spark Theory Sources:

<https://www.dezyre.com/article/apache-spark-architecture-explained-in-detail/338>

<https://spark.apache.org/docs/2.1.1/api/scala/index.html#org.apache.spark.sql.Dataset>

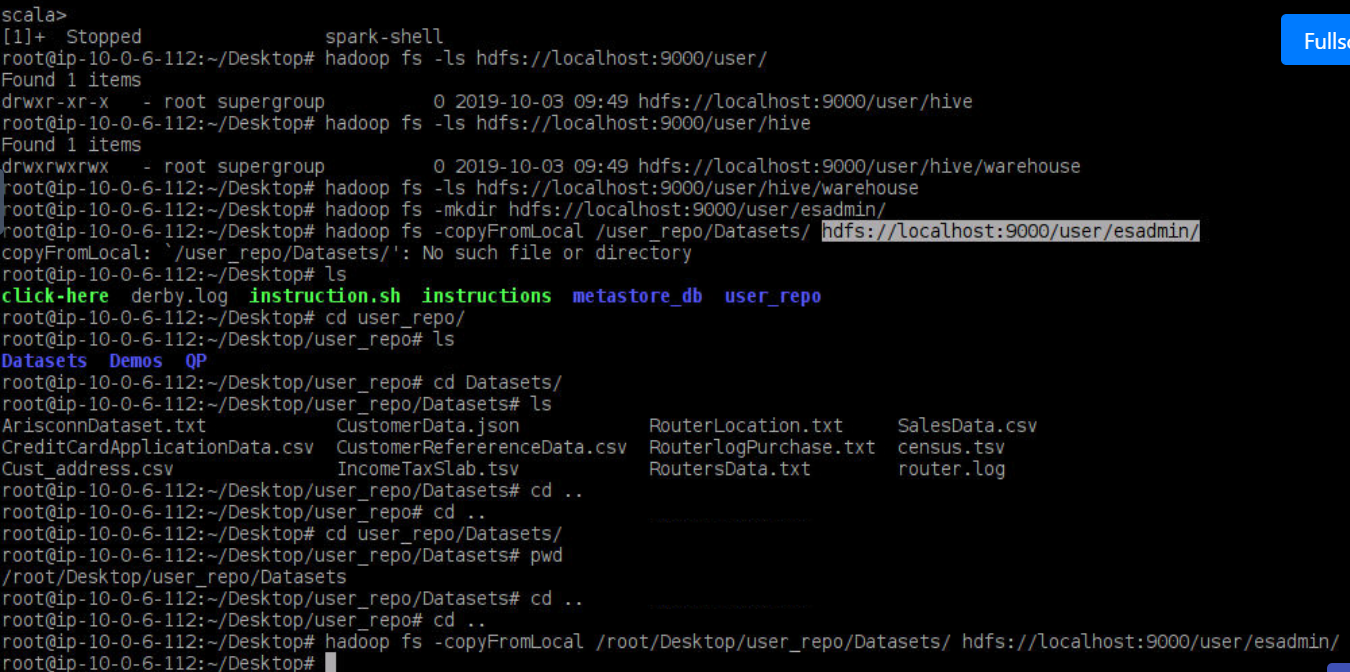
Hadoop Web UI : localhost:50070

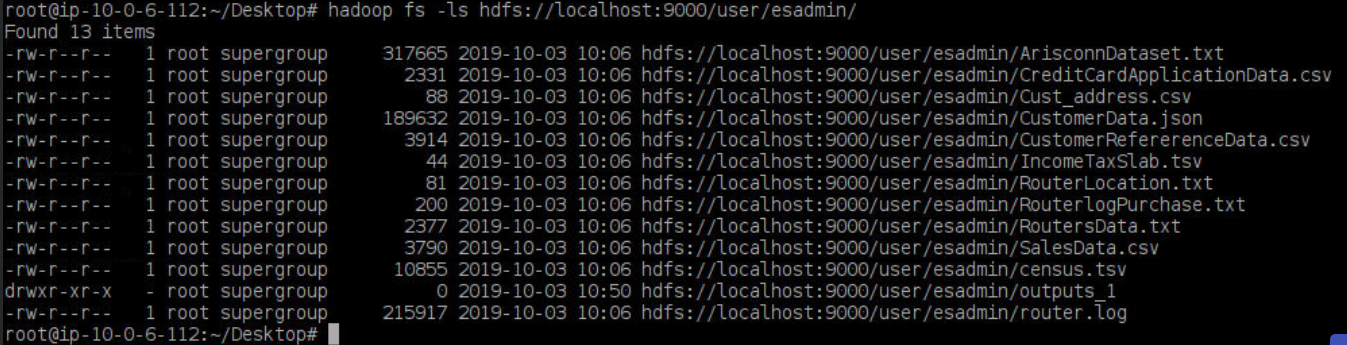
Driver runs on Master

Driver will have Sparkcontext and Worker is having Executor

The driver program that runs on the master node of the spark cluster schedules the job execution and negotiates with the cluster manager.

**What happens when a Spark Job is submitted**? When a client submits a **spark** user **application** code, the driver implicitly converts the code containing transformations and actions into a logical directed acyclic graph (DAG). ... The cluster manager then launches executors on the worker nodes on behalf of the driver.





