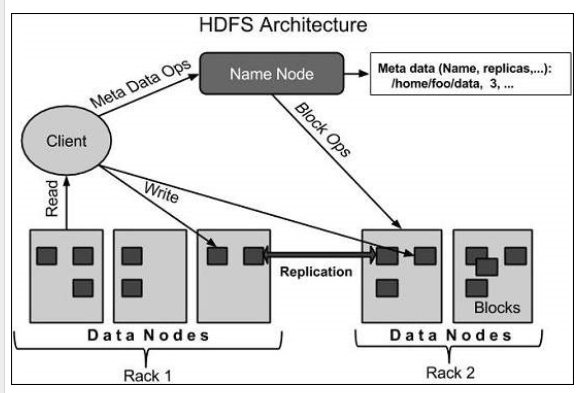
1. **What is HDFS (Hadoop Distributed File System)?**

Distributed file system part of Hadoop framework, designed to be highly fault tolerant and run on commodity hardware.

1. **What are the features of HDFS?**
2. HDFS helps for the distributed storage and processing.
3. Hadoop provides a command interface to interact with HDFS.
4. The built-in servers of namenode and datanode help users to easily check the status of cluster.
5. Streaming access to file system data.
6. HDFS provides file permissions and authentication.
7. **HDFS architecture? Imp**



HDFS follows the master-slave architecture and it has the following elements.

**Name Node:**

The name node is the commodity hardware that contains the GNU/Linux operating system and the name node software. The system having the name node acts as the master server and it does the following tasks

* Manages the file system metadata (name space).
* Regulates client’s access to files.
* It also executes file system operations such as renaming, closing file and opening files and directories.

**Data Node:**

The datanode is a commodity hardware having the GNU/Linux operating system and data node   
Data node does the following operation.

* Manage the data storage of their system.
* They also perform operations such as block creation, deletion, and replication according to the instructions of the name node.
* Data nodes perform read-write operations on the file systems, as per client request.

**Secondary name node:**

Secondary name node constantly read file system metadata of the from the name node and writes it to hard disk or the file system.

So it is not substitute for name node. Name node is SPOF (Single point of failure) if it goes down entire Hadoop system goes down

1. **Is Secondary name node replacement for name node?**

No. Name node is SPOF in Hadoop

1. **What is Block? IMP**

The amount of data that HDFS can read or write is called a Block.

The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

1. **What are the goals of HDFS?**

**Fault detection and recovery** − since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

**Huge datasets** − HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.

**Hardware at data** − A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

1. **HDFS Operations?**
2. ***Starting HDFS:***

Initially you have to format the configured HDFS file system, open namenode (HDFS server), and execute the following command.

*$ hadoop namenode -format*

After formatting the HDFS, start the distributed file system. The following command will start the namenode as well as the data nodes as cluster.

*$ start-dfs.sh*

1. ***Listing Files in HDFS***

After loading the information in the server, we can find the list of files in a directory, status of a file, using ‘ls’. Given below is the syntax of ls that you can pass to a directory or a filename as an argument.

*$ $HADOOP\_HOME/bin/hadoop fs -ls <args>*

1. ***Inserting Data into HDFS***

Assume we have data in the file called file.txt in the local system which is ought to be saved in the hdfs file system. Follow the steps given below to insert the required file in the Hadoop file system.

Step 1

You have to create an input directory.

*$ $HADOOP\_HOME/bin/hadoop fs -mkdir /user/input*

Step 2

Transfer and store a data file from local systems to the Hadoop file system using the put command.

*$ $HADOOP\_HOME/bin/hadoop fs -put /home/file.txt /user/input*

Step 3

You can verify the file using ls command.

*$ $HADOOP\_HOME/bin/hadoop fs -ls /user/input*

1. ***Retrieving Data from HDFS***

Assume we have a file in HDFS called outfile. Given below is a simple demonstration for retrieving the required file from the Hadoop file system.

Step 1

Initially, view the data from HDFS using cat command.

$ $HADOOP\_HOME/bin/hadoop fs -cat /user/output/outfile

Step 2

Get the file from HDFS to the local file system using get command.

$ $HADOOP\_HOME/bin/hadoop fs -get /user/output/ /home/hadoop\_tp

1. ***Shutting Down the HDFS***

You can shut down the HDFS by using the following command.

