Welcome to the Spark Module and i hope you have finished the Pre-Req for this module. If you know Hadoop it would be good.

<https://spark.apache.org/docs/1.6.2/programming-guide.html>

The password if any used are :- hdp or 123

Training Pattern

10 - 11.45

break at 11.45 → 15 minutes

till 1.30 ~ 1.45

lunch break → 45 min

till 3.45

tea break → 15 min

Wrap up by 6.00

Every Day a video will be shared and do watch it before the next day.

<https://tinyurl.com/jp-spark-oct2918>

# Day 1

The Shared Location is :- **\\192.168.1.17\Spark Training**

The basic Software Pre - Req are :-

1. Setup our environment → Hadoop & Spark will be pre-installed. Image is provided to you and it is named as :- Hadoop\_Spark\_2018.rar
2. We need to setup VMWare in our systems and is available at the shared link→ VMware Pro 12 ~ GY
3. Copy the Materials locally and in your own Pen Drive.

The contents of this Training would be

Languages → Python, Scala and one program on Java

Types of Programs: -

1) Shell Programs → REPL - Interactive Mode of python and scala.

2) Batch Mode → Non-Real time environment → eclipse

3) Notebook → Zeppelin, Jupyter

Version of Spark --< 1.6.2

1. Spark Core → Java and Scala will be faster than python
2. Spark SQL → Because of catalyst optimizer all languages will be the same
3. Spark Streaming → Scala and Python or Java as they are the same.
4. Advanced Features of Apache Spark

2nd Module → Advanced Spark in which we will be focusing on Spark ML and GraphX

================================>

Hadoop Modules

1. Setup of Hadoop → 2.7.2
2. What is MR and the basics of it.
3. Apache Hive - What and How we work with it. → SQL
4. Apache Sqoop → Data Movement between HDFS and RDBMS
5. Apache HBase → NoSQL Database

----------------------------------------------------------->

Different forms of Virtualization

1. VMWare
2. Oracle Virtual box
3. KVM

Now we will setup the environment →

1) File - Open - Navigate to the folder [ Hadoop\_Spark\_2018] and open the Ubuntu 64-bit file which is the basic image file. Note: We have to give it ½ of the total RAM and ½ of the processors available.

2) Username → notroot and Password → hadoop123

The basic environment in VMWare is ready.

Now we will setup Putty and Winscp

The settings for the Putty window are

1. Window menu → Change the lines of scroll back to 9999
2. Window - Appearance - Change → font to 18
3. Window - Colors → Click on the checkbox called use System colors
4. Session → Select the session from the drop down list [ This is your IP ] and click on Save

Start WinScp in your system.

To show Hidden files → Options - Preferences - Panels → Show Hidden Files

====================>

Image → Putty → WinScp

Now we will check what is loaded in our image

Go down to Winscp and we will see the folder structure in Winscp

.bashrc → alias or a bash-shell → Basic export variables will be present over there. When we start putty the .bashrc file is automatically called. This file will have the basic environmental variables already loaded.

V1 → hadoop fs → older versions of hadoop

V2 → hdfs dfs → new version of hadoop

downloads → basic tar files for the project

lab

→ data → Sample data Files

→ hdfs dfs -copyFromLocal LocalFS HDFSFS

-put → taking file from any location and putting in to HDFS

-copyFromLocal → taking files from the local location and putting in to HDFS

→ software → Extracted versions of the software on the system.

→ hdfs → Hadoop File System. namenode → metadata and the datanode folder which will hold the actual data.

→ programs → All user defined programs.

We will start our hadoop daemons and see if they are running :-

start-dfs.sh

Namenode

Datanode

SecondaryNameNode

jps → Java Process Status

notroot@ubuntu:~$ jps

1986 Jps

1665 DataNode

1871 SecondaryNameNode

1554 NameNode

The numbers on the LHS are the Process IDs.

start-yarn.sh

ResourceManager

NodeManager

notroot@ubuntu:~$ mr-jobhistory-daemon.sh start historyserver

A new daemon called JobHistoryServer will be started.

