Case Study : Instacart Domain: E-commerce

Instacart is a grocery ordering and delivery app that aims to make it easy to fill your refrigerator and pantry with your personal favorites and staples when you need them. Instacart's data science team plays a big part in providing this delightful shopping experience. Currently, they use transactional data to develop models that predict which products a user will buy again, try for the first time, or add to their cart next during a session.

The dataset is a relational set of files describing customers' orders over time. The dataset is anonymized and contains a sample of over 3 million grocery orders from more than 200,000 Instacart users.

Tasks:

As a Big Data consultant, you are helping the data science team to deal with a large amount of data. To solve this, you are pitching in to move the transactional data from RDBMS to HDFS by:

1. Verify the cluster health including HDFS and Spark

2. Test the spark environment by executing the spark's sort.py example.

3. Try to implement the same example in scala and perform spark-submit.

4. Analyze the behavior of spark application on Spark web UI

5. Add custom logs in your application and re-execute the application. Once executed check the Spark logs.

6. Transfer complete dataset from RDBMS to HDFS

7. Validate the loaded data by comparing the statistics of data both in source and HDFS

8. Create a new directory in HDFS named cheeses and load only rows where aisle is "specialty cheeses"

9. update "specialty cheeses" to "specialty cheese" and transfer only updated rows in the above-created directory.

Solution:

Task 1 : Verify the cluster health including HDFS and Spark - spark's HdfsTest.scala example

Location of HdfsTest.scala

/opt/cloudera/parcels/SPARK2-2.1.0.cloudera2-1.cdh5.7.0.p0.171658/lib/spark2/examples/src/main/scala/org/apache/spark/examples/HdfsTest.scala

Location of the Spark shell and the spark Submit Binary Files

/opt/cloudera/parcels/SPARK2-2.1.0.cloudera2-1.cdh5.7.0.p0.171658/lib/spark2/bin

After changing to bin directory , run

spark-shell

In spark -shell , then use the following command to copy paste the HdfsTest.scala

```
scala> :paste -raw
// Entering paste mode (ctrl-D to finish)

package org.apache.spark.examples

import org.apache.spark.sql.SparkSession


object HdfsTest {

  /** Usage: HdfsTest [file] */
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: HdfsTest <file>")
      System.exit(1)
    }
    val spark = SparkSession
      .builder
      .appName("HdfsTest")
      .getOrCreate()
    val file = spark.read.text(args(0)).rdd
    val mapped = file.map(s => s.length).cache()
    for (iter <- 1 to 10) {
      val start = System.currentTimeMillis()
      for (x <- mapped) { x + 2 }
      val end = System.currentTimeMillis()
      println("Iteration " + iter + " took " + (end-start) + " ms")
    }
    spark.stop()
  }
}

// Exiting paste mode, now interpreting.
```

Then , ran the following command on the shell to execute the Scala Script.

```
scala> HdfsTest.main(Array("AppleStore.csv"))
Iteration 1 took 6142 ms
Iteration 2 took 79 ms
Iteration 3 took 41 ms
Iteration 4 took 51 ms
Iteration 5 took 27 ms
Iteration 6 took 26 ms
Iteration 7 took 30 ms
Iteration 8 took 26 ms
Iteration 9 took 25 ms
Iteration 10 took 26 ms
```

Then same HdfsTest example in python was also run to check HDFS and Spark.

Command Used :

```
./spark-submit                    /mnt/home/edureka_960126/HdfsTest.py
/user/edureka_960126/AppleStore.csv
```

HdfsTest.py

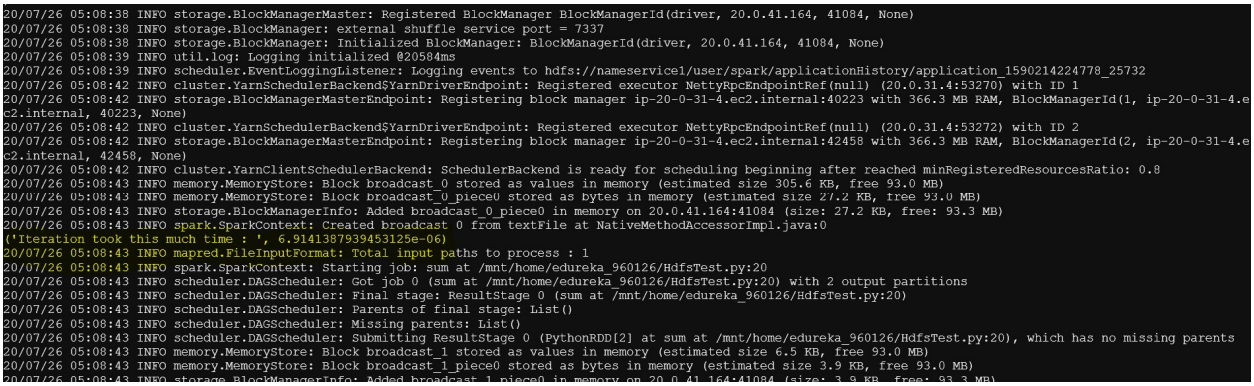
```
import sys
import os
import random
from operator import add, mul
from pyspark import SparkContext, SparkConf
from pyspark import SparkFiles
import time

def main(arg):
   file = arg
   conf = SparkConf().setAppName("HdfsTest")
   sc = SparkContext(conf=conf)
   rdd = sc.textFile(file).map(lambda line: len(line))

   for iter in range(10):
      start=time.time()
      rdd.map(lambda x : x+2)
      duration=time.time()-start
      print("Iteration took this much time : ", duration)
      print("Returned length(s) of: " , rdd.sum())


if __name__=="__main__":
   main(sys.argv[1])
```

Above is the file used to submit the job and the following results were obtained :



Task 2 and 3 :

a. Test the spark environment by executing the spark's sort.py example. b. Try to implement the same example in scala and perform spark-submit.

