Hadoop - Bigdata

What is big data?

* Big data is a problem to store large amount of data.
* It is difficult for store, retrieve, process and do analysis of those data.
* To solve those issues, Hadoop comes into picture.

Why do we need Hadoop?

* Everyday a large amount of unstructured data is getting dumped into our servers.
* The Major challenge is to store those large amounts of data and retrieve and do analysis of those data and those present in different server in different locations.
* In this situation, Hadoop came into picture.
* Hadoop is having an ability to do analysis of those big data in different servers and different location very quickly.
* It uses the concept of MapReduce. it will divide the Query into Small parts and process them in parallel. This is also called as Parallel computing.

How is Analysis of Big data will be useful for organization?

* Effective analysis of Big Data gives a lot of Business profits. From that which area we need to give more focus and which area we need to give less important.
* Big data analysis provides some key points to prevent the company from huge loss or grasping a great opportunity.

What are the four characteristics of Big Data? or Four V's of Big data?

1. Volume 🡪 (Size of Data) Facebook generating 500 terabytes of data per day.

2. Velocity 🡪 (Analysis of Streaming data) Analyzing 2 million records each day to identify the reason for losses.

3. Variety 🡪 (Different form of data) images, audio, video, sensor data, log files, etc. Veracity: biases, noise and abnormality in data

4. Veracity 🡪 Uncertainty of data

What are the steps involved in deploying a big data solution?

* Data Ingestion – The foremost step in deploying big data solutions is to extract data from different sources which could be an Enterprise Resource Planning System like SAP, any CRM like Salesforce or Siebel, RDBMS like MySQL or Oracle, or could be the log files, flat files, documents, images, social media feeds. This data needs to be stored in HDFS. Data can either be ingested through batch jobs that run every 15 minutes, once every night and so on or through streaming in real-time from 100 ms to 120 seconds.
* Data Storage – The subsequent step after ingesting data is to store it either in HDFS or NoSQL database like HBase. HBase storage works well for random read/write access whereas HDFS is optimized for sequential access.
* Data Processing – The ultimate step is to process the data using one of the processing frameworks like MapReduce, spark, pig, hive, etc.

What is a Difference between RDBMS and Hadoop?

1. RDBMS is handing only structured data. But Hadoop will handle Structured, semi structured and Unstructured data.

2. RDBMS is supporting OLTP, But Hadoop is not suitable for OLTP, it is only for OLAP.

3. storing Transactional data, when we want to do analysis, that time we need to take backup and do analysis,

But in Big data, no need to take backup and do analysis, it is storing the data in distributed file system and process it.

4. Cost wise, Hadoop is cheaper, Because Hadoop cluster made up of commodity hardware. But for RDBMS, renewal thing itself it is taking more cost. (Teradata - 49000$)

What basic concept of Hadoop Framework?

Hadoop framework works based on main two components,

1. HDFS - HDFS is a file system which is designed for storing very large files and streaming data access patterns, running on the clusters of commodity hardware.

2. MapReduce: Map reduce is a framework which is used to processing large datasets in parallel across the cluster.

Main Components of Hadoop Application?

Core Components:

1. HDFS

2. Map Reduce

Data Access Components are - Pig and Hive

Data Storage Component is - HBase

Data Integration Components are - Apache Flume, Sqoop, Chukwa

Data Management and Monitoring Components are - Ambari, Oozie and Zookeeper.

Data Serialization Components are - Thrift and Avro

Data Intelligence Components are - Apache Mahout and Drill.

What are the Modes of Hadoop?

1. Standalone mode: - If all Deamons are running same JVM is called as Standalone mode

2. Pseudo distributed mode: - If All Deamons are running in different JVM in Single machine is called as Pseudo distributed mode.

3. Fully distributed mode: - If all deamon are running in a different JVM and different machines is called as fully distributed mode.

Types of data?

1. Structured data

2. Semi structured data

3. Un structured data

* Data which can be stored in traditional database system in the form of rows and columns is called as Structured data.

Ex: Online transactional data

* Data which can be stored as partial traditional database system.

Ex: XML and Json records are referred as semi structured data.

* Data which cannot be referred as Structured and Semi structured data is called as Un-Structured data.

Ex: Facebook updates, Tweets on Twitter. web logs ..etc

What are most input formats in Hadoop?

1. Text Input format 🡪 This is a default input format defined in Hadoop.

2. Key Value input format 🡪 This input format is used for plain text files where the files are broken down into lines.

3. Sequence File Input format 🡪 This input format is used for reading files in sequence.

What is Fault Tolerance?

Data should be available after server down.

Consider you have stored the data in one machine. Due to some technical issue, your file is destroyed. then there are no changes to get back the file.

To Avoid these situation Hadoop is introduced Replication Concept. When storing itself Hadoop will replicate the copy into two more copies of data.

What is a rack?

a) Rack is a storage area which contains all data node puts together.

b) Rack is a physical collection of data nodes which are stored at a single location. There can be multiple racks in a single location.

What is rack awareness?

* Rack awareness is a way which the name node determines how to place the blocks based on rack definitions.

What is Block?

* A disk is having block size which is a minimum size to read and Write operation. A big file (500MB of data) is splitted into multiple chunks. Each chunks is called as Block.
* Default block size of the Hadoop 1.0 is 64 MB
* Default Block size of the Hadoop 2.0 is 128 MB

What is Block Scanner?

* Block scanner used to tracking the list of blocks present in Data node and verify that any Checksum Errors.

What is File System?

* It is used for read and write

Ex: NTFS, HDFS, S3..

Types of file system?

* Standalone (Ex: Windows, Ext)
* Distributed (Ex: HDFS,S3)

What is distributed OS computing?

* More than one machine sharing the data.

Types of distributed file system?

1. Master and Slave Architecture (Hadoop)

2. Peer to Peer Architecture (Cassandra)

What is Process?

* Program in Execution

What is Deamon Process?

* Deamon is a process which runs in the background.

What is JPS commands do?

* JPS is Java process status which gives the status of deamon which are running in the cluster.

How indexing is done in Hadoop?

* Hadoop is having own way of Index Depending upon the block size. When the data is saved, HDFS will keep storing the last part of the data, it will say where is the next part of the data will be.

What is the communication channel between client and name node/data node?

SSH Communication.

Types of Deamons?

1. Name node

2. Data node

3. Job Tracker

4. Task Tracker

5. Secondary name node

Each deamon is running separately in its own JVM

Name Node

1. Name node is a deamon process, which is running in master machine.

2. It is used to store the metadata for files and directories in the form of Edits and fsImage. All recent transactions are available in Edits log, when restarts happen, It will sync with Edits log data. By default, it will take time. So, during the particular time, Edit log will be flushed to fsImage. This duration is called as Safe mode. During the Safe, Read and Write operation will not be happened.

To avoid Safe mode, Secondary name node comes into picture.

Checkpointing

The Process of merging edits with fsImage is called as Checkpointing.

Thats why Every one hour (dfs. namenode.checkpointing.period) or no of transactions reaches 1 million (dfs.namenode.checkpointing.txns) Checkpointing services will do process of merging edits file with fsImage file. This process is called as Checkpointing.

Note: whenever new fsImage is created. It will merge with existing fsImage and renaming to new fsImage. Because if any problem, current fsImage file gets corrupted, then previous fsImage is having older data.

Question is how SNN will know about the transactions happened in NN.?

Every 60 sec, SNN pinging NN about that how many un-merged transactions have happened.?

why multiple fsImage are there in the folder?

whenever checkpointing has happened, new fsImage will be created, because if any problem, current fsImage file gets corrupted, then previous fsImage is having older data.

why md5 file there in fsImage.?:

It is used to check whether fsImage is corrupted or not.

How to decide that fsImage is corrupted?

Actually when reading and writing operation happens, new fsImage will be created, when write operation happens, it compares current checksum with oldest check sum, If its not matched, then fsImage is corrupted.

Why NN Edits and fsImage are not in HDFS.? It is in NN Ram memory?

1. Edit and fsImage are keep updating, If it is in HDFS, you should not be modified it. Write Once. It is read only.

2. Name node is having more security than data nodes.

Limitations of Check point node?

1. NN is required some more time to restart.

2. data loss, Ex: My transaction is happening 8 --> 9 --> 10 --> 10:30 NN is failed, when checkpointing is happening between 10 to 10:30, The data might be lost.

To Overcome this issue Hadoop comes up with Backup Node.

Difference between Checkpoint node and Secondary Name node?

Check Point node Secondary Name node:

1. After check point process, It will update merged new fsimage with Active Name node fsImage. 1. After check point process, It will not update merged new fsimage with Active Name node fsImage.

Backup Node:

1. Name node RAM is always sync with Backup node RAM. Whatever transaction is happening in NN will be synced to Backup node.

2. There is no check pointing will be happening in Active name node.

3. Check pointing is directly happening in Backup node.

Same configuration and directory structure settings in both NN and Backup node.

Limitation of Backup Node:

1. Both NN and backup node is down, rare case, we cannot do anything.

2. For going SNN/Check pointing node, We can any numbers of nodes. But for Backup node. we should connect only one backup node.

High Availability

1. Active NN and Standby Name node

When both are available, all write operation will be take care by Active NN and All read operation will be take care of Stand by NN.

Data Node

1. Data node is a deamon process, which is running in Slave machines.

2. It is used to Store and Retrieve data when its required.

3. It is sending heartbeat to Name node for every 3 seconds.

Job Tracker

1. Job tracker is a deamon process, which is running in Name node.

2. It is used to Submit and Tracking of Map reduce jobs.

3. It will receive heartbeat from Task tracker for every 3 seconds

4. There is an only one job tracker will be available for the cluster

5. Job tracker is a Single point of failure.

Task Tracker

1. Task tracker is a deamon process, which is running in Data node.

2. It is used to launch and monitoring the tasks in corresponding data nodes.

3. It will send heartbeat to job tracker for every 3 seconds.

Secondary Name node

1. Secondary name node is used to do check pointing process for every one hour or number of transactions reaches 1 million transactions.

2. After checkpointing, it will not update recent fsImage with Active name node fsImage.

Write Operations

1. When write operation came to cluster, First Client API will be created, after that Client API sending request to name node, Name node is going to generate 3 meta data's.

a) Generation of blocks, B0,B1,B2,B3...B16

b) Making Replication, B0 B0 B0, B1 B1 B1,.....B16 B16 B16.

c) Where to place those replication blocks.

2. After created meta data, Response sending to client API. Client API starts perform the response through pipeline. It will go to first slave machine and write what are blocks need to be written and then move to next block and do the same.

3. After written all blocks, giving an acknowledgment to Client API as Completed, then Client API will send response to Name node as Job completed.

Read Operations

1. When Job is submitted to Job tracker, Job tracker will contact Name node for block details, Name node will search in meta data and give the response to Job tracker as Block details.

2. Job tracker will split the job to multiple tasks, Assign the task to each Task tracker and monitor those task trackers by heartbeat.

3. Once task is received by Task tracker, Task tracker will launch the Map task based on Resource Availability. Task tracker is requesting data node for data. The time taken for taking the data is called as Look up time.

4. Once Map task is completed, output will be written in temp folder of data node. After all map tasks are completed, Task tracker launch the Reduce task based on No of reducers quoted in Driver program.

5. Output of reducer will be saved to local temp folder. After that result will be flushed to HDFS. and informed to Job tracker as Job completed.

Drawback or Problem for Hadoop

* Hadoop is having a Single Point of Failure (Job Tracker and Name Node). If its down, Entire cluster will be down.

Solution is Hadoop 2.0

1. To Overcome Single point of failure, Hadoop provides two solutions,

a) Federation

b) High Availability

What is Federation?

* To make a multiple Name node is called as Federation, One machine act as an Active name node and Another machine act as a Standby Name node.
* Suppose Active name node is down, Standby name node act as Active Name node. Suppose Second name node also down, then entire cluster will be down.

What is High Availability?

* To overcome the Federation problem, Hadoop provides a solution as High Availability. Putting Journal node on top of all Name nodes,
* All the transactions are not available in Name node, its having only latest transactions. Other transactions are stored in Journal nodes.
* All write operation will happen through Active name node and sync up with Stand by name node.
* All read operation will happen through Stand by Name node.
* It will read the meta data from Journal node and do process. We can have a Multiple Journal Node.
* Zookeeper is monitoring those Journal nodes by Receiving hear beat from Journal node. Data node is sending heartbeat to both Active and Standby name nodes.

MapReduce

* MapReduce is a framework, which is used for processing the data set which is stored in HDFS, MapReduce processes the data in parallel by dividing the job into set of independent tasks, So Parallel processing improves speed and Reliability.