So totally we will have 6 deamons running as part of hadoop

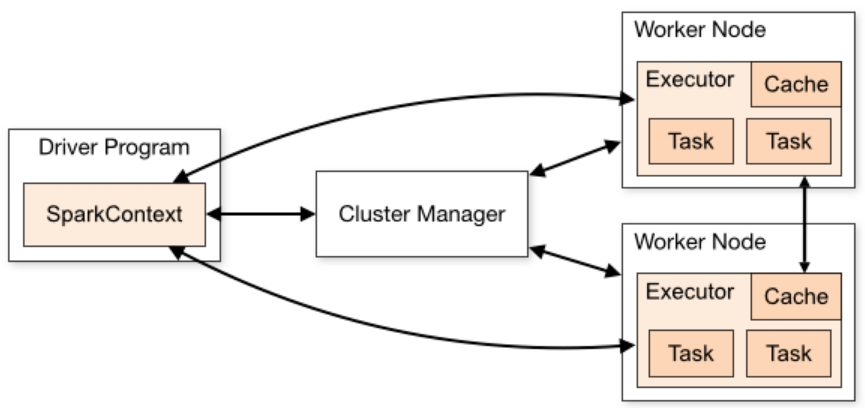
==================> Post Morning Tea

We will start with our spark environment. We will first start with scala prompt as spark was originally developed in Scala.

We will start with REPL mode of working

Let us look at the Spark Architecture :-

https://spark.apache.org/docs/latest/cluster-overview.html



1. Driver Program → This typically on the client side.
2. Spark Context → Handle by which we communicate to the worker nodes
3. SC talks to the Cluster Manager
4. The Cluster Manager allows us to get in touch with the Worker Nodes [ Slaves ] where the processing will happen.
5. The Worker Nodes are the slaves where the actual processing will happen
6. Executor is a component in Spark which has Task and Cache.
7. Cache can be of 2 types
   1. Implicit Cache → Taken care of by the system.
   2. Explicit Cache → This is typically done by the developer and the most important aspect is deallocation of explicit cache.
8. Task → Smallest Unit of Execution,

The are 2 ways in which spark will interact with HDFS

1) HDFS and Spark in the same system.

Spark Cluster

1 2 3 4 5

Need High amount of memory

Everything is performed in memory.

HDFS Cluster [ Where the data is ]

1 2 3 4 5 6 7 8 9 10

2) Hadoop and Spark in distinct Systems.

type → spark-shell in your prompt → What happens when we do this? What gets loaded when we run this.

1) Starts with the HTTP Server → Jetty

2) Display the Spark Version - 1.6.2 and Scala Version → 2.10

3) Starts with the Spark Driver → On the client side.

4) Starts with the Spark UI on 4040.

5) Starts with sc [ Spark Context ] → This is the handle for deployment. Only in a REPL mode it is started by default

6) Hive gets loaded by default → This is going to be used when we start our Spark SQL.

The actual metadata of hive is stored in metastore\_db folder. The default database used by spark is Apache Derby.

Where is the actual location in HDFS? /user/hive/warehouse.

18/10/28 23:48:25 INFO hive.HiveContext: default warehouse location is /user/hive/warehouse

18/10/28 23:48:29 INFO metastore.MetaStoreDirectSql: Using direct SQL, underlying DB is DERBY

Hive → store the actual table structured

HDFS → will contain the actual data.

6) SQLContext is loaded and is readily available to us.

The 4 components that we should see is the port nos of the different components :-

1. NN → 50070
2. SNN → 50090
3. RM → 8088
4. JHS → 19888

Now we will start with the actual coding → Scala →

val -> Immutable

var -> Mutable

1) Please note everything in Spark is by default Immutable.

2) Spark is lazily evaluated.

3) Everything will be in memory

Create a new file in lab/data → sample → We want to perform a Word Count here.

How are you

I am fine

How about you

First Step is that we have to load the file from lab/data →

If the file is locally available → file:///actual path

If the file is in HDFS →

The actual FS is always by default HDFS → /

val fname = sc.textFile("file:///home/notroot/lab/data/sample")

fname is a RDD.