To copy all files from local to hdfs::

Hadoop fs -copyFromLocal /root/Desktop/user\_repo/Datasets/\* hdfs://localhost:9000/user/esadmin/

To change datatype in rdd

After reading give map r=>r as below while splitting based on delimeter

Val rdd2 = rdd1.map(\_.split(“,”)).map(r=>r)

Val rdd3 = rdd2.map(r => r(3).toString)

Now we can filter based on condition

Question 1 Answer:

Create rdd using

Val rdd = sc.textFile(“hdfs:/file”)

Val rdd2 =rdd.map(\_.split(“,”))

Val rdd3 = rdd2.filter(\_.contains(“1995”)).count()

rdd3 – gives class

val out\_1 = Array(rdd3)

val ou\_1 = sc.parallelize(out\_1)

ou\_1.saveAsTextFile(“hdfs://localhost:/user/Output\_1”)

== copyting the file to local as out1

Hadoop fs -copyToLocal hdfs://localhost:/user/Output\_1/part-00000 /root/user/Outputs/out1

It creates out1 file with contents from Output\_1

Question-2

Val rdd = sc.textFile(“hdfs:/file”)

Val rdd2 =rdd.map(\_.split(“,”))

Val action\_c = rdd2.filter(\_.contains(“Action”)).count()

Val drama\_c = rdd2.filter(\_.contains(“Drama”)).count()

Val thriller\_c = rdd2.filter(\_.contains(“Thriller”)).count()

Val out\_arr = Array(“ACTION:”+action\_c,””DRAMA:”+drama\_c,”THRILLER:”+thriller\_c)

val ou\_1 = sc.parallelize(out\_arr)

ou\_1.saveAsTextFile(“hdfs://localhost:/user/Output\_1”)

== copyting the file to local as out1

Hadoop fs -copyToLocal hdfs://localhost:/user/Output\_2/part-00000 /root/user/Outputs/out2

Question -3

import spark.implicits.\_

import org.apache.spark.sql.types.\_

////// create sqlCOntextapwrite

val sqlContext = new org.apache.spark.sql.SQLContext(sc);

import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType}

import org.apache.spark.sql.Row

val customSchema = StructType(Array(

StructField("project", StringType, true),

StructField("article", StringType, true),

StructField("requests", IntegerType, true),

StructField("bytes\_served", DoubleType, true))

)

==== reading csv file

If header is true it will consider 1st row as columns names

If false it wont take 1st row

val cinema\_d = sqlContext.read.option("delimiter",",") .option("header", "false")

.schema(schema\_1).csv("hdfs://local/file.csv”)

Or

val cinema\_r = sqlContext.read.format(“csv”).option("delimiter",",") .option("header", "true")

.schema(schema\_2).load("hdfs://local/file.csv”)

Val ca = sqlContext.read.json(“hdfs/sc.json”)

Val ca = sqlContext.read.parquet(“hdfs/sc.json”)

cinema\_d.printSchema()

cinema\_r.printSchema()

==== creating temp table for Spark-SQL

cinema\_d.registerTempTable(“c\_d”);

cinema\_r.registerTempTable(“c\_r”);

df2.write.format("csv").save("/user/test/us")

Question-3 answer:

Val q\_3 = sqlContext.sql(“select cinemaid,count(\*) as numofpplwatched from rat\_tbl where cinemaid not like ‘null’ group by cinemaid having count(\*) == (select count(\*) from rat\_tbl where cinema\_id not like ‘null’ group by cinemaid order by count(\*) desc limit 1)”)

Q-4 answer:

Val q\_4 = sqlContext.sql(“select cd.cinema\_name, avg(cr.rating) as average from cinema\_d cd join cinema\_r cr on cr.cinema\_id = cd.cinema\_id where cd.genres Like ‘Action’ group by cd.cinema\_name order by avg(cr.rating) desc limit 1 ”)

Q\_5:

Val q\_5 = sqlContext.sql(“select cd.cinema\_name, avg(cr.rating) as average from cinema\_d cd join cinema\_r cr on cr.cinema\_id = cd.cinema\_id group by cd.cinema\_name order by avg(cr.rating) asc, cd.cinema\_name asc limit 10 ”)

# [Using DataFrame.coalesce to reduce the number of output files](https://stackoverflow.com/questions/40603259/using-dataframe-coalesce-to-reduce-the-number-of-output-files)

df.coalesce(1).

Val cl = sqlCOntext.read.parquet(“hdfs/parquet”)

cl.write.options(Map(“path” -> “hdfs://localhost:9000/user/hive/warehouse/CinemaData”)).saveAsTable(“SparkHiveCInemaTable”)

val hContext = new org.apache.spark.sql.hive.HiveContext(sc)

val cl = hContex.sql(“select \* from SparkHiveCInemaTable”)

scala> case class Person(id: Int, name: String)

defined class Person

Import spark SparkSession implicit Encoders:

scala> import spark.implicits.\_

import spark.implicits.\_

And use SparkSession to create an empty Dataset[Person]:

scala> spark.emptyDataset[Person]

case class output1(Number:Int)

import spark.implicits.\_

spark.emptyDataset[output1].toDF

val df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("/user/test/test.csv")

df.registerTempTable("temp")

val df2=sqlContext.sql("select \* from temp where state ='us')

val df3=sqlContext.sql("select \* from temp where state ='UK')

val df4=sqlContext.sql("select \* from temp where state ='London')

df2.write.format("com.databricks.spark.csv").save("/user/test/us")

df3.write.format("com.databricks.spark.csv").save("/user/test/UK")