ORC Improvements for Apache Spark 2.2

1. Introduction

1.1 HDP 2.6.3 provides Apache Spark 2.2 with Apache ORC 1.4

Since Apache Spark 1.4.1, Spark supports ORC as one of its `FileFormat`. This article introduces how to use another faster ORC file format with Apache Spark 2.2 in HDP 2.6.3. First, in order to show how to choose a `FileFormat`, Section 1.2 will show an example which writes and reads with `ORCFileFormat`. Section 2 shows a brief performance comparison, and Section 3 explains more use cases and ORC configurations. Section 4 summarizes ORC-related Apache Spark fixes included in HDP 2.6.3.

1.2 Usage Example: Write and Read with ORCFileFormat

```
%spark2.spark
// Save 5 rows into an ORC file.
spark.range(5).write.format("orc").mode("overwrite").save("/tmp/orc")
// Read a DataFrame from the ORC file with the existing ORCFileFormat.
spark.read.format("orc").load("/tmp/orc").count
// Read a DataFrame from the ORC file with a new ORCFileFormat.
spark.read.format("org.apache.spark.sql.execution.datasources.orc").load("/tmp/orc").count

res4: Long = 5
res7: Long = 5
```

2. Performance Comparison

Here, I'll show you a small and quick performance comparison to show difference. For TPCDS 10TB performance comparison, please refer the presentation at DataWorks Summit in Reference section.

2.1 Prepare a test data (about 100m rows)

```
%spark2.spark
val df = spark.range(200000000).sample(true, 0.5)
df.write.format("orc").mode("overwrite").save("/tmp/orc_100m")

df: org.apache.spark.sql.Dataset[Long] = [id: bigint]
```

2.2 See the difference in 10 seconds

```
%spark2.spark
// New ORC file format
spark.time(spark.read.format("org.apache.spark.sql.execution.datasources.orc").load("/tmp/o
// Old ORC file format
spark.time(spark.read.format("orc").load("/tmp/orc_100m").count)

Time taken: 345 ms
res10: Long = 100000182
Time taken: 3518 ms
res12: Long = 100000182
```

3. How does it works?

3.1 Vectorization

New ORC file format in HDP 2.6.3, `org.apache.spark.sql.execution.datasources.orc`, is faster than old ORC file format. The performance difference comes from vectorization. Apache Spark has `ColumnarBatch` and Apache ORC has `RowBatch` separately. By combining these two vectorization techniques, we achieved the performance gain like the above. Previously, Apache Spark took advantages of its `ColumnarBatch` format with Apache Parquet only.

In addition, Apache Spark community has been putting efforts on SPARK-20901 Feature parity for ORC with Parquet. Recently, with new Apache ORC 1.4.1 (released 16th Oct), Spark becomes more stable and faster.

3.2 Do you want to use new ORC file format by default? Here is `spark.sql.orc.enabled` for that.

```
%spark2.spark
sql("SET spark.sql.orc.enabled=true")
spark.time(spark.read.format("orc").load("/tmp/orc_100m").count)
sql("SET spark.sql.orc.enabled=false")
spark.time(spark.read.format("orc").load("/tmp/orc_100m").count)

res13: org.apache.spark.sql.DataFrame = [key: string, value: string]
Time taken: 273 ms
res14: Long = 100000182
res16: org.apache.spark.sql.DataFrame = [key: string, value: string]
Time taken: 4083 ms
res17: Long = 100000182
```

3.3 Does it work with SQL, too? Yes, it does!

```
%spark2.spark
df.write.format("orc").mode("overwrite").saveAsTable("t1")
df.write.format("orc").mode("overwrite").saveAsTable("t2")
sql("SET spark.sql.orc.enabled=true")
spark.time(sql("SELECT COUNT(*) FROM t1").collect)
sql("SET spark.sql.orc.enabled=false")
spark.time(sql("SELECT COUNT(*) FROM t2").collect)

res21: org.apache.spark.sql.DataFrame = [key: string, value: string]
Time taken: 404 ms
res22: Array[org.apache.spark.sql.Row] = Array([100000182])
res24: org.apache.spark.sql.DataFrame = [key: string, value: string]
Time taken: 4333 ms
res25: Array[org.apache.spark.sql.Row] = Array([100000182])
```

3.4 How can I create a table using new ORC file format only?

```
%spark2.spark
sql("DROP TABLE IF EXISTS o1")
sql("CREATE TABLE o1 USING `org.apache.spark.sql.execution.datasources.orc` AS SELECT * FRO
sql("SET spark.sql.orc.enabled=false")
spark.time(sql("SELECT COUNT(*) FROM o1").collect)

res26: org.apache.spark.sql.DataFrame = []
res27: org.apache.spark.sql.DataFrame = []
res28: org.apache.spark.sql.DataFrame = [key: string, value: string]
Time taken: 213 ms
res29: Array[org.apache.spark.sql.Row] = Array([100000182])
```