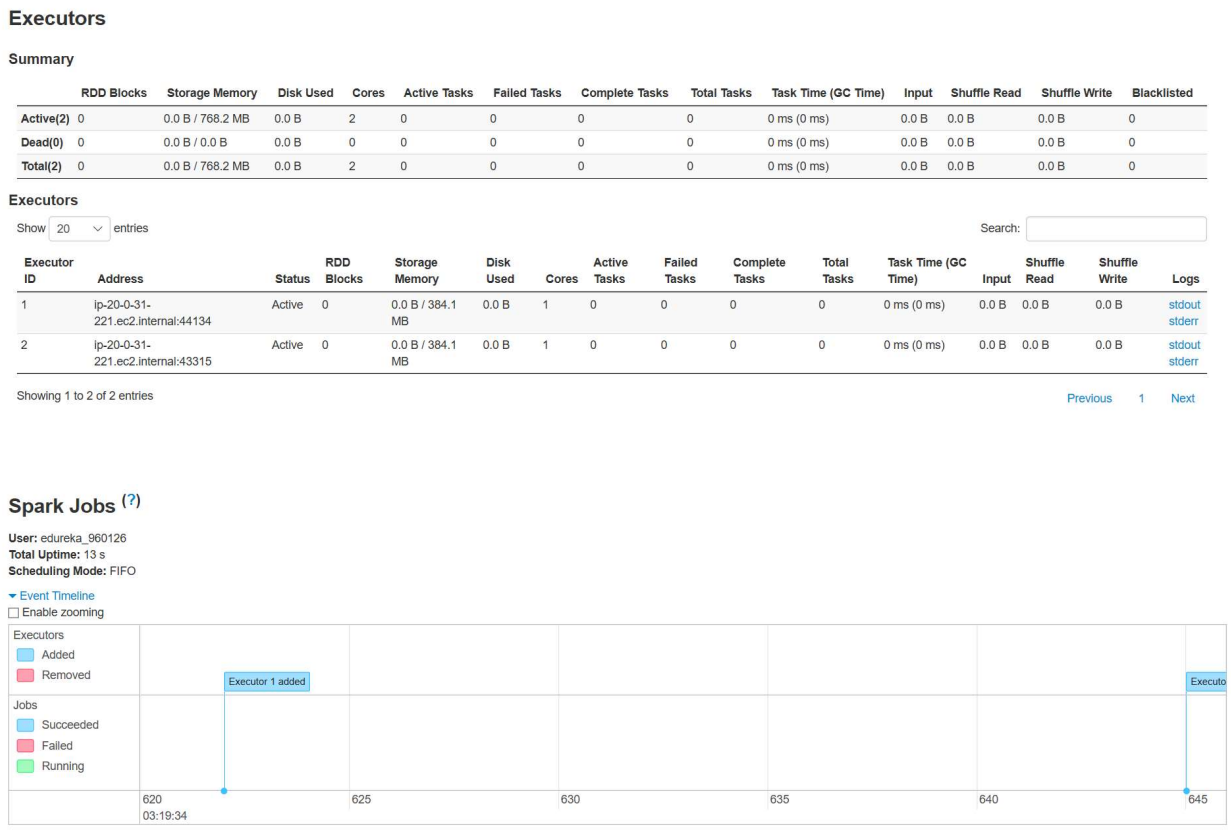
Command Used :

```
spark-submit                    /mnt/home/edureka_960126/sort.py
/user/edureka_960126/input_sort_py.txt
```

```
20/07/27 03:19:38 INFO execution.FileSourceScanExec: Planning scan with bin packing, max size: 4194304 bytes, open cost is considered as scanning 4194304 bytes.
----> Iteration took: 2.2172927856e-05 ms
----> Iteration took: 8.10623168945e-06 ms
----> Iteration took: 5.96046447754e-06 ms
----> Iteration took: 5.96046447754e-06 ms
----> Iteration took: 5.00679016113e-06 ms
----> Iteration took: 5.96046447754e-06 ms
----> Iteration took: 5.00679016113e-06 ms
----> Iteration took: 5.00679016113e-06 ms
----> Iteration took: 4.05311584473e-06 ms
----> Iteration took: 5.00679016113e-06 ms
```

So our environment is working perfectly.

Task 4: Analyze the behavior of spark application on Spark web UI



Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(2)	0	0.0 B / 768.2 MB	0.0 B	2	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(2)	0	0.0 B / 768.2 MB	0.0 B	2	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
1	ip-20-0-31-221.ec2.internal:44134	Active	0	0.0 B / 384.1 MB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr
2	ip-20-0-31-221.ec2.internal:43315	Active	0	0.0 B / 384.1 MB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr

Showing 1 to 2 of 2 entries Previous 1 Next

Stages for All Jobs

Analysis :

From the above it can be seen that stages screen is empty and no DAG is shown.
The reason for this is that DAG is formed only when action is called and because only the transformations were there in the sort.py so executors were assigned but no work was done by these executors as the action was not called.

Task 5. Edit the application and add custom logs. Once executed check the Spark logs.

spark.eventLog.dir is the parameter to set the path where the logs are stored for the spark jobs by default.

Parameter set in our environment is :

hdfs://nameservice1/user/spark/applicationHistory

Using the below command we can see the logs on the console :

```
hdfs dfs -cat hdfs://nameservice1/user/spark/applicationHistory/application_1590214224778_25728
```

We can create the Custom logs for our application :

1. We can change the directory path where we want to store the logs instead of default directory path in the configuration:
   ./spark-submit --conf "spark.eventLog.dir=/mnt/home/edureka_960126/" /mnt/home/edureka_960126/HdfsTest.py /user/edureka_960126/AppleStore.csv
   This was verified and the logs were stored in the desired directory.