Advantages of MapReduce

1. fault tolerance (Automatic Fail over of Task)

2. Parallelization

Steps of MapReduce Job Execution flow

1. Input-Files: - Data which is stored in a HDFS.

2. Input Format: - Input format defines how to Split and read the data from HDFS. Input format creates input split.

3. Input Split: - Its logical representation of data, which will be processed by individual mapper, for each split, one map task is created.

Number of Splits = Number of Mappers

Framework will divide split into Records.

4. Record Reader: - It reads the data from split and converts the data into key-value pairs.

5. Mapper: - It reads the data from Record reader and generates Intermediate key-value pairs. This intermediate key-value pair is completely different from input pair. Because output of mapper is a collection of key-value pairs. Before writing the output of each mapper task, Partitioning will take place on the basics of the key. Partitioning will grouping all the key bases on key. Once Partition completes, Map output will be written to the HDFS.

6. Combiner: - Combiner is the Mini Reducer which performs the local aggregation on the mapper output. It minimizes the data transfer between mapper and reducer. when the combiner functionality completes, framework will pass the output to the partitioner to further processing.

7. Partitioner: - Partitioner comes into picture when you are working with more than one reducer. It takes the output from combiner and perform partitioning. Partition will be decided bases on the hash key of the object. It will group the data based on the key.

No of Partitioner = Number of Reducer

Basics of key- value, records are having the same key value will go to same partition (With in each mapper), Then each partition will send to corresponding reducer.

Partition will take place After Map phase and Before Reduce Phase.

Hash Partitioner is a default partitioner, It computes hash value of the key.

Partitioner in a MapReduce job redirects the mapper output to the reducer by determining which reducer handles the particular key.

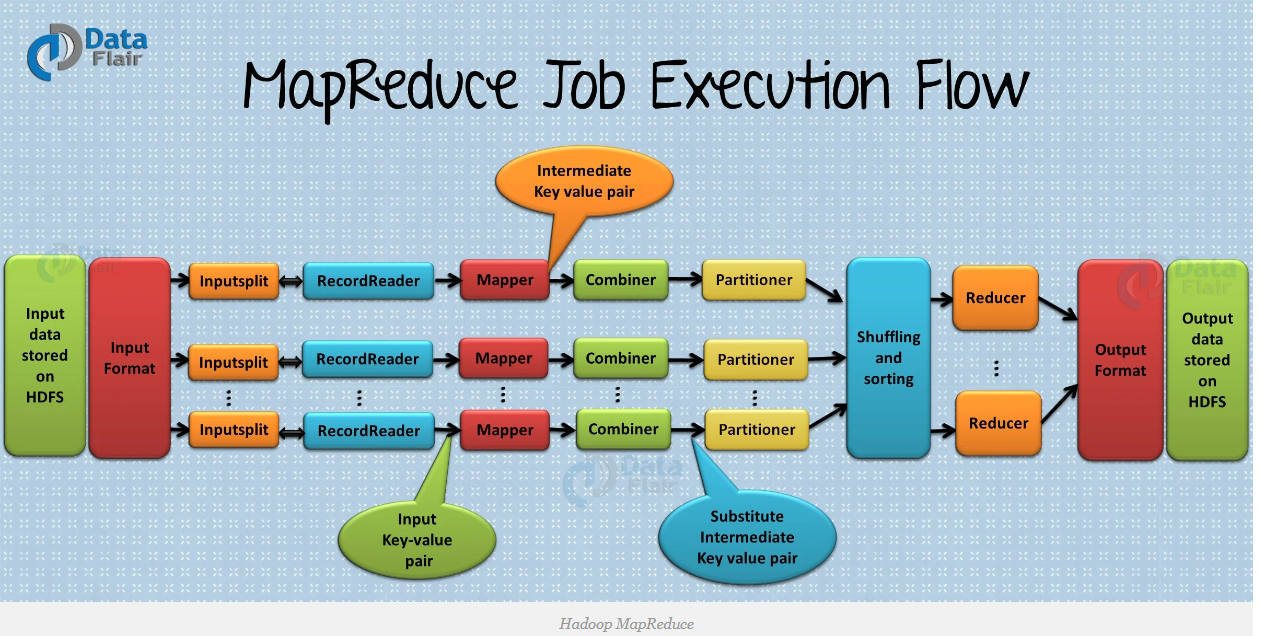
Framework will generates partitioner when there are more than one reducers.

8. Shuffling and Sorting: - After Partitioning, the output is shuffled to reducer node.Shuffling is nothing but physical movement of data which is done over the network. As all the mapper output is shuffled to corresponding reducers, Framework will merge the intermediate output and sort, then provide as input to reduce phase.

9. Reducer: - Reducers takes the input from Mapper and runs the reduce function on each of them to generate the output. output of reducer will be stored in HDFS.

10. Record Writer: - It writes Reducer output to HDFS.

11. output Format: -



Problem with MapReduce

* Limits Scalability
* Poor Resource Utilization (Fixed no of Map and Reduce Slots)
* Limitation of Running other Application.
* Availability Issues.

Solution is YARN

YARN - (Yet Another Resource negotiator)

1. YARN is a new component which is added in Hadoop 2.0, In Hadoop 2.0, new layer had been introduced between HDFS and MapReduce.

2. YARN is a Framework, which is responsible for Cluster resource Management. (Managing the resources of the cluster)

MapReduce - (Cluster Resource Management + Data Processing - Job Management)

YARN --> Cluster Resource Management

MapReduce --> Data processing

Problem with Hadoop 1.0 Approach

* Limits Scalability - Job tracker running on the single machine which doing several tasks,
  + Resource Management
  + Job and task Scheduling
  + Monitoring

Although so many other machines are available, they are not getting used. It limits scalability.

* Availability Issues: - Job tracker is a single point of availability, If job tracker downs, entire cluster will be down.
* Problem with Resource utilization: - In Hadoop 1.0, There is a concept of predefined map and reduce slots are available for each task tracker.
  + Sometimes, Map slots are full and its waiting for resources, That time, Reduce slots might be available. Here Resources could sit idle which is reserved for reduce slots, even when there is a immediate requirement for Map slots.
* Limitation in running non-MapReduce Application: - In Hadoop 1.0, Job tracker was tightly coupled with MapReduce and only supporting Application which supports MapReduce framework.

YARN - Next gen of Map reduce

* In map reduce 1, Scalability is a bottle neck, in 2010, Yahoo starts next generation map reduce with new feature to increase performance by Smarter memory utilization and enhance Scalability and flexibility
* Main Idea to Split the Job tracker responsibilities into two,
  + Resource Manager (Resource Management)
  + Application Master (Job Management - Data processing)

Advantages of YARN

* Increased Scalability to Splitting the responsibilities of Job tracker into two, Resource manager and Application Master.
* Better Memory utilization with the Concept of Containers, Containers are similar to Slots concepts. In Classic map reduce, slots are fixed in nature, But Containers are more flexible. Ex. Single job tracker would have fixed number of slots specific for map task and reduce task, but containers in YARN can run map task, reduce tasks
* Other than MapReduce program also can run in YARN.

Entities in YARN

1. Job Client: - Responsible for Submitting the job.

2. Resource Manager: - Its responsible for allocating the resources to jobs that are required. Resource manager responsibilities are divided into two components, which are Scheduler and Application Manager.

Scheduler - Scheduling the Job

Application Manager - Monitoring the Application

3. Node Manager: - It responsible to launch and manage containers

4. Application Master: - It is responsible for coordinating the task running and monitor the progress and aggregates it and sends the reports to the client.

5. YARN Child: - It is responsible for send updates and progress to Application Master.

Steps to How job runs in YARN: -

1. Job Submission Phase: -

1. Job is submitted to Job client. Job client requesting Resource manager for new Application ID. Resource manager checks the output directory, if it is present, then it will through error and stop the job there itself. Or It will create the Application or Job.

Job Initialization: -

1. Scheduler picks up the job and contacts Node manager to start new container for application Master.

2. Application Master retrieves the split from HDFS and create one task per split.

3. Next Application Master decides how to run MapReduce job. If the job is Small, then it will run the job in Same JVM itself.

Task Assignment Phase: -

1. If the Job is Huge, Then Application Master will contact Resource Manager for resources. Scheduler knows where the splits are located, It gathers the information from Heartbeat of the Node Manager. It will Allocate the resources to Application Master.

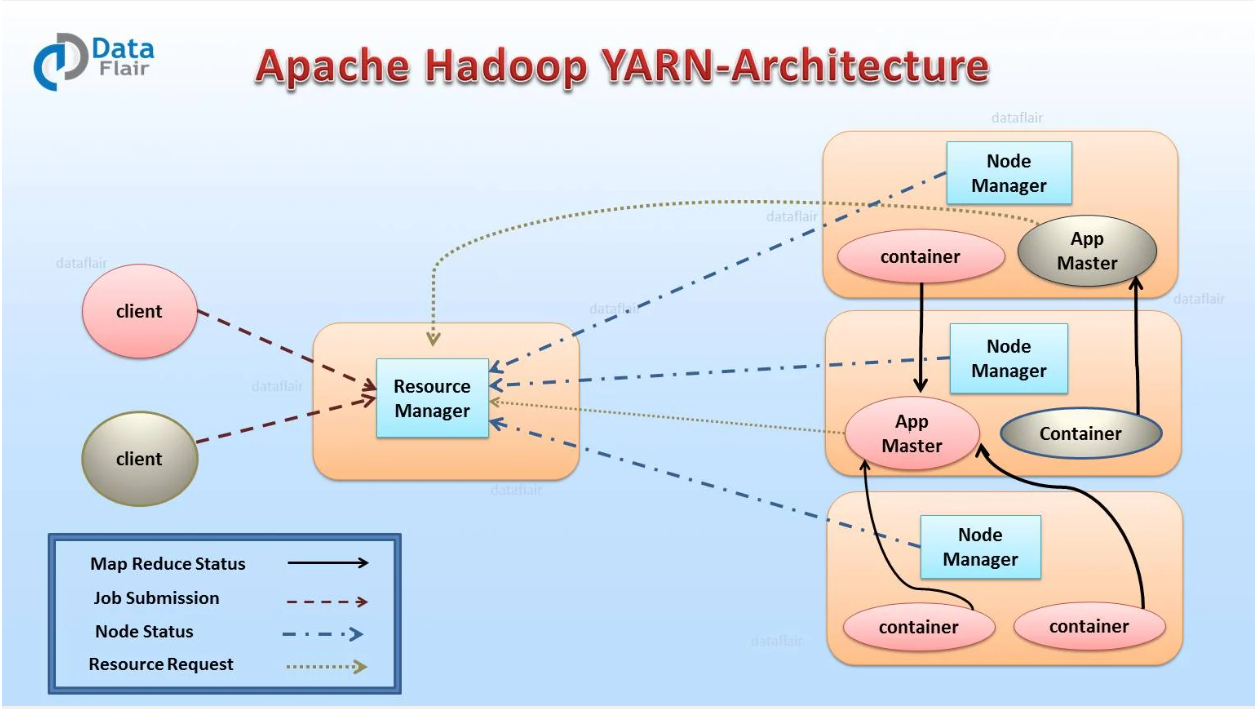
2. Then Application Master contacts Node Manager to start containers for executing the tasks. Next YARN child will be launched, YARN child is nothing but Java program which has a Main program named as YARN Child.

3. YARN child runs on a Separate JVM to isolate the User code from long running daemons, The same step will be happened in Classic MapReduce also. But one difference is, Classic MapReduce, Reuse of JVM is possible, But in YARN, Usage of same JVM for YARN child is not supported.

4. YARN retrieve the resources from HDFS and runs the MapReduce Tasks.

Progress Updates phase: -

1. YARN child send the progress report to Application master for every 3 seconds, Application Master Aggregates the progress and send to Client directly. Once tasks are completed, Application master and Containers are clean up his data.



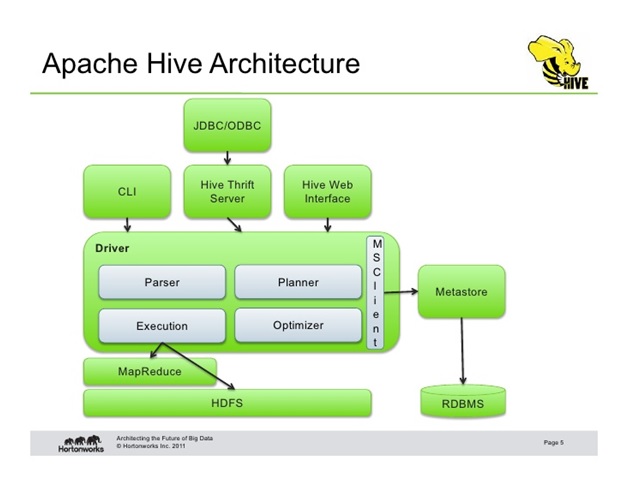
Hive

Introduction

* Hive is a Query Engine, which is used for querying data resides in HDFS. It will launch Map reduce jobs to get the results.