$ stop-dfs.sh

1. **HDFS Command Reference?**

<https://www.tutorialspoint.com/hadoop/hadoop_command_reference.htm>

1. **What the Input file formats supported in Hadoop/HDFS?**
2. Text/CSV Files

Text and CSV files are quite common and frequently Hadoop developers and data scientists received text and CSV files to work upon.

However, *CSV files do not support block compression,* thus compressing a CSV file in Hadoop often comes at a significant read performance cost.

1. JSON Records

JSON records contain JSON files where each line is its own JSON datum.

In the case of JSON files, metadata i s stored and the file is also splittable but again it also doesn’t support block compression.

1. Sequence Files

Sequence file stores data in binary format

 It also doesn’t store metadata

It supports block compression.

Due to complexity, sequence files are mainly used in flight data as an intermediate storage.

1. RC Files (Record columnar)

RC file was the first columnar file in Hadoop.

Significant compression and query performance benefits.

Disadvantage: RC does not support schema evolution. If you want to add anything to RC file you will have to rewrite the file. Also, it is a slower process.

1. ORC Files (Optimized Record Columnar).

ORC is the compressed version of RC file and supports all the benefits of RC file with some enhancements like ORC files compress better than RC files, enabling faster queries.

Some benchmarks indicate that ORC files compress to be the smallest of all file formats in Hadoop.

Disadvantage:

But it doesn’t support schema evolution.

1. Parquet Files

Parquet file is another columnar file

Parquet also enjoys the features like compression and query performance benefits but is generally slower to write than non-columnar file formats.

In Parquet format, new columns can be added at the end of the structure

Note: In Hive Parquet column names should be lowercase. If it is of mixed cases then hive will not read it and will give you null value.

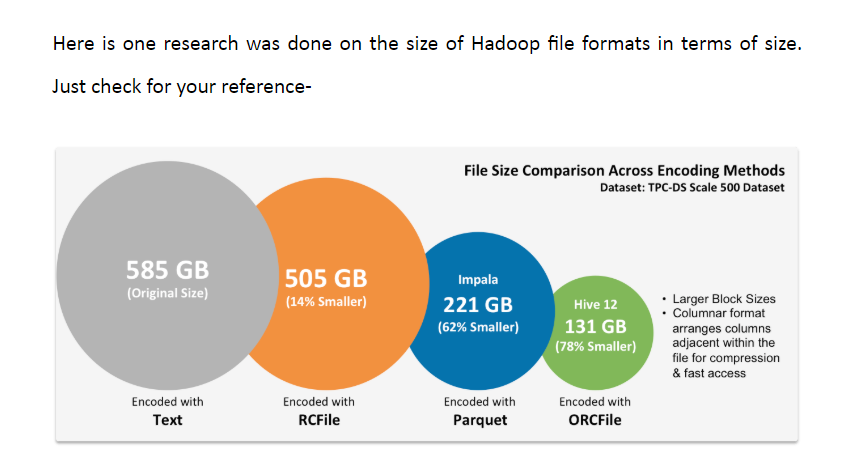
However, Impala can handle mixed cases.

1. Avro

Avro stores metadata with the data itself and allows specification of an independent schema for reading the file.

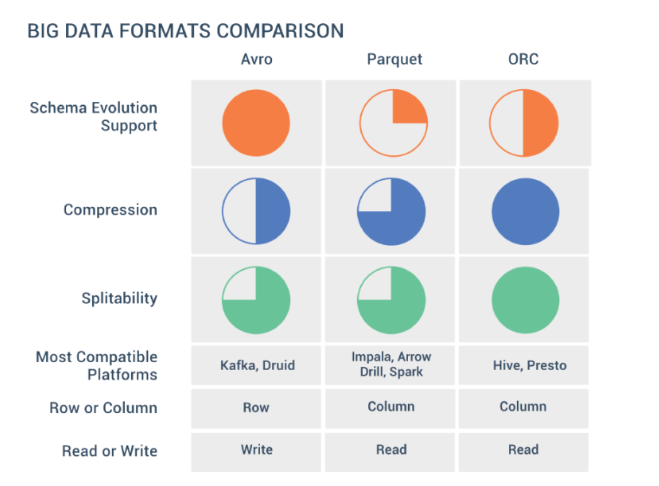
You can rename, add, delete, and change the data types of fields by defining a new independent schema.

