDe.java

```
@InterfaceAudience.Private
@InterfaceStability.Stable
public interface ColumnCompDe {
  public boolean init(HashSet config);
  public byte[] compress(ColumnBasedSet colSet);
  public ColumnBasedSet decompress(byte[] compressedSet);
}
```

Operating in the final task nodes, 'compress' takes a batch of rows contained in a ColumnBasedSet (number of rows will be equal to hive.server2.thrift.resultset.max.fetch.size except for the last batch) and outputs a binary blob. The compressor is free to pack additional details such as look-up tables within this blob.

The client receives results batch-by-batch, calling 'decompress' on each blob to receive a ColumnBasedSet.

2.2 Configuration options

Option	Default	Description
hive.resultSet .compressors	snappy	A list of keywords for available compressors, ordered by preference.
hive.server2 .thrift.resultset .max.fetch.size	1000	Max number of rows sent in one Fetch RPC call by the server to the client.
hive.resultset .compressor .snappy.class	org.apache .hive.resultset .compressor .snappy	Map the snappy compressor keyword to a Java class.
hive.resultset .compressor .<name>.class		Map the <name> compressor keyword to a Java class.
hive.resultset .compressor .<name>.<param>		A default option for <param> for the <name> compressor.

Table 1: Server configuration options

Parameter	Default	Description
compressors	snappy	A hiveConfs list of compressor keywords that the client can use.
compressor .<name> .<variable>		A hiveConfs to override the default compressor config.

Table 2: Client connection parameters

2.3 Client-Server Negotiation

CompDe negotiation proceeds as follows:

1. the client optionally notifies the server of available compressors by using 'hiveConfs' parameters in the connection string
   - a list of compressor keywords
   - parameters to override the server-configured compressor defaults for each plugin
2. the server creates a list of compressors from the intersection of the client and server configured plugins, ordered by the server's preference
3. for each plugin in the intersection
   1. the server merges the server-configured defaults with the client overrides into a HashSet
   2. the server passes the merged config to the 'init' function of the compressor
4. the server notifies the client of the keyword for the first CompDe that successfully initialized

All result sets will be compressed using that CompDe. The results will not be compressed if the client and server cannot agree on a compressor, if all of the available compressors fail to initialize with merged config, or if either the client or server has configured an empty compressor list. This negotiation scheme will maintain compatibility with old clients while using compression whenever it is available.

3 Implementation

3.1 Thrift Structures

TCLIService.thrift

```
// Represents a rowset
struct TRowSet {
  // The starting row offset of this rowset.
  1: required i64 startRowOffset
  2: required list<TRow> rows
  3: optional list<TColumn> columns
  4: optional binary binaryColumns
  5: optional i32 columnCount
}
```

TRowSet is unchanged by this enhancement. It is described here for completeness.

'startRowOffet' and 'rows' are deprecated in favor of 'columns' following HIVE-3746. HIVE-10438 added an option 'hive.server2.thrift.resultset-.serialize.in.tasks' to serialize columns (in batches, and in parallel) to 'binaryColumns' as contiguous TColumn. Following this enhancement, HIVE-13680, 'binaryColumns' will contain the output of the CompDes 'decompress' function when compression is enabled.

3.2 Compression

The output of a final node in a DAG is either a set of rows (from a map task) or a single value (from a reduce task). Following HIVE-12049 , the output is buffered row-by-row into TColumns in TRowSet and serialized in batches to a file in HDFS. The final output file is read by HiveServer2 and sent to the client.

Compressors will operate on the batch-level in ThriftJDBCBinarySerDe. If compression is enabled, 'compress' will be called on a ColumnBasedSet and the output will be stored in 'binaryColumns'. We have elected to give the CompDe access to the full column set rather than compressing column-by-column as this allows for optimizations such as grouping similar columns under the same compression algorithm.

3.3 Decompression

Results are serialized in batches. Consequently, the client must deserialize batch-by-batch in ColumnBasedSet. 'decompress' is called on 'binaryColumns' to get a ColumnBasedSet.