Resilient Distributed Dataset

fname.collect() → Tell me how is this loaded. This is loaded in CPU cores?

res0: Array[String] = Array(How are you, I am fine, How about you)

To check actually how it is loaded in memory →

fname.glom().collect().

res1: Array[Array[String]] = Array(Array(How are you, I am fine), Array(How about you))

When we see Removed broadcast → It is memory eviction in action.

Remember in Spark, we have 2 important components

1) Transformations → Will always add to the DAG → Directed Acyclic Graph.

2) Actions → will actually get executed.

Only the execution component [ actions ] will be shown in the UI page.

We will now look at 2 more of transformations → map and flatMap()

flatMap will perform the MAP operation and then at the last it will give the results as a single Array.

Map do → It will apply some business logic on every element in the RDD.

val fname1 = fname.flatMap( ) → Pass in the function list is a lambda expression. What is lambda expression - it is a anonymous function.

val fname1 = fname.flatMap(sou => sou.split(" ")) So this Anonymous function will apply this function on every element in the collection.

fname1.collect()

res2: Array[String] = Array(How, are, you, I, am, fine, How, about, you)

Now we will create 1 to every element in the collection

val fname2 = fname1.map(k => (k,1))

fname2.collect()

res3: Array[(String, Int)] = Array((How,1), (are,1), (you,1), (I,1), (am,1), (fine,1), (How,1), (about,1), (you,1))

=================> Post Lunch

Input Mapping Shuffling Reducing

Blocks Business Bring together Aggregation

Logic all values for

similar keys

key, LIST (values) key,SUM/COUNT

2 steps that we do in our Map Reduce → Shuffling and Reducing

fname2.glom().collect() → give results at the core level

res1: Array[Array[(String, Int)]] = Array(Array((How,1), (are,1), (you,1), (I,1), (am,1), (fine,1)), Array((How,1), (about,1), (you,1)))

Internally we have to do both shuffling and reducing.

reduceByKey → bring together all the values for the similar keys → shuffling

1st Stage → (How,[1,1,1,1])

2nd Stage → ( business logic of counting)

val count = fname2.reduceByKey((k,a) => (k+a))

This returns a Shuffled RDD.

count.collect()

res2: Array[(String, Int)] = Array((are,1), (fine,1), (am,1), (How,2), (you,2), (about,1), (I,1))

val count1 = fname2.reduceByKey(\_ + \_)

count1.collect()

core1 core2

fname (How are you)(I am fine) (How about you)

fname1 (How,are,you,I,am,fine) (How,about,you)

fname2 (How,1),(are,1).... (How,1)

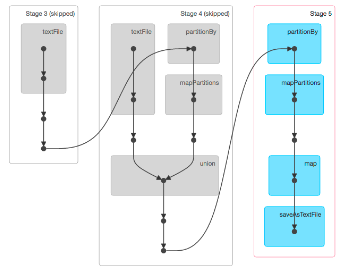
count (How,2)

Tell me what is a stage → Stage is a set of tasks that can be done without any interchange of data between the cores.

count.saveAsTextFile(“file:///home/notroot/lab/data/results”)

We are saving the file in to a resource

Note: The earlier stage was already executed and hence it was not executed again and here we see the benefit of spark where it does not execute every thing from the beginning.



Type in exit to come out of the Scala prompt.

===============> We will see the same example using python

sc.appName → Give the current prompt

sc.master → local[\*] → It will eagarly pick all the available cores.

sc.master → local[1] or local

val fname = sc.XXXXXX

fname.getNumPartitions → Note this is a property and not a method.

Difference between Scala and Python

1. In Python, the sparkSQL variables are loaded in a lazy fashion whereas it is loaded in a eagar fashion in Scala
2. In Python, there is no need for a val or var which is required in Scala
3. For Anonymous functions the word lambda is mandatory in python and not in scala
4. The DAG visualization will not be the same? Why? Because Spark is developed using Scala and we are using Python. so what converts python in to scala byte code. We have a additional compiler called Py4J and because of this the DAG visualization would be different.
5. The performance of python over java will be slower.