2. We can store the custom things in the log files :

Spark uses Log4j which is standard for Logging for various Java applications.

log4j.properties

- Logging Levels
  - Define Priority Order of logged information
- Appenders
  - Define the resources to publish to
  - Define the log level threshold above which to log for each resource
  - Attributes of each appender depend on the type of appender
- Loggers
  - Create logged information
  - Define custom loggers for selected packages and classes
  - Set the log level for each selection
  - Set the appenders to use for each selection
- RootLogger
  - Defines the base logging level for all appenders
  - Only one root logger
  - All loggers inherit attributes from root and override root attributes

Logging Levels are the levels of essentiality we want logs to be collected.

Appenders are the one who publishes to various interfaces like files , e mails etc.

Loggers are the ones who collect the logs from the application.

Root Logger is the default and other Loggers inherit from the Root Logger.

Logging Levels

Log Level	Priority Order
FATAL	High
ERROR	
INFO	
WARN	
DEBUG	Low

Custom log4j.properties file was created and was submitted with spark submit for being used.

Command Used :

./spark-submit --conf "spark.driver.extraJavaOptions=-Dlog4j.debug=true" --conf "spark.driver.extraJavaOptions=-Dlog4j.configuration=log4j.properties" --files /mnt/home/edureka_960126/log4j.properties /mnt/home/edureka_960126/HdfsTest.py /user/edureka_960126/AppleStore.csv

Uploading resource file:/mnt/home/edureka_960126/log4j.properties -> hdfs://nameservice1/user/edureka_960126/.sparkStaging/application_1590214224778_26194/log4j.properties

Got such in the logs in the console which means that our log4j.properties was uploaded to the executors to be used.

Note in this case study I have used the same concept for demonstrating custom logging.

Task 6 :Transfer the sample dataset from RDBMS to HDFS

Table was created with the schema of the file :

In our present case table named aisles and products are already there in the labuser_database .

```
MySQL [labuser_database]> select * from aisles limit 10 ;
+----------+----------------------------+
| aisle_id | aisle                      |
+----------+----------------------------+
|        0 | aisle                      |
|        1 | prepared soups salads      |
|        2 | specialty cheese           |
|        3 | energy granola bars        |
|        4 | instant foods              |
|        5 | marinades meat preparation |
|        6 | other                      |
|        7 | packaged meat              |
|        8 | bakery desserts            |
|        9 | pasta sauce                |
+----------+----------------------------+
10 rows in set (0.01 sec)

MySQL [labuser_database]> select * from products limit 10 ;
+------------+---------------------------------------------------------------+----------+---------------+
| product_id | product_name                                                  | aisle_id | department_id |
+------------+---------------------------------------------------------------+----------+---------------+
|          0 | product_name                                                  |        0 |             0 |
|          1 | Chocolate Sandwich Cookies                                    |       61 |            19 |
|          2 | All-Seasons Salt                                              |      104 |            13 |
|          3 | Robust Golden Unsweetened Oolong Tea                          |       94 |             7 |
|          4 | Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce |       38 |             1 |
|          5 | Green Chile Anytime Sauce                                     |        5 |            13 |
|          6 | Dry Nose Oil                                                  |       11 |            11 |
|          7 | Pure Coconut Water With Orange                                |       98 |             7 |
|          8 | Cut Russet Potatoes Steam N' Mash                             |      116 |             1 |
|          9 | Light Strawberry Blueberry Yogurt                             |      120 |            16 |
+------------+---------------------------------------------------------------+----------+---------------+
10 rows in set (0.02 sec)
```

Then Data was transferred from these tables to the HDFS.

```
MySQL [labuser_database]> create table products_aisles as select p.product_id , p.product_name , p.aisle_id , p.department_id , a.aisle from products as p inner join ai
sles as a  on  p.aisle_id = a.aisle_id ;
Query OK, 49356 rows affected (15.83 sec)
Records: 49356  Duplicates: 0  Warnings: 0

MySQL [labuser_database]> select count(*) from products ;
+----------+
| count(*) |
+----------+
|    49356 |
+----------+
1 row in set (0.01 sec)

MySQL [labuser_database]> select * from products_aisles limit 10 ;
+------------+---------------------------------------------------------------+----------+---------------+----------------------------+
| product_id | product_name                                                  | aisle_id | department_id | aisle                      |
+------------+---------------------------------------------------------------+----------+---------------+----------------------------+
|          0 | product_name                                                  |        0 |             0 | aisle                      |
|          1 | Chocolate Sandwich Cookies                                    |       61 |            19 | cookies cakes              |
|          2 | All-Seasons Salt                                              |      104 |            13 | spices seasonings          |
|          3 | Robust Golden Unsweetened Oolong Tea                          |       94 |             7 | tea                        |
|          4 | Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce |       38 |             1 | frozen meals               |
|          5 | Green Chile Anytime Sauce                                     |        5 |            13 | marinades meat preparation |
|          6 | Dry Nose Oil                                                  |       11 |            11 | cold flu allergy           |
|          7 | Pure Coconut Water With Orange                                |       98 |             7 | juice nectars              |
|          8 | Cut Russet Potatoes Steam N' Mash                             |      116 |             1 | frozen produce             |
|          9 | Light Strawberry Blueberry Yogurt                             |      120 |            16 | yogurt                     |
+------------+---------------------------------------------------------------+----------+---------------+----------------------------+
10 rows in set (1.50 sec)
```

New table was created with joining of aisles with products.

