HIVE-13680: Provide a way to compress ResultSets

Kevin Liew

July 11, 2016

George Chow, Holman Lan, Rohit Dholakia, Thejas Nair, Vaibhav Gumashta, Ziyang Zhao

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 CompDe plugins. A default CompDe will process all data-types using the Snappy algorithm. Type-specific compression-decompression can be implemented by the CompDe. CompDes may also pass specific data-types through unprocessed.

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

```
@InterfaceAudience.Private
@InterfaceStability.Stable
public interface CompDe {
  public Map<String,String>
          init(Map<String,String> config);
  public byte[] compress(ColumnBuffer[] colSet);
  public ColumnBuffer[] decompress(byte[] compressedSet);
}
```

The ‘init‘ function is used to initialize a CompDe plugin with input parameters. Default parameters can be configured in the plugin and by the server and client. The CompDe returns the finalized configuration in a ‘Map’ if it can proceed using this merged configuration, otherwise ‘null’ on failed initialization.

Operating in the final task nodes, ‘compress’ takes a batch of rows contained in a ColumnBuffer array (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 an array of ColumnBuffer.

2.2 Configuration options

Option	Default	Description
hive.server2 .thrift.resultset .serialize.in.tasks	false	Whether we should serialize the Thrift structures used in JDBC ResultSet RPC in task nodes. We use SequenceFile and ThriftJDBCBinarySerDe to read and write the final results if this is true. (Required for compression)
hive.server2 .thrift.resultset .max.fetch.size	1000	Max number of rows sent in one Fetch RPC call by the server to the client. (Determines rows in a compressed batch)
hive.server2 .thrift.resultset .server.compressors	snappy	A list of names for available compressors, ordered by preference.
hive.server2 .thrift.resultset .compressor .snappy.class	org.apache .hive .resultset .compressor .snappy	Map the compressor identified by 'snappy' to a Java class.
hive.server2 .thrift.resultset .compressor .<name>.class		Map the compressor identified by <name> to a Java class.
hive.server2 .thrift.resultset .compressor .<name>.<param>		A default configuration for <param> for the compressor identified by <name>.

Table 1: Server CompDe configuration options

Parameter	Default	Description
hive.server2 .thrift.resultset .client.compressors	snappy	A list of names for compressors that the client can use.
hive.server2 .thrift.resultset .compressor .<name>.<param>		The client configuration for <param> for the compressor identified by <name>.

Table 2: Client 'hiveConf' 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 names
    - 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 Map
    2. the server passes the merged config to the 'init' function of the compressor
4. the server notifies the client of the name and configuration for the first CompDe that successfully initialized
5. the client tries to initialize this CompDe using the same parameters
    - if the CompDe fails to initialize on the client side, the client can restart the negotiation process with another CompDe list
    - if the CompDe initializes successfully, the client continues the session

All result sets will be compressed using the negotiated 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 Negotiation Thrift Structures

When a client connects to hiveserver2, it sends a TOpenSessionReq message to which the server responds with a TOpenSessionResp.

TCLIService.thrift

```
struct TOpenSessionReq {

  ...

  // Configuration overlay which is applied when the
  // session is first created.
  4: optional map<string, string> configuration
}

struct TOpenSessionResp {

  ...

  // The CompDe configuration settings for this session.
  4: optional map<string, string> compressorConfiguration
  // The name of the CompDe
  5: optional string compressorName
}
```

The client can send a configuration overlay to override the server's CompDe settings (and other settings in 'hive-site.xml'). To ensure symmetric compression-decompression, both the client and server must initialize the CompDe using the same parameters. However, the client is not aware of the default settings on the server-side, so we will return the server's CompDe configuration merged with the client overlay in TOpenSessionResp.

3.2 RowSet Thrift Structures

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

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
}
```

'startRowOffet' and 'rows' are deprecated in favor of 'columns' following HIVE-3746.

HIVE-10438 added 'hive.server2.thrift.resultset.serialize.in.tasks' to serialize columns (in batches, and in parallel on task nodes) to 'binaryColumns'

as contiguous TColumn. Following this enhancement, HIVE-13680, 'binaryColumns' of each Thrift message will contain one compressed batch when compression is enabled.

'columnCount' is used for deserializing TColumn from 'binaryColumns' when a result set is not compressed. A compressed result set must encode the number of columns in the binary output, as 'columnCount' is not available to the compressor.

3.3 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 ('serializeBatch') in ThriftJDBCBinarySerDe to output binary which is serialized to 'binaryColumns' of TRowSet via 'TCompactProtocol.writeBinary'. 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.

Currently, if the user has enabled 'hive.exec.compress.output', the setting 'mapred.output.compression.codec' (regardless of the execution engine) is checked by Hive's implementation of a Hadoop RecordWriter to determine the codec used to compress the final SequenceFile. However, this enhancement will have ThriftJDBCBinarySerDe compress data inside 'binaryColumns' which is serialized to a SequenceFile. Applying additional compression on 'binaryColumns' in the SequenceFile will have diminished marginal returns while wasting cpu cycles. Consequently, when a compressor plugin is configured, 'hive.exec.compress.output' will be disabled before serializing the final output. However, 'hive.exec.compress.intermediate' and associated options are interoperable with this enhancement.

3.4 Decompression

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