ORC Format Support

Revision History

Name	Date	Reason For Changes	Version	Status
Ming Li, Hong Wu, Paul Guo, Lili Ma, Lei Chang	2016.06.17	Initial version	v0.1	Draft

1. Introduction

HAWQ is an important component in Hadoop eco-system which is seamlessly integrated with major components. For example, it already uses YARN to manage resource. It also uses HCatalog to import table meta info from hive catalog service.

ORC (Optimized Row Columnar) is a very popular open source format adopted in some major components in Hadoop eco-system. It is also used by a lot of users. The advantages of supporting ORC storage in HAWQ are in two folds: firstly, it makes HAWQ more Hadoop native which interacts with other components more easily; secondly, ORC stores some meta info for query optimization, thus, it might potentially outperform two native formats (i.e., AO, Parquet) if it is available.

Since there are lots of popular formats available in HDFS community, and more advanced formats are emerging frequently. It is good option for HAWQ to design a general framework that supports pluggable c/c++ formats such as ORC, as well as native format such as AO and Parquet. In designing this framework, we also need to support data stored in different file systems: HDFS, local disk, amazon S3, etc. Thus, it is better to offer a framework to support pluggable formats and pluggable file systems.

FDW is a SQL standard (SQL/MED) which has been implemented by a lot of database systems, for example, postgresql. And it can support similar functionalities. So we propose to follow the standard FDW interface to implements the pluggable format and pluggable file system functionalities.

2. Requirement

- Support ORC file format Read/Write
- A pluggable framework for formats and file systems
- Comparable performance with other native formats

3. User Experience

Here we use hdfs FDW as an example.

3.1 For end users

3.1.1 Internal table example

```
// create an internal table
CREATE TABLE internal_table (
  id serial NOT NULL,
  data text)
with(format = ‘orc’);

INSERT INTO internal_table VALUES(123, 'hello world');
SELECT * FROM internal_table;
```

3.1.2 External table example

```
// create a foreign server
CREATE SERVER hdfs_server
FOREIGN DATA WRAPPER hdfs_fdw
OPTIONS(url = hdfs://localhost:8020/);

// create a external/foreign table
CREATE EXTERNAL TABLE foreign_table (
  id serial NOT NULL,
  data text)
SERVER hdfs_server
OPTIONS (
    location '/ext/foreign_table',
    format 'orc',
    ...);

INSERT INTO foreign_table VALUES(123, 'hello world');
SELECT * FROM foreign_table;
```

3.2 For FDW developers

```
// create a fdw handler which implements the fdw functionalities
Datum hdfs_fdw_handler(PG_FUNCTION_ARGS) {
```

```
        // Most API are same with postgreSQL fdw api
        fdwroutine->GetForeignRelSize = hdfsGetForeignRelSize;
        fdwroutine->GetForeignPaths = hdfsGetForeignPaths;
        fdwroutine->GetForeignPlan = hdfsGetForeignPlan;
        fdwroutine->ExplainForeignScan = hdfsExplainForeignScan;
        fdwroutine->mapSegFile = hdfsMapSegFile;
        fdwroutine->CreateSplits = hdfsCreateSplits;
        …
}

// Create a fdw validator
Datum hdfs_fdw_validator(PG_FUNCTION_AGRS) {
        PG_RETURN_VOID()
}

// Create the UDF for the fdw handler
CREATE FUNCTION hdfs_fdw_handler()
RETURNS fdw_handler
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;

// Create the UDF for the fdw validator
CREATE FUNCTION hdfs_fdw_validator(text[], oid)
RETURNS void
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;

// Create the foreign data wrapper
CREATE FOREIGN DATA WRAPPER dfs_fdw
  HANDLER hdfs_fdw_handler
  VALIDATOR hdfs_fdw_validator;


// create a file system handler which implements the fdw functionalities
Datum hdfs_filesystem_handler(PG_FUNCTION_ARGS) {
        routine->gpfs_connect = hdfs_connect
        routine->gpfs_disconnect = hdfs_disconnect,
        routine->gpfs_open = hdfs_open,
```

```
        ...
}
```

// create hdfs file system

```
CREATE FUNCTION hdfs_filesystem_handler()
RETURNS filesystem_handler
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

// Create a file system in HAWQ

CREATE FILESYSTEM hdfs

```
HANDLER hdfs_filesystem_handler
```

```
// create a orc format handler
Datum orc_format_handler(PG_FUNCTION_ARGS) {
    routine->dfsGetForeignRelSize = orcGetForeignRelSize
    routine->dfsGetForeignPaths = orcGetForeignPaths
        ...
}
```

// create a orc format UDF

```
CREATE FUNCTION orc_format_handler()
RETURNS format_handler
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

```
// create ORC format
CREATE FORMAT ORC
HANDLER orc_format_handler;
```

4. Future works

- Support Error tables
- Support Partition tables
- Support Hash distributed tables
- AO/Parquet use pluggable framework

5. References

1) PostgreSQL FDW links:
http://www.postgresql.org/docs/9.5/static/fdw-callbacks.html
https://wiki.postgresql.org/wiki/Foreign_data_wrappers

2) OCR format:
http://www.semantikoz.com/blog/orc-intelligent-big-data-file-format-hadoop-hive/
https://orc.apache.org/