Design Doc: Hive Parquet Vectorization

Ferdinand Xu

@Intel

Contents

Background

Now Hive supports vectorization execution which processes a batch of data instead of a row. It's very efficient for vectorization execution in terms of CPU comparing with row based processing which yields high instruction counts and poor processor pipeline utilization [Boncz 2005]. The vectorization execution requires two parts of work: 1) the execution engine which allows HIVE to process a batch of data 2) the data format level reader which fetches data in batch to be processed by upper layer engine. However, HIVE only supports ORC to fetch data in batch which means that the vectorization execution is useful only when the underlying data format is ORC. Parquet is a columnar data format which is widely used for various SQL on Hadoop solutions. Bringing the performance benefits to Parquet format is very necessary.

Design

The major work for Hive Parquet Vectorization is to support vectorized Parquet reader. It will fetch data as a batch to feed the vectorization execution engine. For the upper layer engine, data need to be fetched in the following format.

```
public class VectorizedRowBatch implements Writable {
  public int numCols;           // number of columns
  public ColumnVector[] cols;   // a vector for each column
  public int size;              // number of rows that qualify (i.e. haven't been
filtered out)
  public int[] selected;        // array of positions of selected values
  ...
}
```

`VectorizedRowBatch` contains a set of `ColumnVector` for each column. For each `ColumnVector`, it contains a batch of data in the following format. The following is the example for long column:

```
public class LongColumnVector extends ColumnVector {
  public long[] vector;
  ...
}
```

To construct these data structure required by vectorization engine, we need to deal with two major problems: data type conversion and Parquet column reader.

Data type conversion

For Parquet, it supports a set of data types including `int, long, Boolean, double, binary` and `float`, while Hive has own type including `int, byte, short, date, interval_year_month, long, boolean, double, binary, string, char, varchar, float, decimal, interval_day_time` and `timestamp`. And the mapping is as follows:

Parquet Type	Hive type
`int`	`int, byte, short`
`long`	`date, interval_year_month, long`
`boolean`	`boolean`
`float`	`float`
`double`	`double`
`binary`	`binary, string, char, vchar`

Parquet reader

Parquet reader is used for fetching data from Parquet files. Its responsibility includes predicate pushing down, schema, and decoding data. And at this point, Hive uses a row based Parquet reader. Directly use existing row based Parquet reader will be CPU inefficiency since it has to do the Ser/De several times. There are two major solutions to implement the vectorized Parquet reader. The 1st solution is about implementing vectorized reader in Parquet project [Parquet Vectorized reader issue]. For this approach, there're two major CONs. It requires to convert all Parquet types to Hive types which will be inefficient to copy a batch of row data. Also not all Parquet type can be converted Hive type like Decimal. It's not able to convert this to decimal type. And the 2nd solution is about implementing vectorized Parquet reader in Hive side. To implement this, it needs to use page level reader to construct the batch needed for vectorization engine.

Implementation

Class hierarchy

As mentioned above, a `VectorizedRowBatch` batch consists of vectors for all needed columns. For the fetching stage, VectorizedParquetRecordReader will fetch `VectorizedRowBatch` with the helper of vectorized column level reader. And there're two kinds of vectorized Parquet reader at Column level. `VectorizedPrimitiveColumnReader` is used for primitive type and `VectorizedStructColumnReader` is for complex type. `VectorizedPrimitiveColumnReader` will read data into the column vector based on the type. It will decode the data for each data page based on its encoding format. And `VectorizedStructColumnReader` consists of a list of primitive type column reader. Both two column readers implements the interface `VectorizedColumnReader`.

```
public interface VectorizedColumnReader {


  /**
             * read records with specified size and type into the columnVector
             *
             * @param total      number of records to read into the column vector
             * @param column     column vector where the reader will read data into
             * @param columnType the type of column vector
             * @throws IOException
             */
  void readBatch(
    int total,
    ColumnVector column,
    TypeInfo columnType) throws IOException;
}
```

Encoding

For Parquet, it supports several encoding like dictionary, plaintext, run length encoding and so on [Parquet encoding]. To implement the column level reader, it's required to deal with the encoding. For the repetition level and definition level which used to define the null value and present nested value, vectorized column reader will use `RunLengthBitPackingHybridDecoder` to decode its value. For the value read, the specified encode will be decided based on the information provided by data page. If the page is a dictionary page, column reader will use dictionary decoder otherwise it will use the corresponding decoder when initializing the column reader.

Reference

[Boncz 2005] Peter Boncz et al., MonetDB/X100: Hyper-Pipelining Query Execution, Proceedings of the CIDR Conference, 2005.

[Parquet Vectorized reader issue] https://issues.apache.org/jira/browse/PARQUET-131

[Parquet encoding] https://github.com/apache/parquet-format/blob/master/Encodings.md