Vectorized Parquet APIs

Table of Contents

Problem

Parquet support row mode read/write currently.
We would like to support multiple rows mode now.

Target

- Parquet APIs can provide multiple rows at a time to SQL engine.
- Only read now.
- No copy from parquet original data to targeted SQL engine's row batch. Load and Generate at the same time.
- Decouple parquet vectorized data reading and targeted SQL engine's row batch data structure generating.

Basic Idea

Review row mode reading in parquet

1. Parquet internal readers have a method read() to read one row data at one time. Then the cursor in one Page, Column, or Row Group will move to the next row.
   These readers are listed here in an invocation sequence:
   ParquetRecordReader<T> => InternalParquetRecordReader<T> => RecordReader<T> => ColumnReader => PageReader => ValuesReader
2. SQL engine use ParquetInputFormat<T> to get a record reader and set the stereotype <T> with a concrete type (the type of the materialized records), which is returned by the record reader's next() method.
3. SQL engine also provides a customized subclass of Converter to transfer raw data to <T>.
4. In Parquet, ColumnReader.Binding binds the ValuesReader of one page with the customized Converter together. It means that Binding reads the data in storage layer through ValuesReader and delegate the data to Converter to generate <T>.
5. Finally, reader returns the converted <T> back to SQL engine.

Introduce vectorized row batch mode reading in parquet

1. Add a method readBatch() or readVector() in Parquet internal readers to read multiple rows data at one time. We need maintain the cursor of the reader and load new Page or Row Group if necessary.
2. SQL engine should has its own vectorized Converter to transfer multiple rows to a data structure <T> consumed by the engine.
3. ColumnReader.Binding binds the Converter with readers together to LOAD and TRANSFER at the same time.
4. Based on above 3 points, we keep the generic raw data reading iteration logic in Parquet readers. And specific SQL engine provides its Converter plugin to transfer raw data to its own data structure <T>.

A sample row batch data structure

A basic representation of vectorized batch of rows for the SQL engine usage could be like below. It might vary dramatically in different engines. (In Hive, it is VectorizedRowBatch. In Drill, it might be RecordBatch)
A RowBatch class represent multiple records. It has the number of records, and an array of each column ColumnVector.
A ColumnVector class represent one column. It has an array containing all the values in this column. The array size is RowBatch.batchsize. It has a set of subclasses representing vectors of different data type.

```
Class RowBatch {
    int batchsize;
    ColumnVector[] columns;
}
```

```
Class LongColumnVector extends ColumnVector{
    long[] vector;
}
```

Design

Proposed APIs and work flow

Vectorizedly read data from storage layer in Parquet

This part is described in the method invocation sequence of a read process.

1. ParquetRecordReader<T>, add a method
   public int readBatch(T next, int batchSize);
   - (T next) the expected returned data structure by SQL engine. The instance is materialized during reading procedure and used by the caller directly.
   - (int batchsize) the expected number of rows in the batch
   - (return int) the actual number of rows in materialized records. A possible optimization may change the size of the returned batch. This size info is also explicitly or implicitly contained in the instance 'next'.
   - It is invoked by a record read wrapper in the SQL engine code (like Hive). The parameter 'next' could be reused as one object.
2. InternalParquetRecordReader<T>, add a method
   public int readBatch(T next, int batchSize);
   - (T next), (int batchSize), (return int) is the same as above.
   - It is row group based iteration.
   - Compare the batchSize and the remaining rows number of the current row group. Iteratively Read the next row group into memory if necessary.
3. RecordReader<T>, add a method
   public int readBatch(T next, int size);
   - (T next), is the same as above.
   - (int size) the number of rows read from this row group.
   - (return int) the actual number of rows read from this row group
   - It iterates on each column and delegate each ColumnReader to read a number of rows.
4. ColumnReader, add a method

public int writeCurrentVectorToConverter(Object next, int size);

- (Object next) is the same object passed in from RecordReader. Since <T> does not apply for a single column, ColumnReader has no stereotype and use an Object type instead.
- (int size) the number of values read from this column.
- (return int) the actual number of values read from this column.
- It is page based iteration.
- Compare the size and the remaining rows number of current page. Read the next page if necessary.
- Invoke Binding.read() to get one value from current page. The value is stored in Binding instance.
- Invoke Binding.writeVector(next, columnId, rowId) to set the value to the Object 'next'. (Refer to the 5th item)
- Repeat above 3 steps, until expected size of rows are read.