Now transferring this Data to HDFS.

sqoop import --connect jdbc:mysql://dbserver.edu.cloudlab.com/labuser_database --username edu_labuser --password edureka --table products_aisles --target-dir /user/edureka_960126/cs2_mod4_data_delim --fields-terminated-by , --escaped-by \\ --enclosed-by '\"'

```
20/07/27 04:17:23 INFO mapreduce.ImportJobBase: Transferred 2.7681 MB in 21.3453 seconds (132.7936 KB/sec)
20/07/27 04:17:23 INFO mapreduce.ImportJobBase: Retrieved 49356 records.
```

```
        Map-Reduce Framework
                Map input records=49356
                Map output records=49356
                Input split bytes=485
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=308
                CPU time spent (ms)=6030
                Physical memory (bytes) snapshot=1100181504
                Virtual memory (bytes) snapshot=11261673472
                Total committed heap usage (bytes)=1807220736
                Peak Map Physical memory (bytes)=281653248
                Peak Map Virtual memory (bytes)=2819002368
        File Input Format Counters
```

7. **Validate the loaded data by comparing the statistics of data both in source and HDFS**

Statistics are validated as the number of rows in MYSQL Table and the HDFS records in the directory are same.

8. **Create a new directory in HDFS named cheeses and load only rows where aisle is "specialty cheese"**

```
MySQL [labuser_database]> select count(*) from products_aisles where aisle = 'specialty cheese' ;
+----------+
| count(*) |
+----------+
|      268 |
+----------+
1 row in set (0.01 sec)
```

Command Used :

sqoop import --connect jdbc:mysql://dbserver.edu.cloudlab.com/labuser_database --username edu_labuser --password edureka --table products_aisles --target-dir /user/edureka_960126/cs2_mod4_cheese_delim_ --where "aisle = 'specialty cheeses'" --fields-terminated-by , --escaped-by \\ --enclosed-by '\"'

```
20/07/27 04:34:33 INFO mapreduce.ImportJobBase: Transferred 14.166 KB in 20.7286 seconds (699.8068 bytes/sec)
20/07/27 04:34:33 INFO mapreduce.ImportJobBase: Retrieved 268 records.
```

9. **Update “specialty cheeses” to “specialty cheese and transfer only updated rows in the above-created directory.**

```
MySQL [labuser_database]> update products_aisles set aisle='specialty cheeses' , timestamp = CURRENT_TIMESTAMP() where aisle='specialty cheese';
Query OK, 268 rows affected (0.04 sec)
Rows matched: 268  Changed: 268  Warnings: 0
```

New column timestamp was added for the last modified purpose.

Update the “specialty cheeses” to “specialty cheese”

```
MySQL [labuser_database]> update products_aisles set aisle='specialty cheeses' , timestamp = CURRENT_TIMESTAMP() where aisle='specialty cheese';
Query OK, 268 rows affected (0.04 sec)
Rows matched: 268  Changed: 268  Warnings: 0
```

Then , sqoop incremental import was used for importing the rows into HDFS directory.

sqoop import --connect jdbc:mysql://dbserver.edu.cloudlab.com/labuser_database --check-column timestamp --username edu_labuser --password edureka --table products_aisles --target-dir /user/edureka_960126/cs2_mod4_cheese_delim_ --incremental lastmodified --last-value 2020-07-27 04:40:00 --fields-terminated-by , --escaped-by \\ --enclosed-by '\"' –merge-key ‘product_id’

Note for the first time the last modified is taken just ahead of the timestamp when all the rows were put first time in RDBMS. After each updation the timestamp change.

And with SQOOP also the last value is provided so that in next round of import we can use that value but as we are doing the first round of import we used the 2020-07-27 04:40:00 just ahead of the 2020-07-27 04:39:50 when I updated the timestamp in all my rows of the table which are the initial timestamp of putting records in RDBMS.

New values for next last modified import.

```
20/07/27 06:06:46 INFO tool.ImportTool: Incremental import complete! To run another incremental import of all data following this import, supply the following arguments

20/07/27 06:06:46 INFO tool.ImportTool:  --incremental lastmodified
20/07/27 06:06:46 INFO tool.ImportTool:   --check-column timestamp
20/07/27 06:06:46 INFO tool.ImportTool:   --last-value 2020-07-27 06:05:38.0
20/07/27 06:06:46 INFO tool.ImportTool: (Consider saving this with 'sqoop job --create')
```

Important points regarding the two above SQOOP tasks:

1. As we have commas in our data fields, we have to properly delimit them so as to identify a particular field because by default the fields are ,(comma) delimited . So parsing of fields cannot be done when fields already have comma.

So , we used below with Sqoop import :

--fields-terminated-by , --escaped-by \\ --enclosed-by '\"' –merge-key 'product_id'

This delim parameters in the command were used again while importing and merging the updated records.

2. Note that here we have merged the rows in the HDFS Directory with the newly updated ones in the RDBMS . As we have to merge the new rows in the existing HDFS folder (the updated records) , we have used merge option while importing.

Using –merge-key 'product_id' (key to be provided)

This is a two step process where in the first step only mappers work for selection

First step :

Second step :

Mappers and reducers both work because selection of new rows from RDBMS (mappers)

and then merging them with the existing data in the HDFS folder and this is a reduction process.