Hive Components

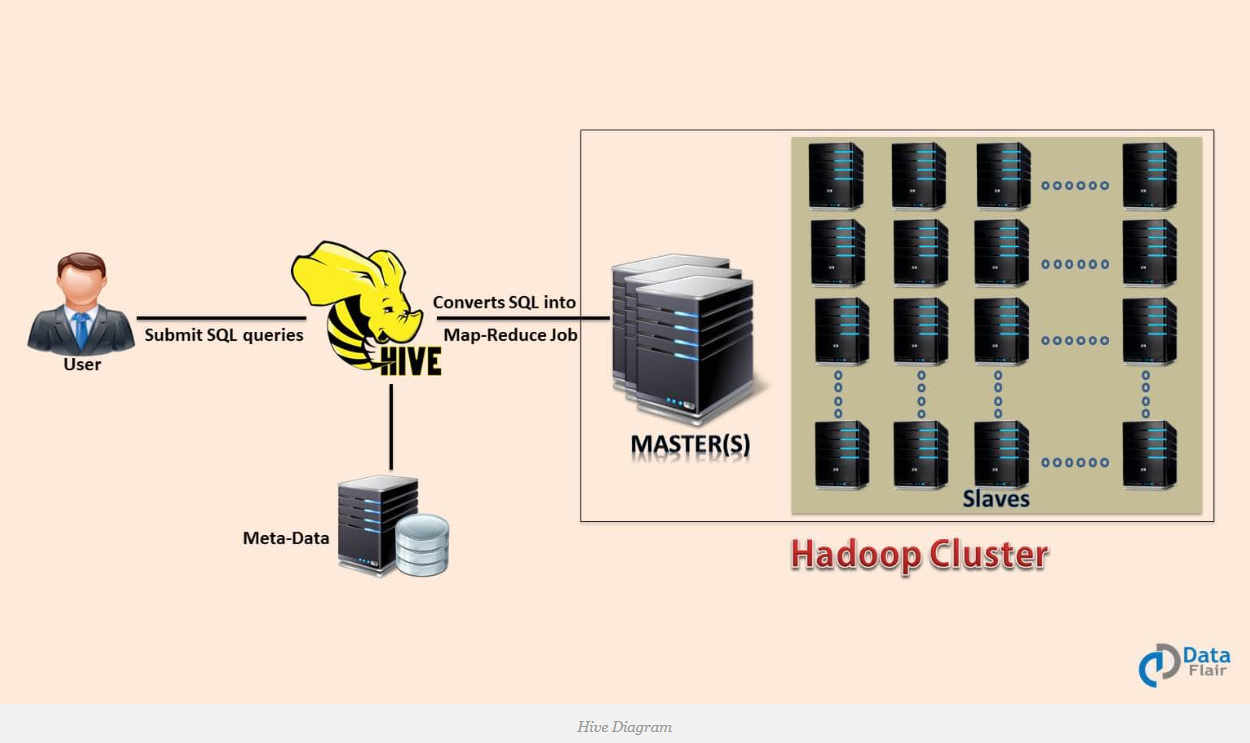
* Hive CLI, Hive Thrift Server, Hive Web Interface
* Driver 🡪 Metastore 🡪 RDBMS
* MapReduce, Tez, Spark
* HDFS



How it is Internal working?

1. The Query is submitted to hive client; it will check the schema in Metastore. Based on schema, it will generate the MapReduce job internally and submitted to the YARN cluster.

2. YARN will execute the job and get the results from HDFS and give it to hive client.



Meta store:

* Meta data stored in a database is called as Metastore
* Default Metastore --> metastore\_db.

Types of Metastore:

1. Embedded Metastore --> Driver, Metastore, Database are running in a Single JVM is called as Embedded metastore.

2. Local Metastore --> Driver and Metastore are running in a Single JVM and Database is connected locally is called as Local Metastore.

3. Remote Metastore --> Driver and Metastore are running in a different JVM and remotely connected with database is called as Remote metastore.

Hive CLI and Beeline CLI:

* Hive CLI is connected to Hive server 1, Can not be able to connect more than one CLI.
* Beeline CLI is connected to Hive server 2, Can be able to connect multiple client.

Variable Substitution:

* It is used for avoiding important value to be hard coded. It will be implemented by --hiveconf parameter. we can pass the value from outside the query.
* Ex: Set hive.variable.substitution = true;

hive> set CURRENT\_DATE='2012-09-16';

hive> select \* from Employee where day >= '${hiveconf:CURRENT\_DATE}'

Similarly, you could pass on command line,

> hive -hiveconf CURRENT\_DATE='2012-09-16' -f test.hql

HCatalog

HCatalog is the metadata management of Hadoop File system. Whatever the tables created in HCatalog can be accessed through hive and pig.

HCatalog can be built on top of hive.

Compression technique

* It is used to compress the data and stored it in HDFS.
* Default compression is GZip.

Different Compression Techniques

* Gzip
* bzip2
* snappy
* LZO
* ZLIP (ORC + ZLIP)

How to enable compression

set hive.exec.compress.output=true;

set mapred.output.compress.codec= (format we can mentuion it)

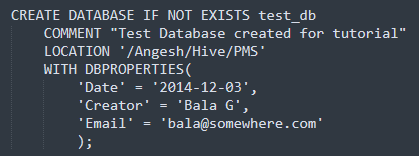
Hive Tables

* 1. Internal table
* 2. External table
* 3. Partitioned Table
* 4. Bucketed Table
* 5. Sorted Bucketed table
* 6. Skewed table
* 7. Temporary Tables
* a) Create Table As Select (CTAS)
* b) Create Table Like

How to execute Hql

bin/hive -f /home/datadotz/FirstScript.hql;

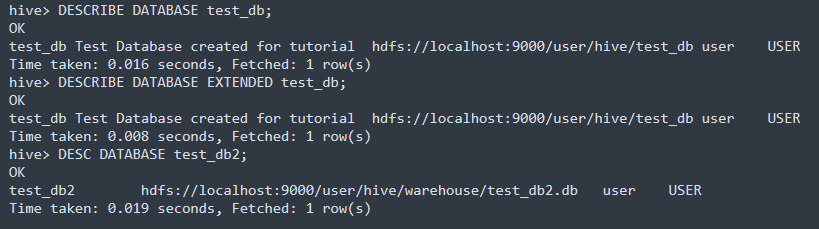
Create Database



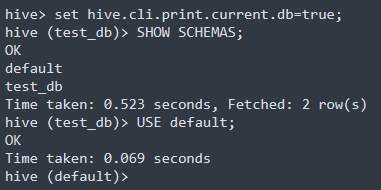
Show Database



Describe Databases



To know the current database:



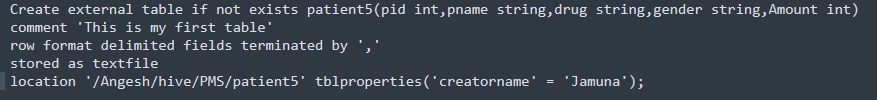
Internal table

* Internal tables are Managed tables which means its managing by Hive.
* In Internal table, Both Meta data and Actual data will be bounded each other. Thats why when you are dropping the internal tables, both schema and data will be dropped.

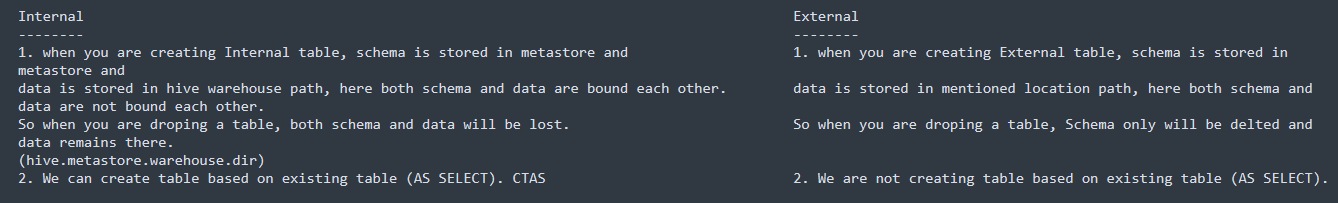


External table

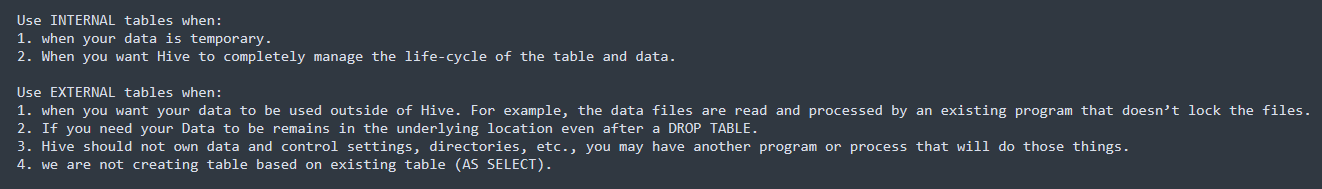
* External Tables are Non managed tables which means it does not managed by Hive.
* Meta data and Actual data will not be bounded each other. So when you are dropping the tables, Schema only dropped, Data will be remains there.



Difference between Internal and External Tables



Use case: When to use?



Partitioned table

* Way of segregating a table into multiple files/directories based on partitioned columns defined into table definition.
* Partitioning gives effective results when,
  + limited number of partition and Equal sized Partition

Types of Partition:

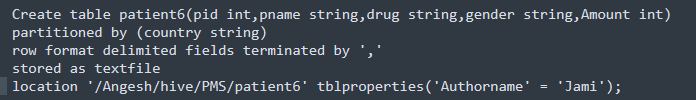
1. Static Partition

2. Dynamic Partition

Static Partition:

* We have to give the partition name during the Insert, It will not take it as dynamically.
* No need to give partition column name in the Select list.

Create Table:



Dynamic Partition

* We don't have to give the partition name during the Insert, It will take it from the data dynamically.
* We should give the partition column as a last column in the Select list.

Properties Need to Set

SET hive.exec.dynamic.partition = true; 🡪 Set to True to Enable Dynamic partitions

SET hive.exec.dynamic.partition.mode = nonstrict;

🡪 Set to Non-Strict to Enable all partitions to be determined dynamically

Set hive.exec.max.dynamic.partitions = 1002;

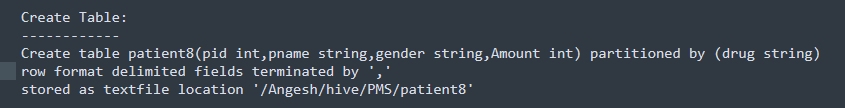
🡪 Maximum number of partitions that can be created by Mapper/reducer

Set hive.exec.max.created.files = 100002;

🡪 Maximum Number of files that can be created globally. Raise the Fatal Error If the limit exceeded.

Difference between Strict and Non-Strict: -

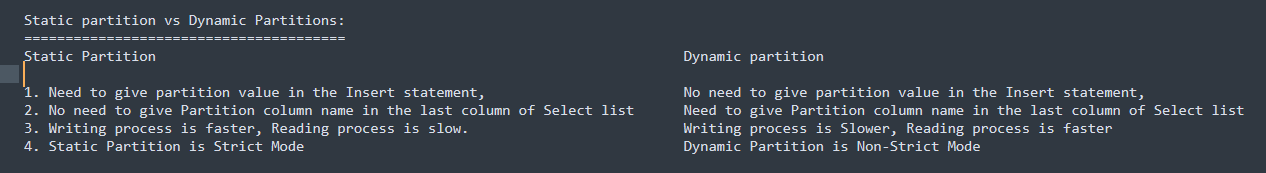
* Strict does not allow all partition to be load data dynamic. But non-strict mode will allow all partition to load dynamically.



How to Sync Files with Partitions

MSCK Repair table Patient.

Static partition vs Dynamic Partitions



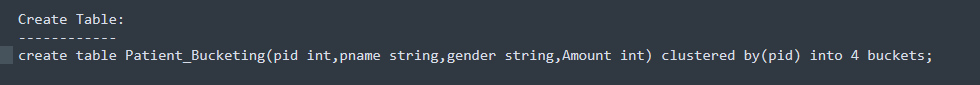
Bucketing Table

1. Another way of segregating a table into multiple files/directories based on Hashing function of the bucketed column mod No of buckets.

2. Records with Same Hash value will get stored in same bucket.

3. We use Clustered by clause to divide a table into Buckets.

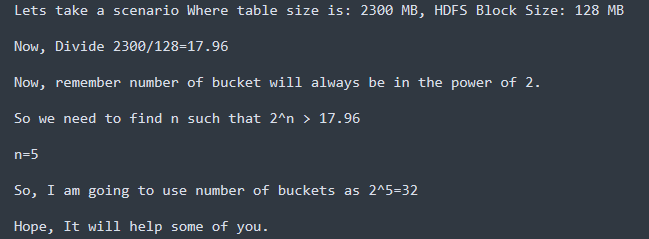
4. Bucketing can be done along with Partitioning Table or without Partition.