Also, Avro files are splitable, support block compression and enjoy broad, relatively mature, tool support within the Hadoop ecosystem.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features** | **Text/CSV** | **Json** | **Sequence** | **RC** |
| **Data Storage** |  |  |  |  |
| **Block Compression** | **No** | **No** | **Yes** | **Yes.**  Columnar file |
| **Storage Requirements** | **585 GB** |  |  | **505 GB** |
| **Schema evolution** |  |  | **Yes.**  Appending new fields | **No**  If you want to add anything to RC file you will have to rewrite the file |
| **Write performance** | **Fast** | **Fast** |  |  |
| **Partial Read Performance** | **Slow,**  Due to lack of compression and column orientation. | **Slow** |  | **Fast.** |
| **Full Read Performance** |  |  |  |  |
| **Hadoop distribution Favorism** |  |  |  |  |
| **Processing tool (Hive, Impala, Spark)** |  |  |  |  |
| **Data Extraction** |  |  |  |  |
| **Use case** | When you have to do lot of data extraction to external databases. |  | If you are storing intermediate data between mapreduce jobs |  |
| **splittable** |  | **Yes** |  |  |
| **Metadata Storage** | **No** | **Yes** | **No.** |  |
| **Format** |  |  | **Binary.** |  |
| **Hadoop Eco system support** |  | **Less.**  Support from Hadoop But can **use Third party** |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Features** | **ORC** | **Parquet** | **Avro** |
| **Data Storage Format** | Column based Format | Column based Format | Row based Format |
| **Block Compression** | **Yes.**  High compression. | **Yes.**  High compression better than Avro but less than ORC | **Yes**  Compression rate is less compared to Parquet. |
| **Schema evolution** | **Yes.**  Not as good as Avro, but better than Parquet.s | **Yes**  Yes But not as good as Avro. When it comes to columnar ORC is better than parquet | **Yes (very good.)**  Avro offers superior schema evolution, Because of innovative use of JSON to describe the data, while using binary format to optimize storage size. |
| **performance** | **Write is Slow.**  **Read is fast** | **Write is Slow.**  **Read is fast** | **Write is fast.**  **Read is slow.** |
| **Compatible platforms** | Hive, Presto.  HortonWorks Data Platform (HDP) | Impala, Arrow drill, Spark.  Cloudera Distrbution of Hadoop (CDH) | Kafka, Druid |
| **Use case** | When you need more read performance  (Horton works and Hive)  We should use column based file format when we are trying to do a query on very few columns from your entire dataset. | When you need more read performance.  (Cloudera Impala)  We should use column based file format when we are trying to do a query on very few columns from your entire dataset. | If schema is going to change over time |
| **Storage Requirements** | **131 GB** | **221 GB** |  |



1. **Use case for ORC and parquet? (Column based)**

A big company has millions of employees where an executive wants to find the salaries paid to workers grouped by each location. If the salary and location data sets are stored in a column-oriented manner, then it’s a relatively simple query that only needs to touch data in those two columns.

But if you want to do row-by-row, you have to fetch millions of rows and do the operation on each of the rows.

We should use column based file format when we are trying to do a query on very few columns from your entire dataset.

1. **Use case for Avro? (Row based)?**

Let’s say we have to display available flights to a user on a Web page.

In that case we are going to get lot of information about the row and get continuous entries.

like say all the entries from 9 o’clock this morning to 1 o’clock this afternoon. That’s a classic SQL row-based query.

1. **What is Block Compression?**

Note: There is no relation to Block compressions “Block” and HDFS File System Block both are different.

1. **If you have requirement of best compression means in case storage is very critical to you then which compression you use?**

ORC : Optimized Record Columnar , Then Next we can use Parquet. Usually columnar based compression provides high compression ratio.

1. **When working with parquet file in Hive what are the precautions to be taken care?**

In Hive Parquet column names should be lowercase.

If it is of mixed cases then hive will not read it and will give you null value.

Whereas impala can handle this.

1. **When to Choose column based and when to choose row based data (file) formats?**

If the data is narrower, has a fewer number of attributes, and is read-heavy, then a column-based approach may be best.

If the data is wide, has a large number of attributes and is write-heavy, then a row-based approach may be best.

1. **What is the block size of your cluster?**

128MB can check hdfs-site.xml’s dfs.blocksize configuration

1. **What is the replication factor of you HDFS cluster?**

3

Can find using

hdfs dfs –ls

hdfs dfs –stat %r

hdfs-site.xml’s dfs.replication

1. **How partitioning is done in HDFS? IMP**

Partitioning in HDFS is done based on block size.