(mappers and reducers)

----------NOT PART OF ASSIGNMENT----------
----------SPARK NOTES MADE AFTER LEARNING the topics , WHICH WERE USED IN PERFORMING THE ASSIGNMENT----------

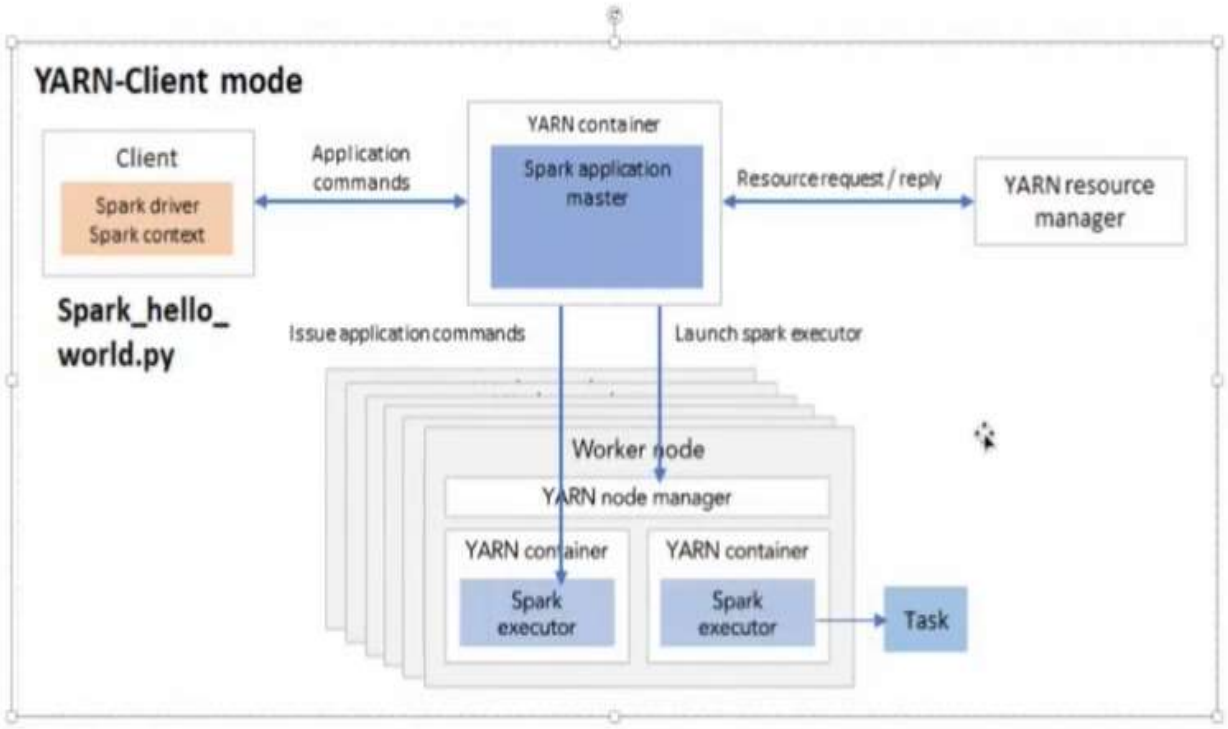
Brief Theory about Spark is presented below:

Spark is deployed in two modes :

1. **Client Mode**
2. **Cluster Mode**

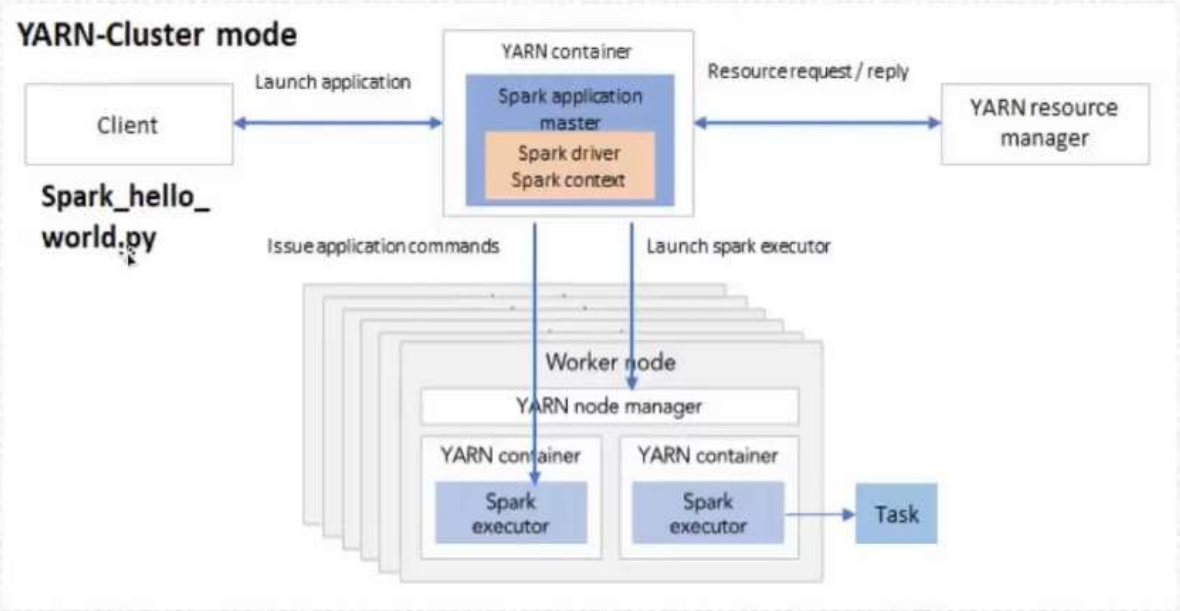
Sequence of Steps in Client Mode :-

1. When you open the Spark Shell , then SparkSession as spark and SparkContext as sc are by default available. These are created by Spark Driver present in your machine.
2. This Spark Context is used to send the application commands to the Spark Application Master running in the cluster which in turn will negotiate for resources with YARN resource Manager.(Note that this Spark Application Master is just a program launched in a yarn container (a JVM machine))
3. Based on negotiation , our application Master will launch executors which are JVM only through the YARN Node Manager which are present there in the worker nodes (or just nodes in the Yarn Cluster)



