Hadoop File Formats

A storage format refers to how the data is stored in the underlying file system. Hadoop storage formats specifically refer to how the data is stored in the HDFS file system.

Why do we need to consider storage formats?

1. **Storage Cost**
    - When the data is processed using Bigdata frameworks such as Hadoop, Spark etc. the data is usually stored in fully denormalized format.
    - In HDFS, each file is stored with a specific replication-factor.
    - These factors would increase the cost of storing huge amount of data.

2. **Processing Cost**
    - As the data is big, processing cost also is more as the data comes into CPU, RAM, and Network IO etc.

So to minimize these costs it is important to choose a proper file format that optimizes these costs and at the same time provide use-case specific advantages such as high read & write speeds, splittability, compression, schema-evolution etc.

Characteristics of File Formats

- Read performance
- Write Performance
- Splittability – ability to be splittable and stored as blocks on HDFS, so as to facilitate distributed processing.
- Compression – support for block-level compression
- Schema Evolution (Support for dynamic schema)

Compression in Storage Formats

Two types of compression can be used in Hadoop

- **File-level compression**:
  - Refers to compressing the entire file regardless of the format
  - Here we can only chose those compression codecs that allow splitting of the compressed files. Otherwise, you can not store the compressed file as blocks on HDFS.

- **Block-level compression**:
  - Is internal to a file format.
  - Here individual blocks of the file are compressed using an appropriate compression codec.
  - Gives better compression ratio.
  - Here, we can use any compression-code, even if the codec itself is not-splittable, as we are applying the compression on already splitted file i.e. on blocks of the file.

Some of the common compression codecs are:

- **Snappy (non-splittable)**
- LZO (splittable if indexed)
- Bzip2 (splittable)

Columnar File Formats

- Columnar file formats are gaining a lot of popularity on Hadoop because of their inherent advantages.

- In columnar file formats, instead of just storing rows of data adjacent to one another, you also store column values adjacent to each other. So datasets are partitioned both horizontally and vertically.

- This is particularly useful if your data processing framework just needs access to a subset of data that is stored on disk as it can access all values of a single column very quickly without reading whole records.

Parquet

- **Parquet file format is also a columnar format.**
    - Instead of just storing rows of data adjacent to one another you also store column values adjacent to each other. So datasets are partitioned both horizontally and vertically.

    - This is particularly useful if the data processing framework just needs access to a subset of data that is stored on disk as it can access all values of a single column very quickly without reading whole records.

- **Provides very good read performance**
    - It reduces a lot of I/O cost to make great read performance esp. if you read only specific columns (as you would normally do in Spark & Hive) as against reading entire rows (as in MR).

    - If you're chopping and cutting up datasets regularly then these formats can be very beneficial to the speed of your application.

- **Provides moderate write performance**
    - Parquet format is computationally intensive on the write side.

- **Provides very good compression-ratio and supports both File-level and Block-level compression**
    - Data in the same column tends to be compressed together which can yield some massive storage optimizations, as data in the same column tends to be similar.

- **Supports schema evolution to some extent (better than ORC)**
    - You can add new columns to the end of the structure.
    - Schema segregated into footer.
    - Due to merging of schema across multiple files, schema evolution is expensive.

- **Works very well with Spark as there is a Vectorized reader in Spark for reading Parquet files.**

AVRO

- Row oriented storage format.
- Apache Avro is a **language-neutral data serialization and de-serialization framework**.
- **Good support for schema evolution** and faster serialization & deserialization
    - Avro is a preferred tool to serialize data in Hadoop.
    - Avro format encodes the schema of its contents directly in the file (defines file data schemas in JSON for interoperability) which allows you to store complex objects natively.
    - It allows every data to be written with no prior knowledge of the schema. It serializes fast and the resulting serialized data is lesser in size.
- **Good support for compression**
    - Supports both File-level and Block-level compression
- **Ideal for write-heavy data operations**.
- **Good read performance for entire row consumption** and processing (as in MR)

ORC

- **Column oriented storage format**
- **Limited schema evolution support**
    - Schema is stored in the footer of the file.
    - Can add new columns at the end of the file (cant delete or rename columns)
- **Very good read performance**
    - Works well with selected column data consumption & processing.
- **Works well with Hive** (Hive has a Vectorized reader for ORC)

	AVRO	Parquet	ORC
Schema Evolution	Most efficient, as the schema is stores as JSON with the file.	Schema evolution is expensive as it needs to be read and merged across all the parquet files	Schema evolution here is limited to adding new columns and a few cases of column type-widening.
Compression	Less efficient	Best with Snappy	Best with ZLIB
Splitability	Partially	Partially	Fully
Row/Column Oriented	Row	Column	Column
Read/Write	Write-Heavy	Read-Heavy	Read-Heavy
Optimized Processing Engines	Kafka	Spark(Cloudera)	Hive, Presto (Horton Works)