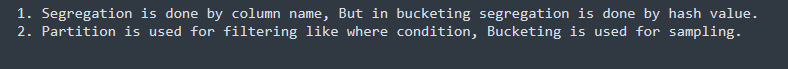
Properties to be set

Set hive.enforce.bucketing = true;

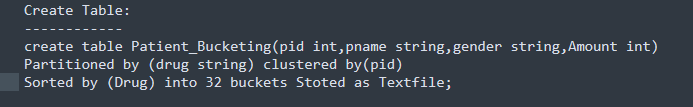
How to decide no of Buckets in Hive



Partitioning VS Bucketing

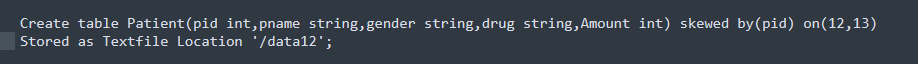


Sorted Bucketed Table



Skewed table

* Skewed Tables are special type of tables, where values that appear very often are split into separate files and other values are going to some other files. during the Skewed join, job will it.



Hive Joins

* Shuffle Joins
* Map Join (Broadcast Join)
* Bucket Map join
* Sort Merge Bucket (SMB) Map Join
* Skew join

Shuffle Join

* It is a default join and it includes a map stage and a reduce stage.
* Use case
  + It works for any Table size
  + Especially when another join cannot be used. for example, full outer join

Broadcast Join

* If one or more tables are small enough fit into memory, The Mapper scans the large table and do join.
* 2. No shuffle and Reducer stage.

Use case

1. Small Table joins Big Table, It is very fast because It saves Shuffle and reduce stage.

Bucket Map Join

* Mapper process Bucket 1 from Table A only fetch Bucket 1 from Table B
* Both the tables should be Bucketed

Use cases

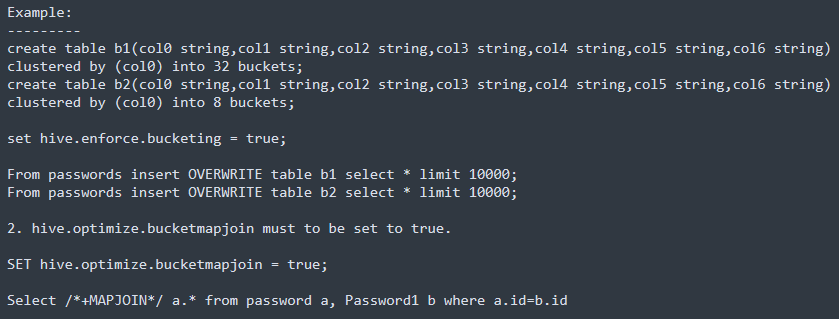
When all tables are

1. Large

2. Bucketed using the Join columns

3. Number of Buckets in one table is a Multiple of Number of Buckets in another Table.

4. Not sorted



Sorted merge Bucket (SMB) Map Join

* The corresponding buckets are joined with each other at the mapper

Use cases

When all tables are:

a) Large.

b) Bucketed using the join columns.

c) All tables have the same number of buckets.

d) Sorted using the join columns.

Skew Joins

* IF Table A joins B and A has Skew data "1" in joining column. First read B and store the rows with the key 1 in in-memory hash table, Now run the set of mappers to read A and do the following,
  + 1. If it has key 1, use the hashed version of B to compute the result.
  + For all other keys, send it to reducer which do the join and Reducer will get the rows of B from mapper.
  + This way, we end up reading only B twice. The skewed keys in A are only read and processed by the Mapper, and not sent to the reducer. The rest of the keys in A go through only a single Map/Reduce.
  + The assumption is that B has few rows with keys which are skewed in A. So these rows can be loaded into the memory.

Use cases: -

1. One table has huge skew values on the joining column.

File formats

* It is used to find out which information is stored in a file system.
* 2. Hive does not verify whether the data that matches the schema of the table or not. However, it verifies if the file format matches the table definition or not.

Types of File Formats

1. Text File

2. Sequence File

3. RC File (Record-Columnar File)

4. ORC File (Optimized Row Columnar)

5. Parquet

6. Avro

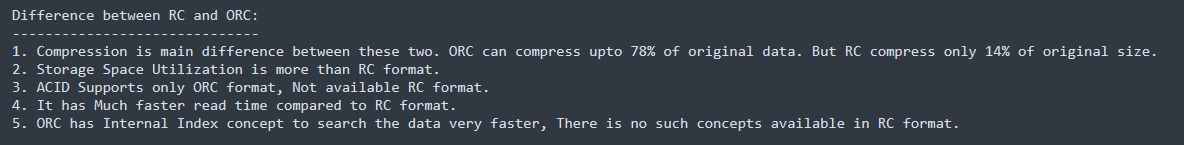
RC File

* RC files are flat files, it splits data horizontally into row groups and store it in a columnar format. For Example, if we have 1000 records in table, 1 to 100 records are stored in a one group, 101 to 200 stored in a next group and so on. RC file stores the Row Group in a columnar format.
* Benefits of RC
  + 1. Fast data loading
  + 2. Fast data processing
  + 3. highly efficient storage space utilization

ORC File: (Optimized Row Columnar)

1. ORC stands for Optimized Row Columnar which means it can store data in an optimized way than the other file formats.

2. ORC reduces the size of the original data up to 75%. As a result the speed of data processing also increases. ORC shows better performance than Text, Sequence and RC file formats.

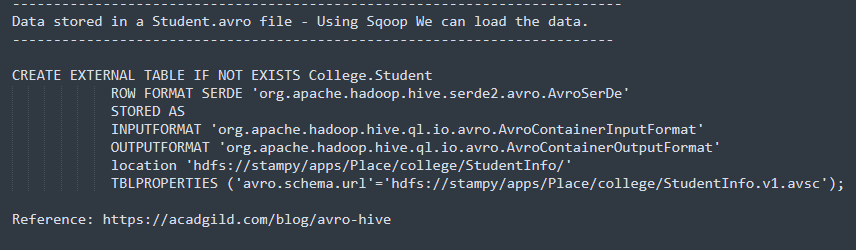


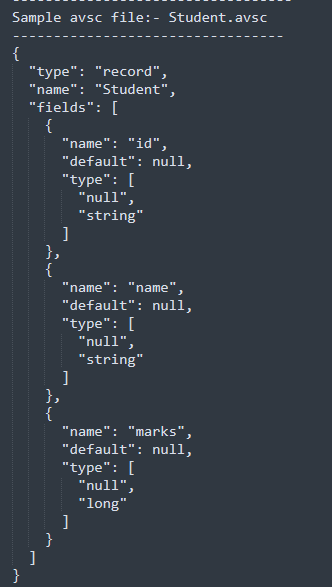
Avro

* Avro is data Serialization format, Because of Its language neutrality. Due to Lack of Language portability, Avro becomes Natural choice, which can able to handle Multiple Languages data.
* Avro is much preferable for Serializing the data in Hadoop.
* It uses the JSON for defining columns, data types and other stuffs.
* Schema is available in Separate file and actual data ia available in HDFS.

🡪 Schema is available in avsc file is the form of JSON.

🡪 data is stored in a Avro file (1.avro)





Parquet

* Parquet format stores the data in Columnar format. which will increase speed of Querying data. Parquet is used in Cloudera and ORC is used in Hortonworks.
* Parquet is having a high compression compared to Avro.
* Parquet will be highly suitable when you want to do Aggregation work.
* Parquet Compress 64% of Original data size

Difference between Avro and Parquet

1. Parquet is a columnar format, But Avro is Row Format.

2. Parquet is having high Compression rate when compared to Avro.

3. Processing Speed is High when compared to Avro.

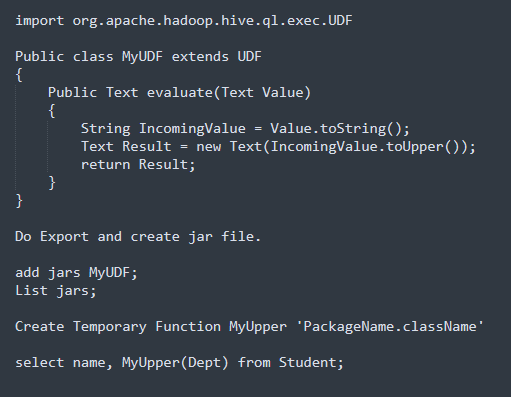
4. Parquet does not maintain schema in separate file as like Avro.

ACID Transactions (Atomicity, Consistency, Isolation, Durability)

* Initially, Hive did not support Update tables. Version 0.14 onwards, Hive supports ACID transactions. Using ACID, you can delete and Update the hive Tables.
* For implementing ACID transactions, below conditions are there,
  + Table should be bucketed.
  + File format should be ORC
  + Below table property should be enabled,
    - TBLPROPERTIES ('transactional'='true')
  + Below properties should be set,
    - SET hive.support.concurrency=true;
    - SET hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
    - SET hive.enforce.bucketing=true;
    - SET hive.exec.dynamic.partition.mode=nonstrict;

UDF (User defined Functions) and UDAF (User defined Aggregate functions)

* Hive is having some default Build in Functions. If your requirement is not satisfied with this build in functions, then you need to write UDF's to achieve your requirement.



Order by vs Group by vs Sort by vs Distribute By vs Cluster by

Order By 🡪 It will group the data based on Columns mentioned in Order by and Making order them as Ascending or Descending. Total Ordering is guaranteed.

Group by 🡪 It will group the data based on Columns mentioned in Group by and display the aggregation result.

Sort By 🡪 It will sort the data before feeding the data into Reducer. It will sort the data based on columns mentioned in sort by.

Distribute By 🡪 Distribute the Rows among Multiple Reducers based on column mentioned in the Distribute by. All Distribute by columns will go same reducers. It does not sort the output in each Reducer.

Cluster By 🡪 Combination of Distribute by + Sort By. It will sort the data before feeding into Reducer and distribute rows among Multiple Reducers.

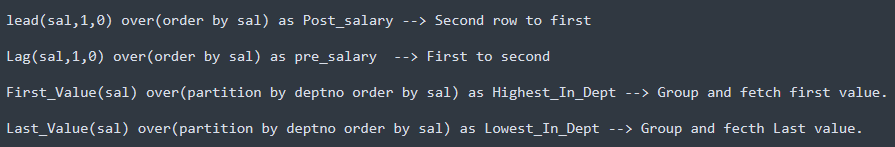
Window Functions

1. Lead 🡪 Lead function is used to return the data from next row. (Second Row to First)

2. Lag 🡪 The LAG function is used to return the data from a previous row. (First Row to Second)

3. First\_Value 🡪 Group and fetch first value.

4. Last\_Value 🡪 Group and fetch Last value.



Analytical Functions

1. Rank 🡪 used to assign a rank in a Group of Records

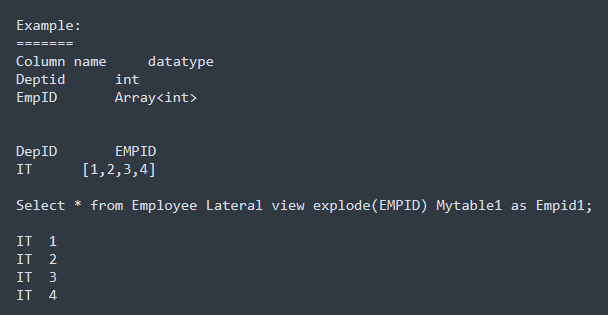
2. Dense\_Rank 🡪 same like rank() except It assigns consecutive ranks

3. Row\_Number 🡪 Used to assigns a unique number to each row.



Lateral View (Converting column into Rows)

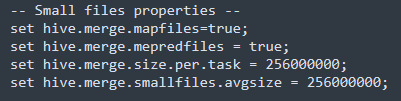
* Lateral view is nothing but If we have a table with array data type, and if we want to explode your data then you go for Lateral view. Converting column into Rows



Optimization Techniques

* Partitioning
* Bucketing
* Types of Joins
* Parallel Execution 🡪 SET hive.exec.parallel = true
* ORC format
* Change Execution Engine to TEZ or Spark.
* Vectorization
  + Processing the batch of rows together instead of processing one row at a time. Default It is false, must enable it to true
  + Set hive.vectorized.execution.enabled=true;

How to handle small file issue



Impala

Introduction

* Impala is a MPP SQL query engine which performs analytical queries on top of HDFS, HBase, S3 and Kudu storage via SQL or BI tools without migrate data sets to anywhere. it can be integrated with BI Tools like Tableau, Looker etc.
* It helps to read text, RC, ORC, Parquet and Avro file formats. Specially optimized for reading data in the Parquet.
* It is directly executing Map reduce jobs on top of HDFS. not through YARN.
* Impala uses the same Hive Metastore (HMS), Hive SQL syntax, Hive ODBC driver, and Hive user interface (in Hue). As Impala integrates with the Hive Metastore, enables users to create tables in Hive and query them using Impala seamlessly.