4. Then further issue commands to those Spark Executors.

Sequence of Steps in Cluster Mode :-

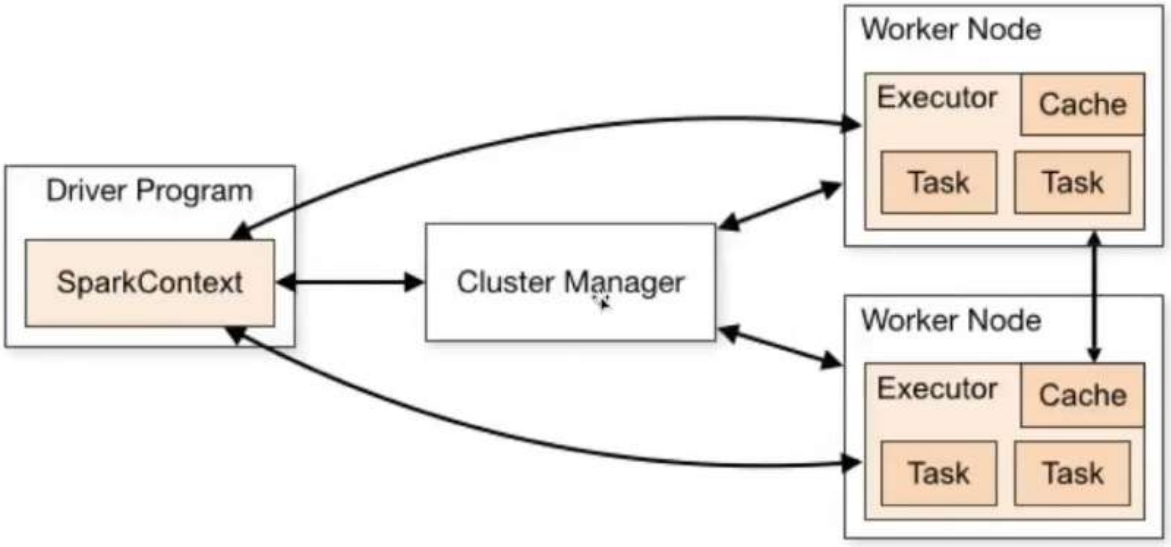


1. Client Launch application
2. Assuming here the YARN Cluster , in one of the Yarn Container in one of the nodes of the cluster , Spark Application Master gets created(or may be already running so connection between the client and the spark application Master is there)
3. Spark Application Master itself create the SparkContext or Session and Spark Driver Program further negotiate for resources with Yarn resource Manager to launch the executors.

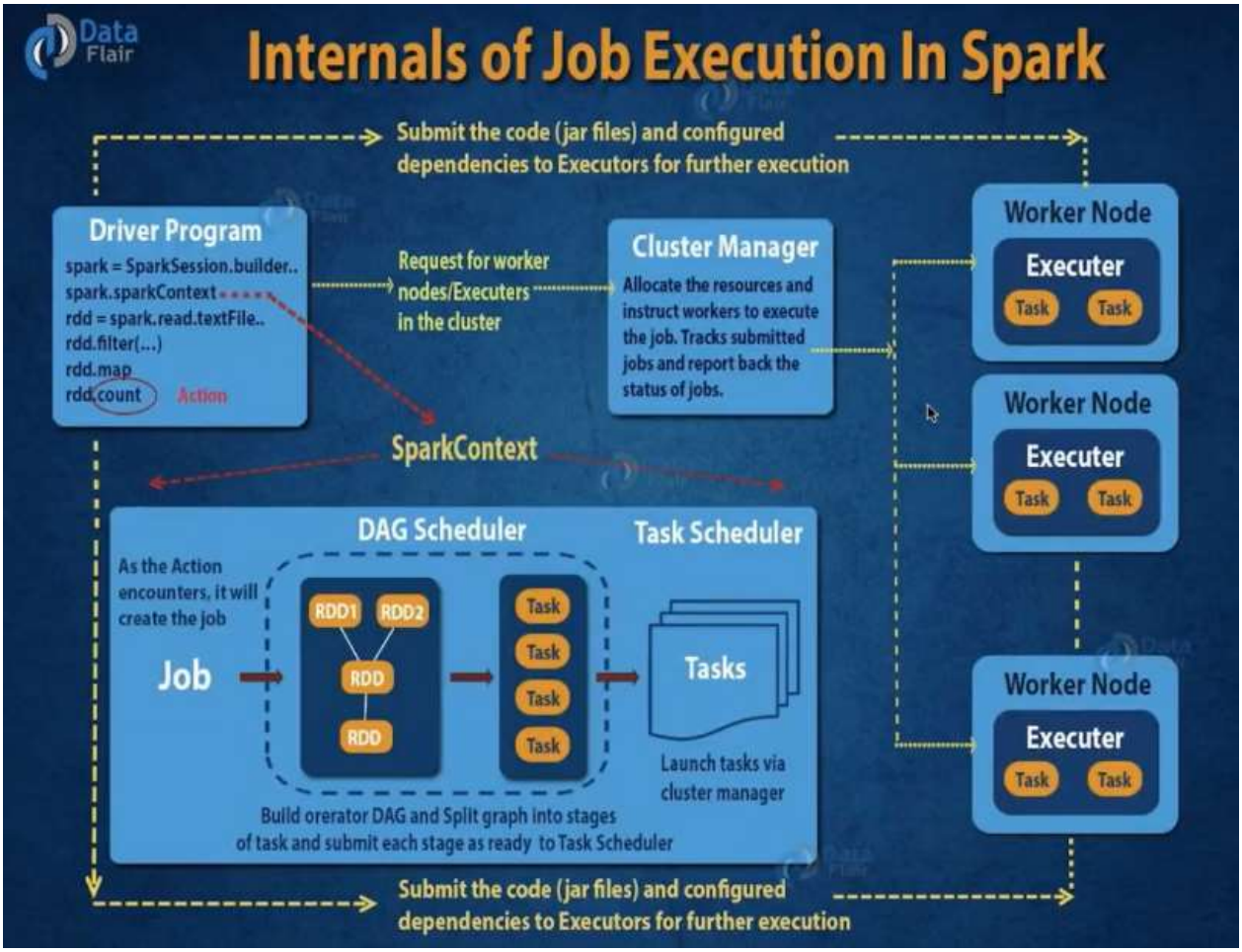
When we submit our job through the Spark Shell then we use the Client Mode .

