HIVE-13680: Provide a way to compress ResultSets

Kevin Liew

May 14, 2016

Abstract

Hive data pipelines invariably involve JDBC/ODBC drivers which access Hive through its Thrift interface. Consequently, any enhancement to the Hive's Thrift data pipeline 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 simply transfers these objects to a remote client. This parallel serialization strategy reduced latency in the data pipeline.

However, network load is often the most scarce resource in a system. As a further enhancement, this 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 rather than in the task nodes. 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 Interface

We will define a compressor interface which must be implemented by compressor plugins. A Snappy compressor will compress all data-types using

the Snappy algorithm. Type-specific compression can be achieved by implementing a compressor that delegates compression to other compressors based on a case-switch block.

2.2 Client-Server Negotiation

The client will specify a preferred compressor (and crc for the plugin's jar file) upon connection. If the server does not have that compressor plugin, it will request the jar from the client. The client should send the compressor plugin and all dependencies.

To prevent breaking compatibility with old clients, the server will not compress results unless the client has requested compression. However, new versions of beeline will try to negotiate a compressor scheme by default.

2.3 Server Security

We place the onus of security on the server administrator, recommending Kerberos authentication with user-impersonation to limit the jar file to the same permissions as the authenticated user. The server administrator also has the option to disable compressor plugins altogether.

2.4 Configuration options

Option	Default	Description
hive.resultSet.compressor.enabled	true	Enable or disable compressor negotiation

Table 1: Configuration options

3 Implementation

3.1 Negotiation

TBD

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 as a binary blob and sent to the client.

Compressors will operate on the batch-level in ThriftJDBCBinarySerDe and compressed batches will be serialized contiguously in the output file.

In the event that a column is not compressible, the column will be serialized as an uncompressed TColumn.

3.3 Decompression

Results are serialized in contiguous compressed batches. Consequently, the client must deserialize batch-by-batch.

TBD

3.4 Compressed RowSet Structures

3.4.1 HiveServer2 Structures

TBD

3.4.2 Thrift Objects

TEnColumn must now store a list of batch sizes to delimit the compressed blob and allow decompression of contiguous batches of rows within each column. Otherwise, the Thrift structures used to support compression are largely unchanged from HIVE-10438 . They are described here for completeness.

TBD

3.5 Compressor-Decompressor Interface

```
public interface ColumnCompDe {
    public byte[] compress(ColumnBuffer columns);
    public ColumnBuffer decompress(
        byte[] columnBlob, int startByte, int endByte);
    public boolean isCompressible(ColumnBuffer columns);
}
```

4 Custom Compressors

A default compressor will be provided for Snappy compression. To use other compression algorithms or to have type-specific compression, the user must implement the CompDe interface.

TBD