Why we need Impala.?

* Provide faster interactive data exploration on top of HDFS.
* Perform Analytics on BI tool. (Connect BI tool with Hadoop)

Pros of Impala

* We can perform analytical queries on top of HDFS, Kudu, S3 or HBase without migrate data to anywhere. Also, Impala integrates with HMS, So, that, you can query Hive tables in Impala as well.
* Impala is optimized for reading data in columnar storage formats, especially Apache Parquet.
* MPP processing.

Cons of Impala

* Impala relies on the Hive Metastore for managing metadata, and any issues with the Hive Metastore can affect Impala's ability to access metadata and tables.
* Impala has limited support for update and delete operations.
* There can be limitations in terms of the number of concurrent queries that can be effectively processed.

Limitations of Impala

* Whenever new records/files are added to the data directory in HDFS, the table needs to be refreshed.
* Impala can only read text files, not custom binary files.

Components of Impala

Coordinator node

* A Node which co-ordinates the request is called as Coordinator node.
* Co-Ordinator node will update metadata changes to HMS.

Catalog Service

* Catalog Service relays the metadata changes from Impala SQL statements to all the Impala daemons in a cluster.
* It uses a daemon process named catalogd

State Store

* Impala State store, which is responsible for checking the health of each Impalad.

Impala daemon (Impalad)

* Impalad is a daemon which runs on each node of cluster where Impala is installed. It accepts the queries from various interfaces and processes them.

Impala Metadata & Meta Store

* Impala uses traditional MySQL or PostgreSQL databases to store table definitions.
* The Table, columns and table definitions are stored in a centralized database known as a meta store.

Each Impala node caches all the metadata locally. When a table definition or table data is updated, we have to update metadata cache of each node by running below commands,

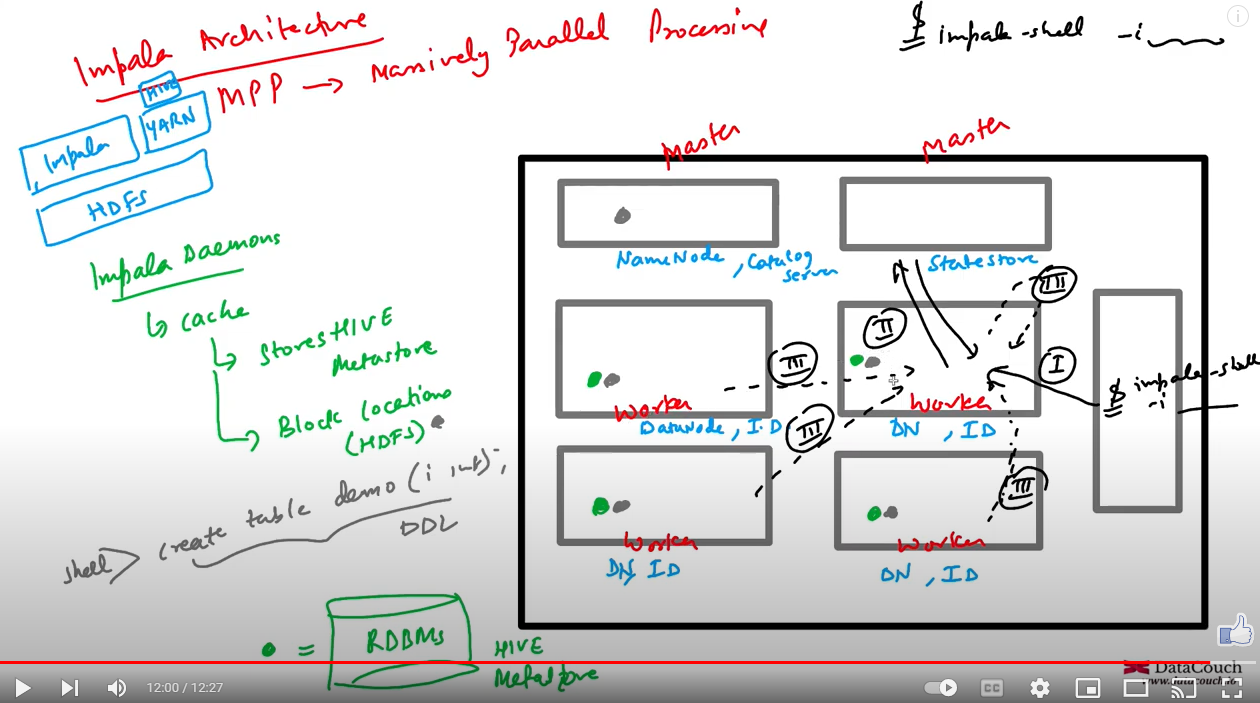
* invalidate metadata.
* refresh

Architecture of Impala

* It works on existing Master-slave architecture. but works like Peer-to-peer architecture.
* It will not use YARN to execute Map reduce jobs. It will directly execute Map reduce jobs on HDFS.
* Master nodes (Name node)
* Catalog Server
  + Catalog server responsible for rely on all metadata changes to all data nodes where Impalad deamon running.
  + Also, Co-Ordinator node will update metadata changes to HMS.
* State Store
  + It is responsible for checking the health of each Impalad.
* Worker nodes (Data node)
* Impalad
  + It accepts the queries from various interfaces and processes them.
  + Impalad deamon are having a Cache which helps to store HMS.
  + Cache also stores the block locations from HDFS. (All nodes)

How It works internally?

* When you are submitting the Impala query, the Co-Ordinator node pick up the job. It will parse the query and divide multiple smaller queries and execute those queries in respective Impalad deamon. Once processed, collect those results and combine it and share it to the client.



How to connect.?

* Impala-shell
* Hue interface
* ODBC/JDBC drivers

Important Commands

* Invalidate metadata hive\_db\_name.table\_name;
* Refresh <db\_name>.<table\_name> partition(key1=value1,key2=val2);
* Explain <query can be either select or insert or CTAS>
* Compute stats <database\_name>.<table\_name>(column1, column2 etc)
* History;
* help;
* desc <database\_name>;
* create table <table\_name>
* Show databases;

Tez

Introduction

* Tez is a DAG based execution engine which can be integrated with Hive and Pig for executing the jobs. Tez sits on top of YARN and executing its own optimized MR jobs on YARN.
* Tez creates a DAG plan and Tez AppMaster processes the DAG Plan by calling the Yarn Resource Manager to allocate the task containers. Finally, the Node Manager executes the tasks within the allocated container.
* Tez is suitable for large datasets. not suited for small datasets.

Why we need Tez.?

* Tez helps to resolve some limitations of MR
  + Frequent read and writes
  + overhead of job startups.
  + No Caching.

Why DAG based.?

* To avoid multiple Disk IO.

Pros of Tez

* Performance improvement of Hive and Pig when using Tez as execution engine
* Dynamic DAG Execution
  + Tez supports the dynamic creation of DAGs at runtime, providing flexibility for applications that need to adjust their execution plans based on evolving requirements.
* Shuffle and Sort Optimization
  + Mapper output is partitioned and sorted before sending to the corresponding reduce tasks.

Cons of Tez

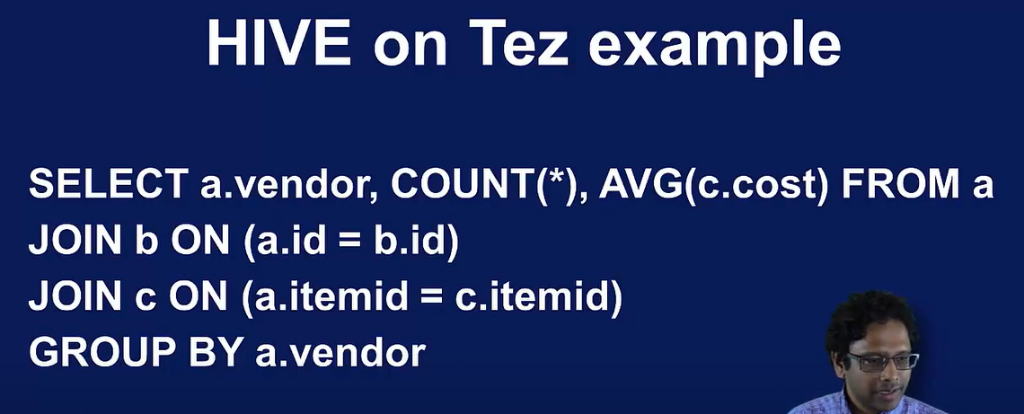
* Limited Use Cases for Small Datasets
  + Tez is optimized for large datasets. not for small datasets.
* Dependency on YARN
  + Tez relies on YARN for executing the tasks.

How it works internally.?

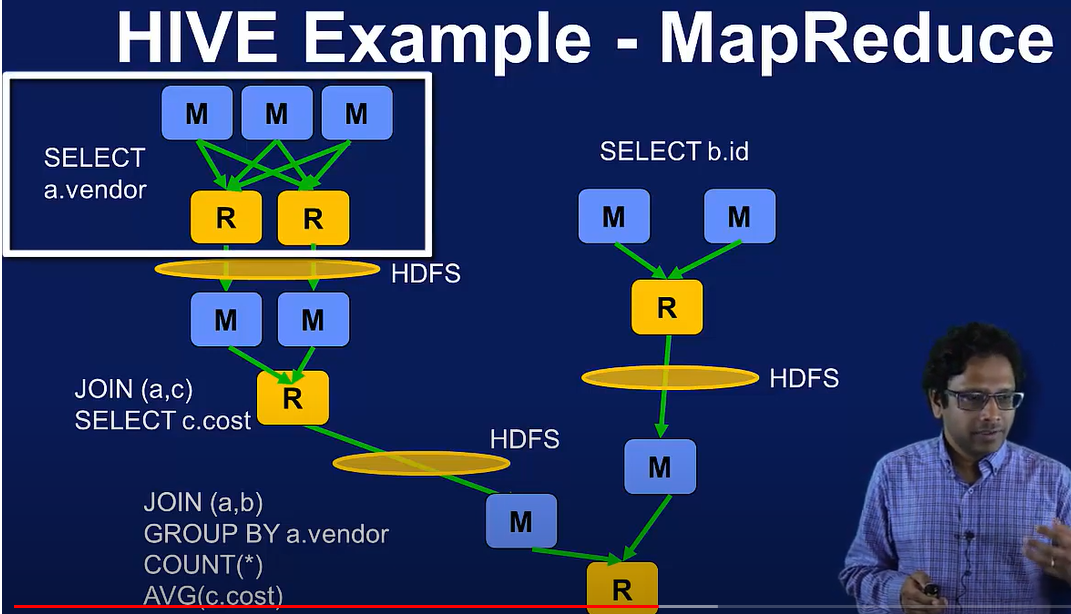
* Application Submission
  + User is submitting Tez application to YARN cluster. Tez application includes DAG that contains logical execution plan.
  + RM asks node manager to create container called Tez application master and driver program will be executed in the Tez application master.
* Tez Application Master
  + Tez application master negotiates resources with the RM and RM allocates containers to launch the tasks.
* DAG processing
  + Tez processes the DAG and converts physical execution plan
* Vertex Processing
  + Tez divides each vertex into tasks that run on individual nodes in the cluster.
* Tasks execution
  + Tez tasks run in containers allocated by YARN.
* Shuffle and Sort
  + During the shuffle and sort phase, output data from map tasks is partitioned and sorted before being sent to the corresponding reduce tasks.

Difference between Tez and Spark.?