3.5 Do you want to read existing Hive tables created by `STORED AS`? Here is `spark.sql.hive.convertMetastoreOrc`

```
%spark2.spark
```

```
sql("DROP TABLE IF EXISTS h1")
sql("CREATE TABLE h1 STORED AS ORC AS SELECT * FROM t1")
sql("SET spark.sql.hive.convertMetastoreOrc=true")
sql("SET spark.sql.orc.enabled=true")
spark.time(sql("SELECT COUNT(*) FROM h1").collect)

res30: org.apache.spark.sql.DataFrame = []
res31: org.apache.spark.sql.DataFrame = []
res33: org.apache.spark.sql.DataFrame = [key: string, value: string]
res34: org.apache.spark.sql.DataFrame = [key: string, value: string]
Time taken: 227 ms
res35: Array[org.apache.spark.sql.Row] = Array([100000182])
```

3.6 ORC Configuration

To utilize new ORC file format, there are more ORC configurations which you should turn on.
The followings are the summary of recommended ORC configurations in HDP 2.6.3 and the above.

1. `spark.sql.orc.enabled=true` enables new ORC format to read/write DataSource Tables and files.
2. `spark.sql.hive.convertMetastoreOrc=true` enables new ORC format to read/write Hive Tables.
3. `spark.sql.orc.filterPushdown=true` enables filter pushdown for ORC formats.
4. `spark.sql.orc.char.enabled=true` enables new ORC format to use CHAR types to read Hive Tables.
   By default, STRING types are used for performance reason.

4. More features and limitations

4.1 Fixed Apache Spark issues

- SPARK-14387 Enable Hive-1.x ORC compatibility with spark.sql.hive.convertMetastoreOrc
- SPARK-16060 Vectorized Orc Reader
- SPARK-16628 OrcConversions should not convert an ORC table represented by MetastoreRelation to HadoopFsRelation if metastore schema does not match schema stored in ORC files
- SPARK-18355 Spark SQL fails to read data from a ORC hive table that has a new column added to it
- SPARK-19809 NullPointerException on empty ORC file
- SPARK-20682 Support a new faster ORC data source based on Apache ORC
- SPARK-20728 Make ORCFileFormat configurable between sql/hive and sql/core
- SPARK-21422 Depend on Apache ORC 1.4.0
- SPARK-21477 Mark LocalTableScanExec’s input data transient
- SPARK-21791 ORC should support column names with dot
- SPARK-21787 Support for pushing down filters for date types in ORC
- SPARK-21831 Remove `spark.sql.hive.convertMetastoreOrc` config in HiveCompatibilitySuite
- SPARK-21912 ORC/Parquet table should not create invalid column names
- SPARK-21929 Support `ALTER TABLE table_name ADD COLUMNS(..)` for ORC data source
- SPARK-22146 FileNotFoundException while reading ORC files containing special characters
- SPARK-22158 convertMetastore should not ignore table property
- SPARK-22300 Update ORC to 1.4.1

4.2 Limitation

- Schema evolution and schema merging are not supported officially yet (SPARK-11412).
- Apache Spark vectorization techniques can be used with a schema with primitive types. For more complex schema, Spark uses non-vectorized reader.
- Old ORC files may be incorrect information inside TIMESTAMP. Filter Pushdown will be ignored for those old ORC files.

5. Conclusion

HDP 2.6.3 provides a powerful combination of Apache Spark 2.2 and Apache ORC 1.4.1 as a technical preview.
In Apache Spark community, SPARK-20901 Feature parity for ORC with Parquet is still on-going efforts.
We are looking forward to seeing more improvements in Apache Spark 2.3.

Reference

- ZEPPELIN NOTEBOOK for this article.
- PERFORMANCE UPDATES: WHEN APACHE ORC MET APACHE SPARK, DataWorks Summit 2017 Sydney, Sep. 20-21

Appendix - How to reset to default options

%spark2.spark

sql("SET spark.sql.hive.convertMetastoreOrc=false") sql("SET spark.sql.orc.enabled=false") sql("SET spark.sql.orc.filterPushdown=false") sql("SET spark.sql.orc.char.enabled=false")

```
res36: org.apache.spark.sql.DataFrame = [key: string, value: string]
res37: org.apache.spark.sql.DataFrame = [key: string, value: string]
res38: org.apache.spark.sql.DataFrame = [key: string, value: string]
res39: org.apache.spark.sql.DataFrame = [key: string, value: string]
```