Big Data Technologies

Agenda

- Apache Hive

Apache Hive

File formats

- Hive data is stored in HDFS in supported file formats.
- The file formats decide data processing in generated MR job.
- Big data file formats are specialized storage formats designed for efficient storage, querying, and processing of large-scale datasets. These formats are optimized for use in distributed systems like Hadoop, Spark, and data warehouses. Each format has unique characteristics and is suited for specific use cases. Here's an overview of popular file formats like **RCFile**, **ORC**, **Avro**, **Parquet**, **Sequence**, **Text**, **CSV**, **JSON**, etc.

RCFile (Record/Row Columnar File)

RCFile is a columnar storage file format primarily used in Hadoop. It splits the data into row groups, stores column data separately within each group, and compresses the data for efficient storage and processing.

Features:

- Combines row and columnar storage approaches.
- Good for data warehousing applications.
- Efficient for analytical queries requiring specific columns.

Limitations:

- Slower compared to newer formats like ORC or Parquet.
- No support for advanced schema evolution.

Use Case:

- Legacy systems using Hive for analytical workloads.

ORC (Optimized Row Columnar)

ORC is a columnar storage format designed for efficient storage and querying in Hadoop ecosystems, particularly Hive. It is optimized for read-heavy operations.

Features:

- Highly efficient columnar storage format.
- Built-in indexing and metadata for faster querying.
- Compression support (e.g., ZLIB, Snappy).
- Schema evolution support.

Advantages:

- Smaller file sizes compared to RCFile.
- Fast and efficient for analytical workloads due to built-in indexes.

Limitations:

- Less efficient for small data files or write-heavy operations.

Use Case:

- Data warehousing and OLAP workloads.
- Hive and Spark SQL queries with large datasets.

Avro

Avro is a row-based file format used for data serialization and deserialization in Hadoop ecosystems. It stores data along with its schema in a compact binary format.

Features:

- Schema evolution support (e.g., adding new fields).
- Row-based format for faster writes and efficient streaming.
- Language-independent (supports multiple programming languages).

Advantages:

- Lightweight and compact.
- Ideal for data exchange and streaming pipelines.

Limitations:

- Less efficient for analytical workloads compared to columnar formats.

Use Case:

- Real-time data processing (e.g., Kafka, Flume).
- Streaming pipelines and ETL jobs.

Parquet

Parquet is a columnar storage format optimized for analytical workloads. It is widely used in distributed systems like Spark and Hive.

Features:

- Columnar storage with efficient compression and encoding.
- Schema evolution support.
- Optimized for selective queries.

Advantages:

- Highly efficient for read-heavy analytical workloads.
- Small file sizes due to advanced compression techniques.

Limitations:

- Slower writes compared to row-based formats like Avro.

Use Case:

- Data lakes and warehouses.
- Analytical workloads in Hive, Spark, and Presto.

Sequence File

A flat-file format in Hadoop designed for key-value pair storage.

Features:

- Splittable for parallel processing.
- Supports compression at record or block level.

Advantages:

- Simple and efficient for sequential access.

Limitations:

- Not optimized for complex analytical queries.

Use Case:

- Intermediate storage of key-value data in MapReduce jobs.

JSON (JavaScript Object Notation)

A human-readable format used for semi-structured data.

Features:

- Self-describing and language-independent.
- Compatible with most tools and programming languages.

Advantages:

- Easy to debug and manipulate.

Limitations:

- Larger file size compared to binary formats.
- Slower processing for large datasets.

Use Case:

- Lightweight data exchange and REST APIs.

CSV (Comma-Separated Values)

Description:

- A simple and widely used format for tabular data storage.

Features:

- Easy to create and manipulate.
- Supported by almost all tools and databases.

Advantages:

- Human-readable and widely compatible.

Limitations:

- No support for schema or data types.
- Poor performance for large datasets.

Use Case:

- Data ingestion and export.
- Small-scale datasets.

Comparison of Key File Formats:

Format	Type	Compression	Schema Evolution	Optimized For
RCFile	Columnar	Yes	No	Analytical workloads
ORC	Columnar	Yes (efficient)	Yes	Analytical queries
Avro	Row-based	Yes	Yes	Data exchange, streaming
Parquet	Columnar	Yes (efficient)	Yes	Analytical queries
Sequence File	Key-Value	Yes	No	Sequential access
JSON	Semi-structured	Limited	No	Semi-structured data
CSV	Tabular	No	No	Simple, small datasets

How to Choose the Right Format:

- **Avro**: For row-based processing, streaming, or scenarios requiring schema evolution.
- **ORC/Parquet**: For columnar storage and analytical queries on large datasets.
- **JSON/CSV**: For simple or human-readable data exchanges.
- **Sequence File**: For key-value pairs in Hadoop MapReduce jobs.