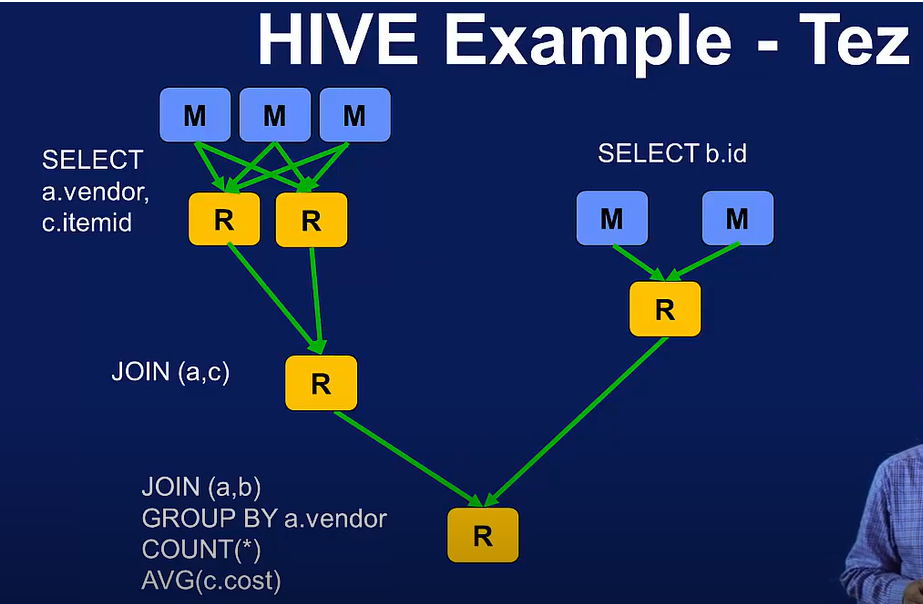
* Tez serves as an execution engine for Apache Hive and Apache Pig. Spark is having own execution engine to operate on data.
* Tez is primarily designed for batch processing. but Spark can support batch, streaming, Machine learning and graph processing.
* Tez optimizes data processing through efficient DAG execution and optimization of the shuffle and sort phase. but Spark optimizes data processing through in-memory processing, reducing the need for repetitive data reading from storage.
* Integration with Other Tools
  + Tez integrates with Hive and Pig. but Spark can integrate with various data sources, databases, and third-party tools.



Hive works with MapReduce



Hive works with Tez



Presto

Introduction

* Presto is a SQL query engine which is designed for processing on large datasets across multiple data sources. It uses different connectors to connect other data sources. Also, Supports Federated Query. Also, Presto can be integrated with external meta stores like Hive Metastore (HMS), Glue catalog. Also, Presto supports SQL Compatibility.
* Presto is a read-only system. Presto does not support traditional write operations like insert, update, or delete on data.
* Presto is a Batch processing Engine. not a real-time processing engine. We can use external system like Spark, Sqoop, Flink etc. for Data ingestion, Data transformations. It is not running Map reduce jobs.
* Presto process the data from In-memory processing.

why we need to go for Presto.?

* If we want to analyze multiple data sources like HDFS, RDBMS, No SQL, Cloud Data sources without move into one platform, then Presto would be good choice.
* Those data should be structured. not be a Semi-structured.
* If you don't want to be ACID transactions.

Pros of Presto

* Compatibility with Various Data Sources
* Presto is having Compatibility with Various Data Sources including HDFS, relational databases, cloud-based storage (e.g., S3, GCS), and more.
* Integration with BI Tools
  + Presto integrates with popular Business Intelligence (BI) tools such as Tableau, Looker, and others.
* Query Federation
  + Presto supports query federation, enabling users to query multiple data sources in a single SQL query.
* Dynamic Scaling
  + Presto supports dynamic scaling, allowing clusters to scale up or down based on demand.

Cons of Presto

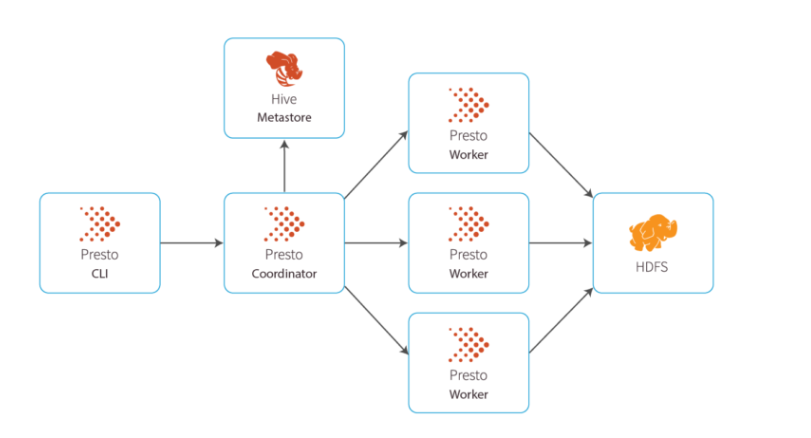
* Presto is not suitable for real-time streaming analytics.
* Presto is not suitable for Semi-structured and un-structured data.
* presto is not suitable for ACID transactions.
* Presto is primarily focused on SQL queries and analytics. While it can interact with systems that support machine learning (ML).
* Presto is not designed to be a full replacement for traditional relational databases.

Presto Architecture Components

* Client (Presto CLI) - Client helps to submit Presto Query into Co-Ordinator to get the results.
* Co-Ordinator - It is Master Deamon. It will Parse the query and Create Execution plan,
* Scheduler performs pipeline execution to Worker machine and Monitor the Progress.
* Worker
  + The coordinator assigns task to worker nodes.
  + The workers get actual data from the connector. Finally, the worker node delivers result to the client.
* Catalog
  + Presto Catalog contains Metadata information which can be where data stored, Table and Schema information.
  + Example - Hive Meta Store, Glue Data Catalog
* Connector
  + Connectors are nothing but Plugins. The connector provides metadata and data for queries.
  + Coordinator uses the connector to get metadata for building a query plan.

How Presto works internally.?

* Presto is a distributed system that runs on a cluster of nodes.
* Presto Client is submitting the SQL query to Co-Ordinator (Master Deamon). Co-Ordinator will parse the query and Create Execution Plan to Execute.
* Scheduler Connects through Execution Pipelines and Assign the work to Worker. Worker will perform the Query execution and give the results to Client.



Apache Drill

Introduction

* Drill is a Schema-Free SQL Query Engine which is designed to process large datasets from multiple data sources such as HDFS, RDBMS, No SQL Databases, Cloud Storages (S3, Blop and Cloud Storage). It uses a distributed execution engine to process SQL queries across multiple nodes in a cluster.
* Drill dynamically discovers the schema of data during query execution. Users can query datasets without define schema.
* Users can use different connectors (Plug-In) to connect different data sources.
* Supports both Structured and Semi-Structured datasets. Different file formats such as JSON, Parquet, Avro, and more.
* Follows Peer to Peer Architecture. no Master-Slave Architecture.
* There is no need to load data and transform it before Drill can process it or create and maintain schemas. Users can query raw data in almost any system, regardless of type.
* Drill is a distributed MPP query layer that supports SQL and alternative query languages against NoSQL and Hadoop data storage systems.
* It was inspired in part by Google's Dremel.
* Drill was originated at Google for data analytics

Why we need Drill.?

* If you want to process JSON data along with multiple data sources, then you can go for Drill.

Pros of Drill

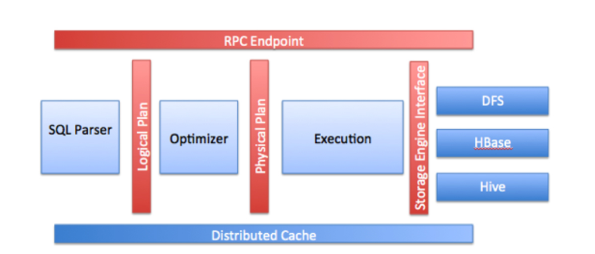
* Schema-Free Data Exploration
  + Drill allows users to explore and query data without the need to define a schema beforehand.
* Support for Various Data Formats
  + Including JSON, Parquet, Avro, and more.
* Nested Data Support
  + its ability to handle nested and complex data structures, such as JSON arrays and objects.
* Pluggable Storage Plugins
  + It can connect different data sources using different storage plugins.
* Self-Service Analytics
  + Users to explore and query data interactively without extensive preprocessing or data engineering efforts.

Cons of Drill: -

* Supports batch processing, not suitable for real time processing.
* limited support for update and delete operations.

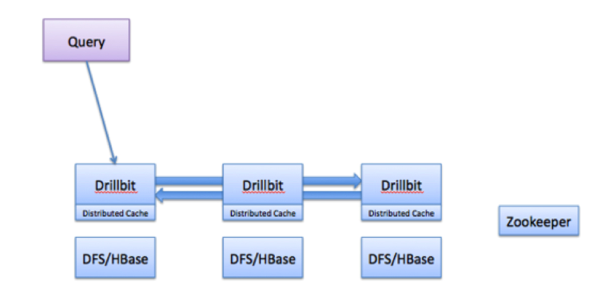
Core Components of Drill:-

* Drillbit
  + Drillbit is deamon process which runs of each node of the cluster. which is responsible for parsing and managing the query execution in distributed manner.
* Connectors (Storage plugins)
  + Connectors helps to connect and query various data sources.
* Metadata Store
  + Drill maintains a metadata store that stores metadata about the datasets being queried.
* SQL Parser
  + SQL parser will parse the query and (creates Logical plan) converted into the form which can processed by Query Planner and Query Execution Engine.
* Query Planner and Query Optimizer
  + Query planner helps to transform SQL queries into an execution plan (Physical plan) that can be distributed across the Drill cluster.
  + Query Optimizer analyzes query plans and optimizes them based on statistics and execution conditions.
* Query Execution Engine
  + Query Execution Engine helps to executing the query on the different nodes of the cluster.



How Drill works internally.?

* The Client is submitting the query request to Drillbit’ Zookeeper. One of the Drillbit in the cluster accepts the query and became foreman for this query. For the entire subsequent process, this Drillbit remains the master and has the responsibility of managing the execution of the query.
* SQL Parser will parse the query and create Logical execution plan and Query planner will convert them into Physical execution plan.
* Execution Engine will execute each sub task in corresponding node of the cluster and get the output.
* Foreman Drillbit will collect the all the output and merge and give it back to client.



Difference between Presto and Drill.?

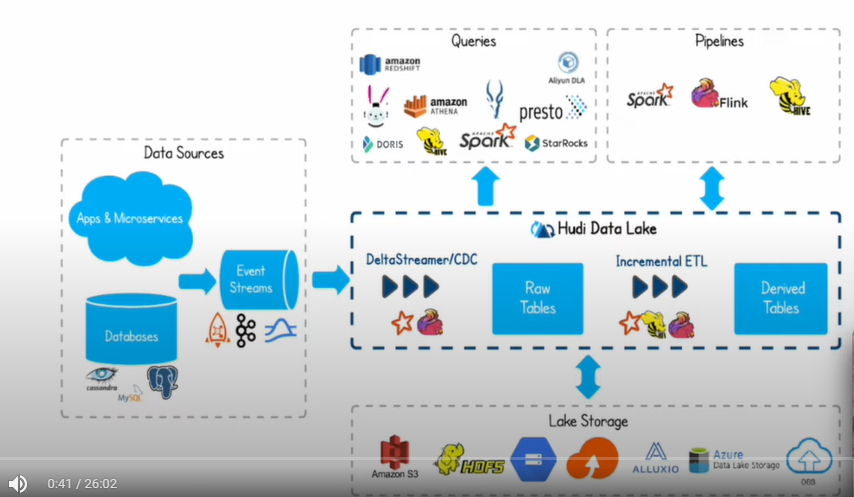
* Presto supports only structured data. but Drill supports Structured and Semi-Structured data.
* Presto has a schema-on-read approach where the data schema is defined during query execution. But Drill is having Schema free approach.
* Presto has a cost-based query optimizer that generates efficient execution plans. Drill uses a rule-based query optimizer. While it lacks a traditional cost-based optimizer.
* Presto is known for its high performance, especially in ad-hoc and interactive query scenarios. Drill also offers good performance, and its schema-free approach can be advantageous for certain use cases.

Apache Hudi

* Hudi is an open-source table format which provides a transactional support (Row level Update and Row level delete) on your data lake.
* Invented by Uber.

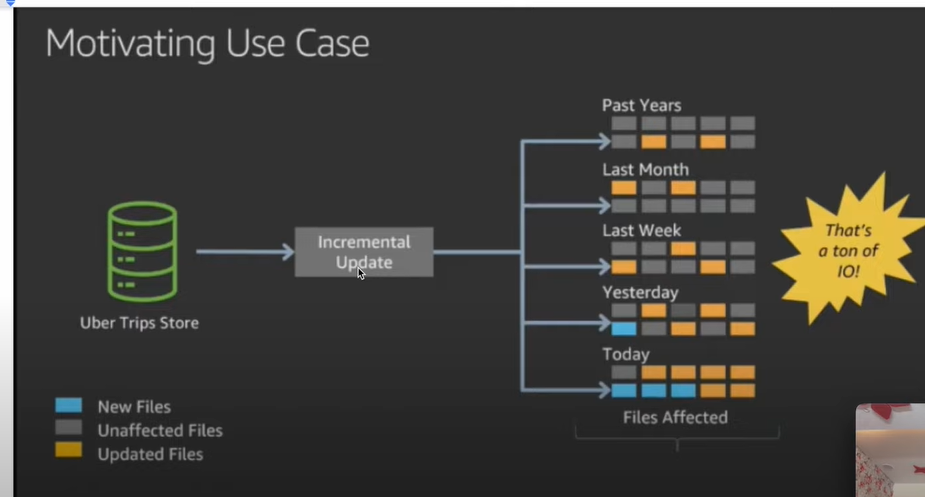
Why we need Hudi.?