General Steps in case of Spark execution are :-

1. Spark Driver creates a Spark Context
2. Spark Context establishment is the establishment of connection with the Cluster Manager
3. Driver will ask for resources to the Cluster Manager and create executors in containers available in the other nodes in the cluster which are hereby called worker nodes.
4. So, the Driver provides a task (code/compute + Data) to Cluster manager(here Yarn) , Yarn decides decide where(i.e. in which worker node /executor) Data resides and in turn will allocate memory etc. resources to the executor to execute the task.
5. Note that the Driver and Yarn has an intermediary of the **Application Master and hence Driver communicate with the Yarn through the Application Master.**



Above is the simplistic diagram showing the Driver/Leader how it first creates SparkContext , then communicate with Cluster Manger through Spark Application Master and then get the executors launched which in turn execute tasks assigned by the driver.



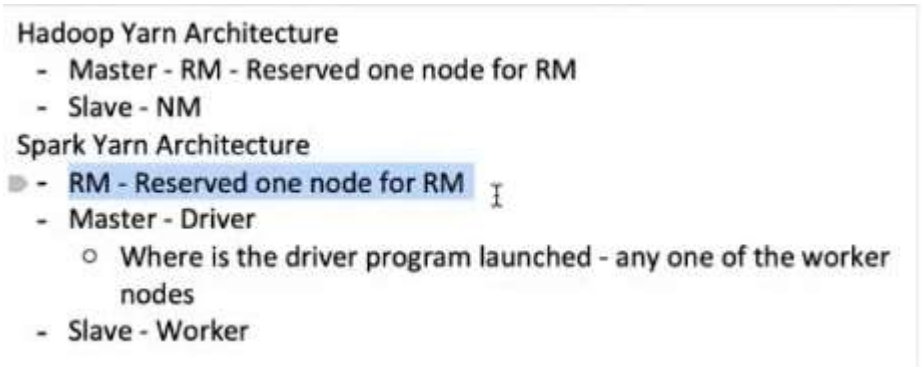
Description of the Diagram :

1. When Action is performed then the Driver Program request for Worker Nodes/Executors through the Cluster Manager.
2. Whatever program/spark application we will write will constitute a driver program .
3. As it can be seen that when action is encountered the DAG is created , tasks division etc. take place (number of tasks depend on the number of available resources) and then code files are being submitted to executors through Yarn(i.e. Code / Driver Program / Jar Files) to the Executors for executing the tasks

Spark Benefits :

In memory Data sharing against Hadoop storing intermediate results using HDFS and then shuffling.(so avoid lot of disk input and output)

Yarn Vs. Spark on Yarn Architecture



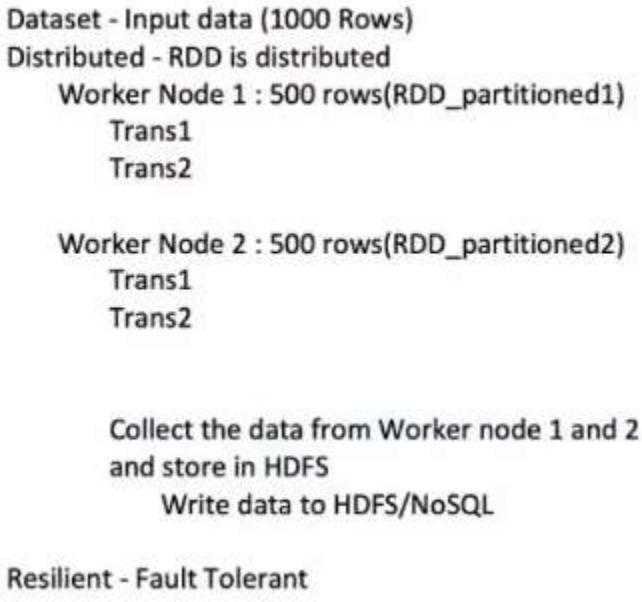
Yarn has Master Resource Managers and individual Nodes which have Nodes Manager.

Spark On Yarn has Spark Driver running on any Yarn Node and then through Yarn Master/RM ask for resources and hence executing tasks on Yarn worker nodes.

Important Spark Concepts :