Would you like a deeper dive into a specific format or examples of using these formats in a big data framework?

Hive SerDe

- Serde is Serializer & Deserializer.
- Hive uses the SerDe interface for IO.
- Internally encapusulate Hadoop InputFormat (& RecordReader) and OutputFormat (& RecordWriter).
- Types: Built-in Serdes, Third party Serdes, Custom Serdes
- Built-in SerDes
    - Avro (Hive 0.9.1 and later)

- ORC (Hive 0.11 and later)
- RegEx
- Thrift
- Parquet (Hive 0.13 and later)
- CSV (Hive 0.14 and later)
- JsonSerDe (Hive 0.12 and later in hcatalog-core)

OpenCSVSerde

- Loads CSV file into hive table
- Comma separated file
- If data contains comma, cell is enclosed in double quote.
- If data contains double quotes, it is escaped by "".

```
CREATE TABLE movies_staging(
id INT,
title STRING,
genres STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
'separatorChar' = ',',
'quoteChar'     = '\"',
'escapeChar'    = '\\'
)
TBLPROPERTIES('skip.header.line.count'='1');
```

RegexSerde

- Only Deserializer i.e. only used to read records.
- Mainly used for data cleansing/extraction.

```
CREATE TABLE ncdc_staging(
yr SMALLINT,
temp INT,
quality TINYINT
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES(
'input.regex'='^.{15}([0-9]{4}).{68}([-\+][0-9]{4})([0-9]{1}).*$'
);
```

DML operations in Hive

- Hive is designed for OLAP (Analysis).
- However later versions (0.14+) added support for Update and Delete operation.
- DML operations are carried out by MR job and hence need considerable time.
- DML is possible in Hive tables, if following conditions are fulfilled.
    - In hive-site.xml, set transaction manager.
    - Table must be in ORC format.
    - Table should be bucketed (mandetory in Hive 2.x and recommended in Hive 3.x).
    - Table must be transactional.
- Hive data is stored in HDFS. HDFS is "write once read multiple times" type of file system.
- Delete and update operations doesn't (cannot) change the files that are already uploaded.
- Internally, these commands create new files (delta) to keep information about deleted/updated records.
- When SELECT operation is performed next time, it will get the records from the main data file + changes recorded in these delta files. The effective data will be presented to the user.
- If number of delta files are many, the performance of SELECT operation will hamper.
- Eventually, these delta files are to be merged in main data file to produce new updated data file. The other small delta files can be deleted. This process is called as "compaction".
- Transactions (DML) performed on the table can viewed with command -- SHOW TRANSACTIONS;

Hive Compaction

- Minor compaction: Smaller changes files (delta) are mereged together to reduce the number of delta files. This will save space in HDFS and also speed-up further queries. In minor compaction, data in main file is not modified (e.g. older records are still present there). Minor compaction process takes relatively less time.

```
ALTER TABLE ncdc COMPACT 'minor';
```

- Major compaction: All delta files and main data files are mereged to created a new data file. All older versions (delta files and older main file) are deleted. This will significantly improve speed to further queries. Major compaction needs more time.

```
ALTER TABLE ncdc COMPACT 'major';
```

Hive Views

- View of main data-table.
- Hive views are same as RDBMS views.
  - Simple view -- can also perform DML (in MySQL)
  - Complex view -- cannot perform DML
  - Inline view -- SELECT in projection
    - SELECT empno, ename, sal, (SELECT SUM(sal) FROM emp) total FROM emp;
  - Materialized view -- store result of view query -- Not supported in MySQL.
- Applications
  - Simplify few queries.
  - Hides table creation/data.
  - Security.
- Hive view create query is stored in Hive metastore. But no data/result of view is stored in HDFS.
- Hive 2.x views are not materialized.
- Hive 3.x added materialized views.

Hive Materialzed Views

- Materialzed views are added in Hive 3.x.
- Materialzed views can be created only on transactional tables.
- Hive Materialized view create query is stored in Hive metastore and data/result of view query is stored in HDFS.

```
CREATE MATERIALIZED VIEW viewname AS
SELECT ... FROM tables ...;
```

- When any query executed on materialized view, it is executed faster (no need to recalculate data from underlying table).
- If any changes are done in underlying, they are not auto reflected in materialized view. However, view can be rebuild i.e. view query can re-execute and old calculated results overwritten (in hdfs).

```
ALTER MATERIALIZED VIEW viewname REBUILD;
```