* If you are running a business with a petabyte of data in your data lake and you want to update 2 records in the data set. It’s difficult to update from storage. Also, Delta records are keep coming from source. and Consumers are waiting for the Refreshed data.
* So, we have to build ETL data pipeline to solve this. In this ETL pipeline also, having issue that difficult to load large datasets and make changes. It will take time.
* Now a days, Consumer are expecting data in near real time. In this situation, Hudi helps to resolve this issue. It is supporting merge operation on the delta records with history data.



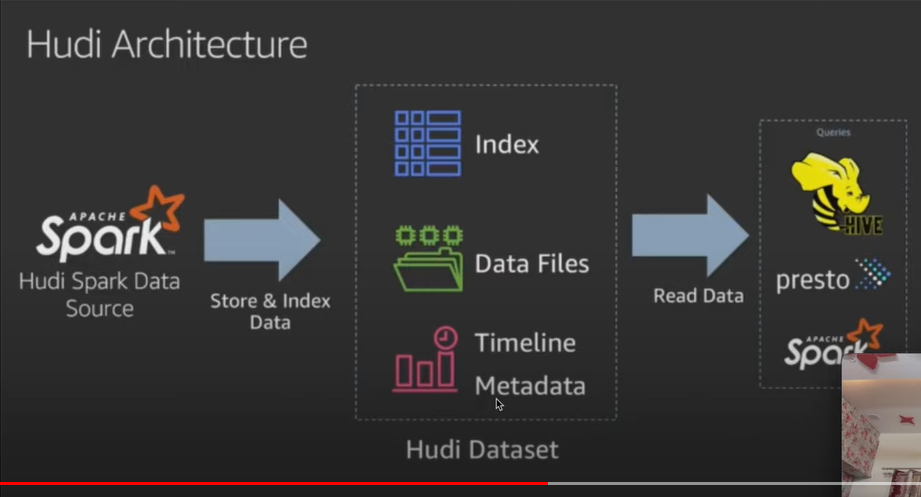
Problem with Uber Use case

* Uber stored data into data lake with distributed contains history data. Users want to give feedback for the yesterday trip. Lot of incremental data are coming. what happens usually, long running batch jobs will read all petabyte of data and recreate those petabytes of data for these small updates.



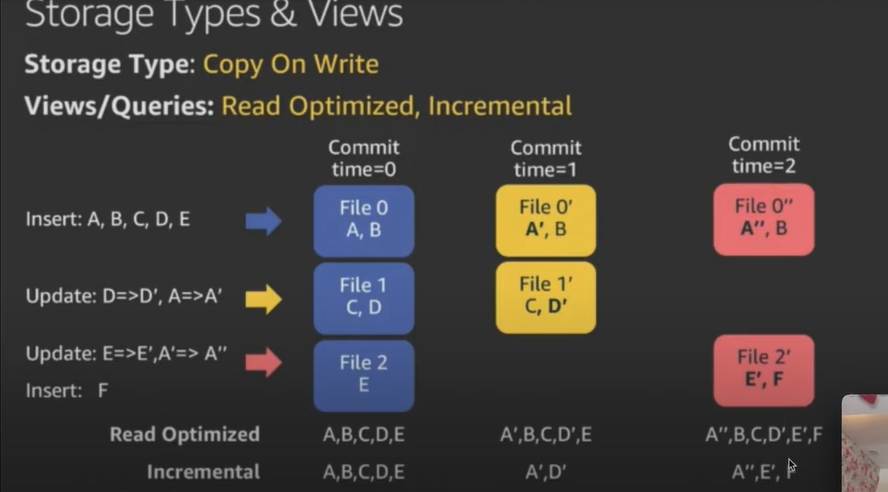
Hudi Architecture

* Components inside Hudi dataset
  + Index - By this Index, we can identify the records uniquely.
  + data files - actual data files.
  + Timeline Metadata - Hudi used to take Snapshot at timeline when you are committing the datasets or records. Ex: at T0, what are records are updated during the commit and T1, what are records are affected during the commit.
  + Hudi dataset can be consumed by Hive, Presto or Spark.

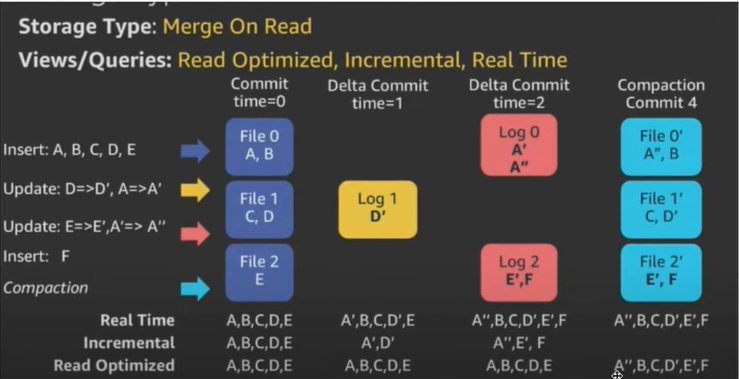


Storage Types

* Copy on Write (Read Heavy) – Default
  + In this model, when you are updating the data, it will copy existing data file and make changes to new file. it will not update the existing file.
  + Suitable for scenarios where data updates are infrequent.

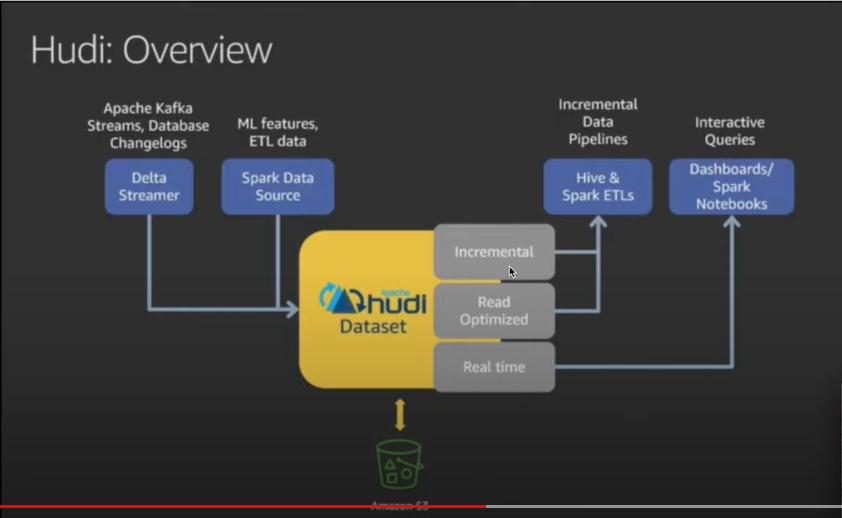


* Merge on Read (Write Heavy)
  + In this model, when you are updating the data, it will not copy existing data file and create new log file about the change. When you are doing compaction, Log files and data files are merged and have a new data file.
  + Suitable for frequent updates and incremental processing.



How Hudi works.?

* For Streaming, Delta Streamer will consume data from Streams data source and Ingest to Hudi datasets with Hudi formats. (Internally it stores parquet format in S3)
* For Batch, Spark will ingest to Hudi datasets. (Raw)
* On top of Hudi datasets, we can perform other ETL steps like data quality, transformations using Hive or Spark ETLs. or Dashboards or BI tools can directly consume data from Hudi datasets.



Apache Iceberg

Introduction

* Apache Iceberg is an open-source table format which helps to manage and track all the files that making a Table. It is kind of Abstraction layer between Physical data and Schema of the table.
* It is not a storage system or query engine. Just Table format.
* The actual data residing in the Iceberg table are Parquet, ORC file formats.
* Iceberg has been designed to resolve the various data inconsistency and performance issues that Hive suffers when data stored in S3. Because, Hive keeps track of data at the “folder” level (i.e. not the file level). This can lead to performance problems. But Iceberg is keep tracking the complete list of all files within a table using a persistent tree structure. Data file --> Manifest file --> Manifest list --> Metadata file

Data file

* Physical data file, which is in Text, Parquet, ORC, Avro formats etc.

Manifest file

* Manifest file contains metadata information’s about the data file like path of the file, format, Schema level information etc.

Manifest List

* Manifest list contains metadata information about the Manifest file like, Partition stats, data file counts.

Metadata file

* Metadata file contains metadata information about the Manifest list file such as location, manifest list file count
* Also, Table level metadata information’s such Schema, Partition details.

Iceberg Features

* Support ACID transaction
  + We can do insert/update and delete operation in Iceberg table.
* Schema Evolution
  + It is allowing users to add or remove columns from a Iceberg table without requiring a full rewrite of the data.
* Time Travel
  + It is allowing users to query data at specific points in time.
* Partitioning
  + It is allowing users to organize data within a table based on one or more partition columns. Partitioning improves query performance by restricting the amount of data that needs to be scanned for specific queries.
* Snapshot Isolation
  + which means the queries consistently read from a snapshot even if other transactions are in progress.

what is Table formats.?

* Table format is a metadata layer that allow tools to interact with data lake storage like a traditional database.

Difference between Hudi and Iceberg?

* Hudi supports both copy-on-write (COW) and merge-on-read (MOR) storage formats. Iceberg primarily uses a merge-on-read (MOR) storage format.
* Both Hudi and Iceberg supports schema evolution. we can update, delete or add columns seamlessly. Hudi does not store the history for Time travel. Iceberg stores history of changes for Time travel.
* Both Hudi and Iceberg supports ACID transactions.
* Hudi Performance is dependent on storage format (COW or MOR) and the specific use case. It is optimized for incremental processing scenarios. Iceberg performance based on time-travel queries with its snapshot-based architecture.

Use cases

* If you want time travel, then Iceberg will be choice. If you want to do ACID transactions, Incremental data processing, change data capture, and scenarios --> Hudi will be choice.

Difference between Delta and Iceberg.?

* Delta format tightly integrated with Spark only. But Iceberg format support various data processing frameworks.
* Delta format is maintained by Databricks and can be used for Databricks projects. But Iceberg is a open source table format can suits for all projects.
* Both delta and Iceberg supports schema evolution, allowing users to add, remove, or modify columns without affecting existing data. Delta tracking changes through transactional log file and Iceberg maintains history of schema changes.
* Delta performances are optimized for merge-on-write scenarios. Iceberg performance optimized for merge-on-read scenarios.

Apache Spark

Introduction

* Spark is a Processing tool, which is used for processing datasets in parallel across the cluster.
* Scala is a native language for spark

Why we go for Spark?

* Unified Engine for Batch and Stream - Hadoop needs separate tools for Batch and Stream processing. Hadoop is for Batch process and Storm is for Stream process. But in Spark Supports both batch and stream process.
* Speed - Normally Map reduce process the data from the disk, but not in memory. In Spark can process the data from the memory. If spark process the data from memory, then Spark will be 100 times faster than Map reduce. Suppose huge amount of data, memory will not be sufficient. So, it will process the data from disk, that time also spark will be 10 times faster than Map reduce.
* No Intermediate Storage - In Map reduce, Mapper output will be stored temp location of the mapper, and Second mapper will read the data from that location and so on. It will cause High disk IO, Time, Cost is more for processing, But in Spark, no intermediate storage happened. It will keep mapper output in in-memory, second mapper will read the data from that memory
* Interactive - I am running one query like Select \* from patient, it will run the map reduce job and again I am running the same job, and it will run again map reduce job. It is not good. It will fetch the result from previous result instead of running the job again. We solved this issue Map reduce by Distributed cache. But in spark it will be done very well.

Advantages of Spark

* Unified engine
* Processing Speed
* No Intermediate data storing
* Support Various Datasets and Various formats
* Various cluster managers (CM's)

Supported Language

Java, Scala, Python, R

Framework or Extension of Spark: -

1. Spark-SQL

2. Spark-Streaming

3. Graph-X

4. Mlib

How to Interact with Spark: -

Scala 🡪 spark-shell

Python 🡪 PySpark

RDD: - (Resilient Distributed Dataset):-

* RDD is a collection of records which are splitted as a partition across the cluster that can be operated in parallel.
* R 🡪 Resilient 🡪 Fault tolerance of RDD, It can be able to re-compute the failed partitions automatically
* D 🡪 Distributed 🡪 Data are distributed in Multiple nodes
* D 🡪 Dataset 🡪 Collection of records.

Three ways to create RDD

1. External Source

2. In-Memory (Parallelizing existing collection in driver program)

3. Another RDD

What is Spark Context

SparkContext is used to connect Spark cluster with help of Resource Manager.

A screenshot of a computer program

Description automatically generated

In-memory (Parallelize method)

* parallelize method used to create parent RDD from in-memory data.