1. RDDs
2. DAG
3. Lazy Evaluation

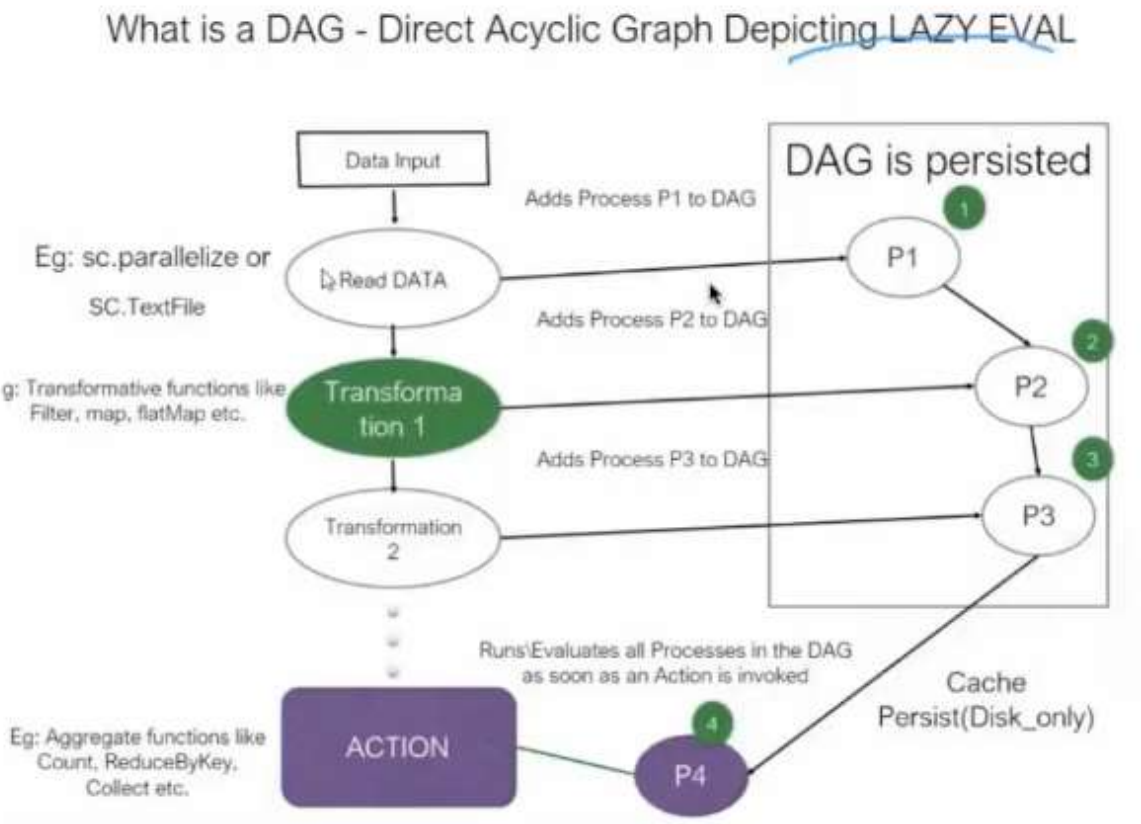
RDDs

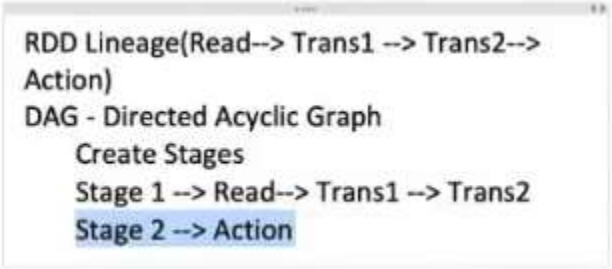


Dataset is what is input by us.

Distributed as when we transform our input data into the RDD then our Data gets distributed over the worker nodes.

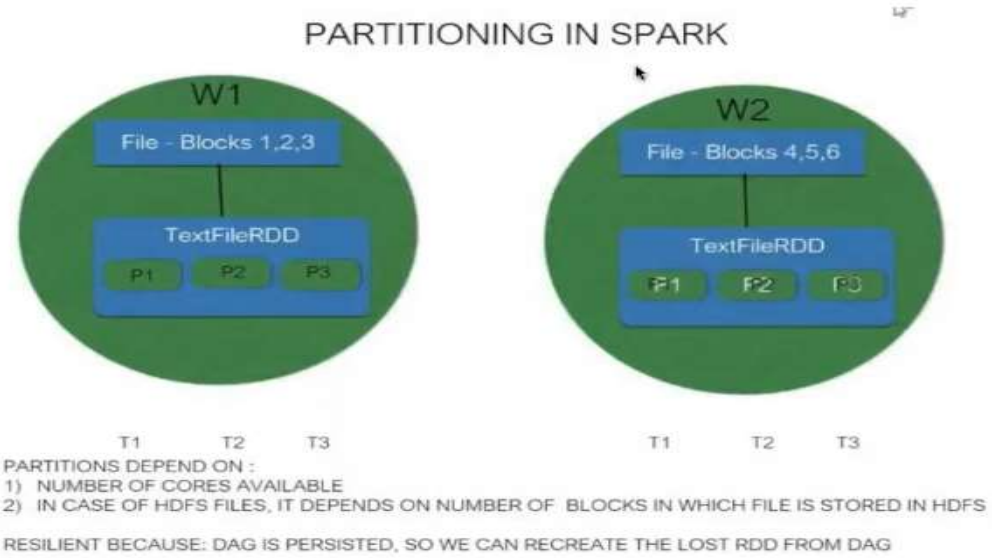
Resilient – due to fault tolerance way of replication of partitions





Important point :

- In Memory Data Sharing happens
- Action will not just lead to data collected at Driver but in any of the worker also like then saving all data collected to HDFS/NoSQL etc.
- Driver is responsible only for storing the DAG and workers only have to do any transformations and actions.
- When you read Data it creates RDD and when you perform Trans1 then RDD and then Transform 2 then RDD so on. So the concept of RDD was introduced in Spark.



Why RDD and how the Spark provides you with Fault tolerance ?

10. Persistence of DAG , which makes possible to move to another node in case of Node Failure and again use DAG to perform the failed tasks and recreate RDDs doing transformation on previous immutable RDDs on another Node.
11. DAG keeps the Lineage of RDDs in case of failure happens it(Spark) will recreate an RDD
12. Replication of Distributed partitions of Datasets.

Note RDDs are immutable. As we know that RDDs are created at every stage of reading , transformation etc. and are important to be immutable

Difference between the ReduceByKey and GroupByKey