parquet vs avro vs orc

Data format

- **Storage Format**
  - Text
  - Sequence File
  - Avro
  - Parquet
  - Optimized Row Columnar (ORC)
- RCFILE
- ORCFILE

Text

- More specifically text = csv, tsv, json records…
- Convenient format to use to exchange with other applications or scripts that produce or read delimited files
- Human readable and parsable
- Data stores is bulky and not as efficient to query
- Do not support block compression

Sequence File

- Provides a persistent data structure for binary key-value pairs
- Row based
- Commonly used to transfer data between Map Reduce jobs
- Can be used as an archive to pack small files in Hadoop
- Support splitting even when the data is compressed

https://acadgild.com/blog/file-formats-in-apache-hive/

We know that Hadoop's performance is drawn out when we work with small number of files with big size rather than large number of files with small size. If the size of a file is smaller than the typical block size in Hadoop, we consider it as a small file. Due to this, the amount of metadata increases which will become an overhead to the NameNode. To solve this problem sequence files are introduced in Hadoop. Sequence files acts as a container to store the small files.

Sequence files are flat files consisting of binary key-value pairs. When Hive converts queries to MapReduce jobs, it decides on the appropriate key-value pairs to be used for a given record. Sequence files are in binary format which are able to split and the main use of these files is to club two or more smaller files and make them as a one sequence file.

In Hive we can create a sequence file by specifying STORED AS SEQUENCEFILE in the end of a CREATE TABLE statement.

There are three types of sequence files:

• Uncompressed key/value records.

• Record compressed key/value records – only 'values' are compressed here

• Block compressed key/value records – both keys and values are collected in 'blocks' separately and compressed. The size of the 'block' is configurable.

Hive has its own SEQUENCEFILE reader and SEQUENCEFILE writer for reading and writing through sequence files.

Avro

- Widely used as a serialization platform
- Row-based, offers a compact and fast binary format
- Schema is encoded on the file so the data can be untagged
- Files support block compression and are splittable
- Supports schema evolution

Avro is one of the preferred data serialization system because of its language neutrality.

Avro is also very much preferred for serializing the data in Hadoop.

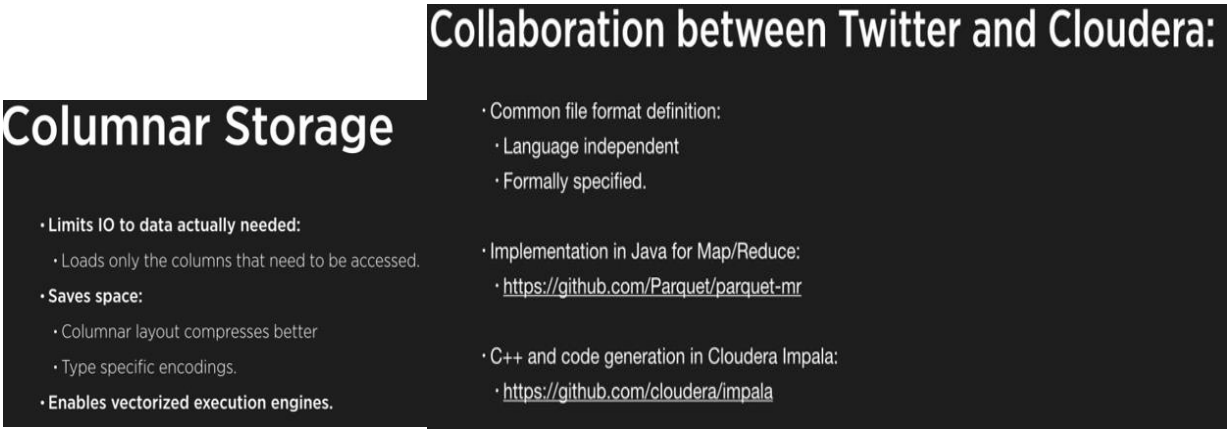
It uses JSON for defining data types and protocols,and serializes data in a compact binary format. Its primary use is in Apache Hadoop, where it can provide both a serialization format for persistent data, and a wire format for communication between Hadoop nodes, and from client programs to the Hadoop services.

By this we can define Avro as a file format introduced with Hadoop to store data in a predefined format.This file format can be used in any of the Hadoop's tools like Pig and Hive.

Parquet

- Column-oriented binary file format
- Uses the record shredding and assembly algorithm described in the Dremel paper
- Each data file contains the values for a set of rows
- Efficient in terms of disk I/O when specific columns need to be queried

https://blog.twitter.com/2013/dremel-made-simple-with-parquet



- Binary format in Lzo, compressing per column.

- When we scan particular column, there will be a lot more performance gain.

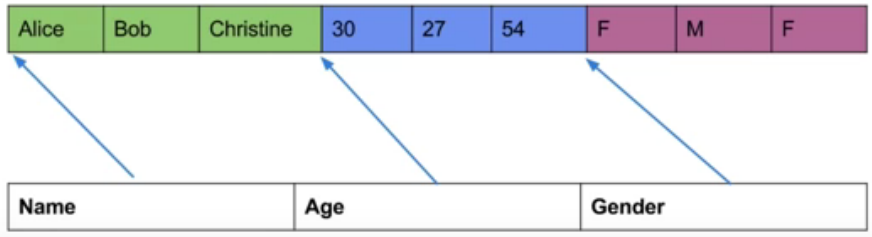
Optimized Row Columnar

- Considered the evolution of the RCFile
- Stores collections of rows and within the collection the row data is stored in columnar format
- Introduces a lightweight indexing that enables skipping of irrelevant blocks of rows
- Splittable: allows parallel processing of row collections
- It comes with basic statistics on columns (min ,max, sum, and count)

How to choose: For read

- **Use case**
  - Avro – Query datasets that have changed over time
  - Parquet – Query a few columns on a wide table

Name	Age	Gender
Alice	30	F
Bob	27	M
Christine	54	F



Benefits

- Only read columns you need
- Like data together -- better compression
- Type-specific encodings possible
- Skip unnecessary deserialization
- Possible to operate on encoded data
- Natural fit for vectorized operations

To illustrate what columnar storage is all about, here is an example with three columns.

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

In a row-oriented storage, the data is laid out one row at a time as follows:

A1	B1	C1	A2	B2	C2	A3	B3	C3

Whereas in a column-oriented storage, it is laid out one column at a time:

A1	A2	A3	B1	B2	B3	C1	C2	C3

There are several advantages to columnar formats.

- Organizing by column allows for better compression, as data is more homogenous. The space savings are very noticeable at the scale of a Hadoop cluster.
- I/O will be reduced as we can efficiently scan only a subset of the columns while reading the data. Better compression also reduces the bandwidth required to read the input.
- As we store data of the same type in each column, we can use encodings better suited to the modern processors' pipeline by making instruction branching more predictable.

https://acadgild.com/blog/hive-beginners-guide/

https://acadgild.com/blog/integrating-hive-with-hbase/