5. ColumnReader.Binding, add a method
   public void writeVector(Object next, ColumnDescriptor columnId, int rowId);
   - (Object next) is the expected <T> type data for SQL engine.
   - (ColumnDescriptor columnId) indicate in which column the value should be set.
   - (int rowId) indicate the index where the value should be set in the column vector.
   - Binding should has a set of anonymous subclass to bind different Converter and ValuesReader for different primitive data type.
   - The actual value is stored in Binding instance, and it should be already read before invoking this method.
   - This method delegate the conversion to a concrete Converter, by invoke VectorizedPrimitiveConverter.addXxxVector(…). (Refer to the 6th item)
6. Add a new class VectorizedPrimitiveConverter extending Converter, with a set of methods for different primitive type.
   public void addLongVector(Object next, ColumnDescriptor columnId, int rowId, long value);
   public void addBinaryVector(Object next, ColumnDescriptor columnId, int rowId, Binary value);
   public void addBooleanVector(Object next, ColumnDescriptor columnId, int rowId, boolean value);
   ……
   - The first 3 parameter are the same as above.
   - (long value) is the raw data from storage layer and primitive.
   - The SQL engine code base should extend this class and override corresponding method based on the data type. (Refer to the 8th item)
7. ValuesReader
   No changes. It reads one value in a column at a time. The original set of subclasses of ValuesReader could be reused. The iteration happens in ColumnReader level.

Transfer raw data to SQL engine specific vectorized data type

8. XxxVectorizedPrimitiveConverter. (Not in Parquet project. In SQL engine project, like Hive.)
   These should be a set of new classes extending VectorizedPrimitiveConverter.

In the overridden method *addXxxVector(Object next, ColumnDescriptor columnId, int rowId, Xxx value);*

- Cast Object next to proper type <T>
- Set value to the position (rowId, columnId) in the instance of <T>
- Could also compute and set the specific attributes of type <T>, such as Boolean noNulls, Boolean isRepeating, etc.

Other consideration

1. Filter (TBD)
   - Pass filter to Parquet, so that it can do filtering first and then materialize vectors and row batch. This is for considering that Presto do lazy materialization.
   - Filter exists for the current parquet reader. I think we can do some improvements so that the filter can be applied on batch mode.

SQL engine plugin

Hive

1. Hive vectorized SQL engine wants the <T> type to be VectorizedRowBatch, which is composed of ColumnVector.
2. ColumnVector should has metadata like noNulls, isNull[], isRepeating. They can be computed and set during extracting data from storage layer and transferring data in Converter.
3. Add new wrapper classes VectorizedParquetInputFormat and VectorizedParquetRecordReader. The reader will invoke ParquetRecordReader.readBatch(…).
4. For conversion, Hive may add a class VectorizedRecordMaterializer, and a set of classes like VectorizedHiveGroupConverter and VectorizedPrimitiveConverter. These converters will transfer corresponding primitive values read from Parquet and fill them into VectorizedRowBatch.

Drill

I'm not familiar with Drill and not sure whether this design could work for Drill.
Zhenxiao, Jacques, what do you think?

1. In Drill, following classes might be used in its customized Converter. (Just a guess)
   - RecordBatch (the top container of vectors)
   - RecordBatchLoader (create the batch from DrillBuf)
   - DrillBuf (hold the real primitive data)