A screenshot of a computer program

Description automatically generated

Feature of RDD

* Fault Tolerance
  + Spark will recompute the data automatically from failed RDDs.
* In-Memory Computation: -
  + Spark will Store the processed data in memory instead of disk.
* Lazy Evaluation
  + All the transformations are Lazy in Spark, because it will not compute any result when transformation is called, instead of that It will remember that transformation is applied to dataset. Transformation is computed only when Action is called.
* Partitioning
  + The RDD is splitted into no of partitions across the cluster, It will be operated in parallel.
* Persistence
  + We can also persist the RDD during our program execution.
  + We can persist RDD results in In-MMEORY or DISK using Different Storage levels.

what is Parent RDD?

RDD is created from source or in-memory is called as Parent RDD.

what is Pair RDD?

Create key, value pair RDD from normal RDD that become a Pair RDD.

RDD Operations

RDD supports two types of operations,

a) Transformations 🡪 Creating new RDD from existing RDD

b) Actions 🡪 Retrieve something from RDD

Transformations

* Transformation is a function which will take the RDD as a Input and produce one or more RDD as a output
* All the transformations are Lazy in Spark, because it will not compute any result when transformation is called, instead of that It will remember that transformation is applied to dataset. Transformation is computed only when Action is called.

Types of Transformations

1. Narrow Transformation: - (map(), flatMap(), filter(), mapPartition(), Sample(), Union)

One Child RDD partition is expecting the input from One Parent RDD partition is called as Narrow Transformation.

2. Wide Transformation: - (Shuffle Transformation) - check Shuffle Transformation

One Child RDD partition is expecting the input from Multiple Parent RDD partitions is called as Wide Transformation.

Various Transformations

1. map() 🡪 It iterates each element in the RDD and produces one output. one input and one output. Input and output data type may differ from each other.

A computer screen shot of a program code

Description automatically generated

filter

* It iterates each element in the RDD and produces one or more many elements
* It filters the element based on logic written in the function.

A computer screen shot of a computer code

Description automatically generated

flapMap

* It iterates each element in the RDD and produce multiple elements

difference between map() and flatMap

* Both map() and flapMap() iterates each element in the RDD, But the difference is map() returns one element and flapMap() returns many elements.

A screen shot of a computer code

Description automatically generated

mapPartitions(func)

* instead of iterates each element in RDD, it will be applied to each Partition level. It will be Iterated in entire RDD and produce the result.
* Ex: - Consider, one dataset is having 1000 rows and splitted as 10 partitions. so, each partition is having 100 rows.
* When you apply map(func) to the RDD, It will be applied to each row in the RDD. so it will be called for 1000 times. Time consuming is more in this case for large dataset.

A screen shot of a computer code

Description automatically generated

mapPartitionWithIndex

* It’s like mapPartitions(func) and extra thing it will return the result with Index, So we can come to know this result belongs to this particular RDD. If any issue with data, we can easily find out the Error RDD.

Union(dataset)

* We can get the elements of both RDD's into new RDD.
* Condition is Both RDD should be same type.

A computer screen shot of a computer code

Description automatically generated

intersection(dataset)

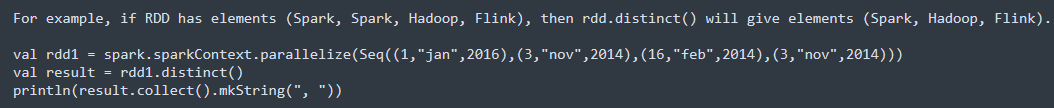
* We can get only the common element of both RDD into new RDD.
* Condition is Both RDD should be same type.

A computer code on a dark background

Description automatically generated

Distinct

* It returns only the distinct elements of the source RDD into new RDD.
* It helps to remove the duplicates in the RDD.



groupByKey([numTasks])

* It works only on Pair RDD. It grouping the records based on the key.
* The drawback is lot of unnecessary data is transferred over the network.

A computer code with white text

Description automatically generated

reduceByKey(func, [numTasks])

* grouping + Aggregation (Both groupByKey and reduceByKey gives the same result only. Performance wise there will be a difference.)
* It will be grouped and combined in the same machine before shuffled the data. It will shuffle less data when compared to groupByKey()
* It requires minimum two inputs.

A screen shot of a computer code

Description automatically generated

AggregatebyKey

* It is used to Aggregate the values based on the each key. but you can provide initial values when performing aggregation.
* same as reduceByKey, which takes an initial value.
* 3 parameters as input i. initial value ii. Combiner logic iii. sequence op logic

A computer code with white text

Description automatically generated

A screenshot of a computer program

Description automatically generated

Join

* The join () transformation will join two different RDDs on the basis of Key.
* Common elements from both RDDs will be coming as output. Just like INNER join from SQL.

A screen shot of a computer

Description automatically generated

Co-Group

* It will give all the elements of both RDDs like Full outer join.

A screen shot of a computer program

Description automatically generated

Coalesce

* It is used to reduce the no. of partitions.
* To Avoid the full shuffling, we use coalesce(), After used coalesce, Less data will be shuffled.



Re-Partitions

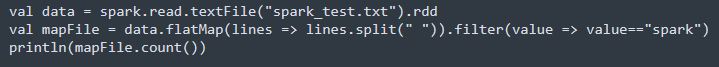
* Reshuffle the entire data in the RDD and create a new RDD. No of partitions may be less or more than previous RDD.
* Full shuffling will be happening over the network.

Actions

* Retrieve Something from RDD is called as Actions.

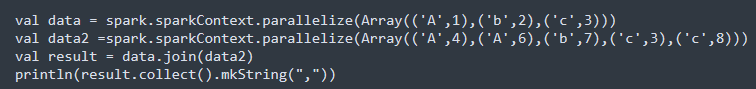
count

* Returns the No of elements in the RDD.



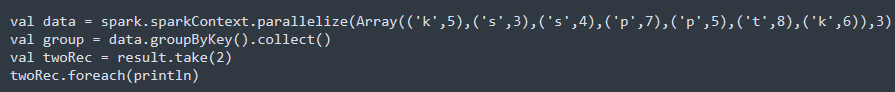
Collect

* Returns the elements in the RDD



Take

* Returns n number of elements in the RDD.



Top

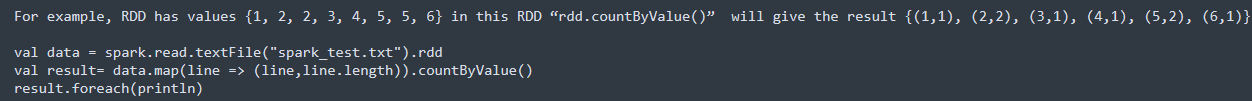
* Returns n number of elements with default ordering

A screen shot of a computer code

Description automatically generated

CountByValue

* It returns, how many times each element occur in the RDD.



Reduce

* It is used to Aggregate the elements of the Source RDD.
* It expects two parameters.

A screen shot of a computer

Description automatically generated

fold

* fold() action is like reduce() actions. but it takes the Zero value as input.

Difference between fold() and reduce()

* reduce() will throw error if the source is empty, But fold() will not throw any error due to its having an initial value.

A computer code with white text

Description automatically generated

Shared Variables

* The value in the variable will be shared across all the tasks. So, Spark supports two type of Shared Variables

a) Broadcast Variables 🡪 It used to copy the value in memory on all nodes

b) Accumulators 🡪 (That are only added to such as Counters and Sums)

Why we go for Broadcast Variables? (Closure Issue)

* If you have huge dataset, this dataset will be shipped to each spark node with closure, that’s need to be shared across all the tasks, If you have 100 tasks are running in 10 machines, 100 times that dataset need to be accessed and get processed. To avoid the closure issue, Broadcast Variables comes into picture.

Broadcast Variables

The value will be copied to memory of all nodes. Task will access the value from the memory of node.

A screen shot of a computer program

Description automatically generated

Accumulators

Accumulators are used for aggregating information across the executors.

A screen shot of a computer program

Description automatically generated

RDD Persistence: - (Caching and Persistence):-

* The RDD can be persisted using persist or cache method. First time, It is computed in an action and store it in memory, next time, It will take it from memory.
* In addition to that, each persisted RDD can be stored in a different storage level, It can be Memory, disk. These storage level can be set by passing storage level object to persist () method.
* Cache() method will be used as Default, It will store the RDD in memory only (StorageLevel.MEMORY\_ONLY) and it store deserialized objects in memory.

A screenshot of a computer program

Description automatically generated

How to Un-Persist the RDD

Spark monitors the cache of each node automatically and drop the old data partition in Least recently used manner.

We can also remove the cache manually by RDD.unpersist() method.

Benefits of RDD Persistence in Spark

* Time efficient
* Cost Efficient
* Less Execution Time

Difference between Cache () and Persistence ()

1. When we use cache () method, Default storage level is in MEMORY-ONLY mode.

2. When we use persist () method, we can use other Storage levels.

Cluster Manager

* Cluster Managers are used to manage the resources, Resources are nothing but memory and CPU.

Spark supports three Cluster Managers,

1. Standalone

2. YARN

3. Mesas

Spark Execution in YARN Cluster

1. Spark job is submitted to Resource manager, First Resource manager will check the security things, whether request is coming from authorized client or not, Then Resource manager negotiates resource from scheduler and Ask Node manager to create a process called as Application Master,

2. Resource manager will execute driver program in Application Master. Application master is sending a heartbeat to Application manger.

3. Application Master will negotiate resource from Scheduler and Scheduler will check Min max configurations, Resource manager contact node manager to start process called as Executor.

4. Executor will execute the task in Node manager. One executor will be created for one job in one node manager.

5. Once Executor job is completed, Next job is assigned to executor by Application Master. Executor sending heartbeat signal to Application Master.

6. All executor job is completed; Next stage executor task is launched. Then all stage tasks are completed, entire job is completed.

Note:

1. For every 10 Mints, Node manager sending heart beat signal to Resource manager.

2. If any task is failed in node manager, Application Master will contact Resource manager and create another container in another node manager. 3 times Application master launches the container, If 3 time task is failed, then Entire job will be failed.

3. If the Container is created, for 15 mints if any task is not allocated, then Node manager will revoke the container, Then Application Master will create container again.

A screen shot of a computer

Description automatically generated

What is Spark-SQL

* Spark-SQL is a extension of spark, which is used to working with Structured and Semi-Structured data and also We can query Big data with SQL.

Spark SQL provides three main capabilities

* DataFrame
* Read and Write data in different data formats.
* we can query data with SQL.

What is DataFrame? (Introduced in Spark 1.3)

* DataFrame is a collection of data with named columns.
* It is like Table in RDBMS.
* DataFrame can be created by External data file, Hive tables, External database or Existing RDD.
* DataFrame API is available in Java, Scala, Python and R.

What is Dataset.? (Introduced in Spark 1.6):-

* Dataset is an extension of DataFrame, which provides compile time Safety and also providing object-oriented programming.
* It uses Encoders for fast Serialization. instead of Java/Kyro Serialization.

Difference between RDD, DataFrame and Dataset

RDD API:- (2011) Spark 1.0

* It is a collection of data which is splitted as a partition across the cluster that can be operated in parallel.
* Fault tolerant and Immutable.
* Processing Structured and Semi-Structured data.
* Functional Transformation

Limitation of RDD

1. No Schema is associated with data.

2. Optimization should be done from User end

3. Reading from Multiple datasets is difficult.

4. Combining Multiple datasets is difficult.

DataFrame: - (2013) Spark 1.3

1. Distributed collection of data with named columns.

2. Fault tolerant and Immutable.

3. Processing Structured data.

4. Datasource API. Can connect Multiple Datasource.

5. Catalyst Query Optimization.

Limitation of DataFrame: -

1. Compile time Safety 🡪 We can't find the issue in Compile time.

2. Functional Programming API 🡪 you can use only default function available in Spark-SQL, if you want any other function, you can create UDF.

Dataset: - (2015) Spark 1.6

1. Dataset is an extension of DataFrame which provides type safety and object-oriented programming interface.

2. It uses Encoders for fast Serialization. RDD is having Default Serialization as Java Serialization,

3. It is Inter operable --> Easily convert dataframe into dataset.

Difference between Spark-Context and SparkSession:-

* Prior to Spark 2.0, SparkContext is used to connect Spark Cluster using Resource Manager.
* SparkConf is required to create SparkContext, which stores the configuration parameters like Application name, Executor memory, executor core etc. In order to use API of SQL, Hive and Streaming, Separate context is required.
* Spark 2.0, Spark Session is a Single point of Entry where we can do all Spark Functionalities and Spark programming with DataFrame and Dataset APIs. To Use, SQL, Hive and Streaming, no need to Create Separate Context, Spark Session includes all APIs.

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program

Description automatically generated

Running SQL Queries Programmatically

A screenshot of a computer program

Description automatically generated

Global Temporary View

* The Scope of the Temp view is per session, The Temp view will be disappear when the session terminates. If you want to Temp view to be shared among all sessions and keep alive until the spark application terminates. you can create Global Temp View.

A screen shot of a computer code

Description automatically generated