=====================================> Post Tea Break

REPL mode → Spark and Python

So for running java codes we would require a specific version of eclipse, which support both java and scala.

In the eclipse for java code, ensure that the perspective is set to java.

Steps for creating the Source Code File

1. Click on the Open Perspective button and change the language to Java
2. File - New - Java Project --< Give the name as Java WC
3. Create a package called com.jpmc
4. Copy the java code given [ WordCount.java in the Day 1 folder ] to this package in eclipse.
5. Add the libraries → Copy assembly,jar from spark / lib folder to any location [ d:\ ] in windows
6. Right Click on the project - Properties - Java Build Path - Libraries Tab - Add External Jars and select the assembly,jar which you created.
7. Change the package name to com.jpmc
8. Build a jar file out of the source code → Right click on JavaWC and click export → Java - Jar → click on next → In the jar file location → d:\JavaWC.jar
9. Move JavaWC.jar to programs directory

notroot@ubuntu:~$ spark-submit --class com.jpmc.WordCount lab/programs/JavaWC.jar file:///home/notroot/lab/data/sample /JavaWCResults

You can run this code from /home/notroot itself.

Check the final output in HDFS and we will see only 1 part file as we have said setMaster(“local”) in the main function.

In Java, How we will see the DAG? We cannot see it in the browser as the Java Prompt is closed.

Start with the History Server for Spark →

A] Create a directory in HDFS for logging, say /sparkevents

hdfs dfs -mkdir /sparkevents

B] Write down these lines in spark-defaults.conf in the conf directory of spark after moving the template word from this file.

spark.history.fs.logDirectory hdfs://localhost:9000/sparkevents

spark.eventLog.dir hdfs://localhost:9000/sparkevents

spark.eventLog.enabled true

C] from the spark/sbin dir --> ./start-history-server.sh

D] Check with JPS is the HistoryServer daemon is running

E] Execute the spark-submit command again.

F] Then check with the IPADDRESS:18080 port no.

The order of precendence is

1. Code
2. Runtime while executing it via spark-submit
3. config

==========> Check how a WordCount example is done in Scala.

Note: Remember to create the input directory and put the same file inside that.

notroot@ubuntu:~/lab/data$ hdfs dfs -mkdir /input

notroot@ubuntu:~/lab/data$ hdfs dfs -copyFromLocal sample /input

This is the way to run the jar file :- notroot@ubuntu:~/lab/programs$ spark-submit --class com.jpmc.ScalaFirst ScalaWC.jar

Check for the following line in the output → Number of Lines in the Dataset 3

Now we will have to remove the INFO messages and change the Log Level.

Change the Logging level →

a) Remove the template word from log4j.properties file

b) Line No 19 → Change the INFO to ERROR

Rerun the earlier example and we will not see any output other than ERROR logs.

We need to set up Jupyter in our environment and all the steps are in Jupyter\_Install.txt document.

# Day 2

Stuff done on Day 1

1. Started with Hadoop 2.7.2 and Spark 1.6.2 in our image
2. What is Apache Spark - Attributes
3. Architecture of Spark
4. What happens when we start REPL - spark-shell
5. Analysis of Word Count using Scala - REPL
6. Analysis of Word Count using Python - REPL
7. Analysis of Word Count using Java - spark-submit
8. Word Count using eclipse in Scala - spark-submit
9. Job History Server of Spark
10. Remove the additional lines in Spark
11. Jupyter Notebook

<https://neo4j.com/blog/analyzing-panama-papers-neo4j/>

<https://neo4j.com/blog/analyzing-paradise-papers-neo4j/>

redhat → yum or rpm

ubuntu → apt-get

Plan for Day 2

1. Various Transformation & Actions in jupyter notebook - 15 and 1 action
2. Small POC →
3. Spark SQL

==============> Setup our Anaconda and it is ready.

Founder of Spark → Matie Zaharia

The company behind Spark is Databricks →

The 2 types of RDDs are

1. Scala RDDs
2. Paired RDDs

fname → sc.textFile Narrow

fname1 → flatmap Narrow

fname2 → map Narrow → always a part of 1 stage

count → reduceByKey Wide → shuffle involved

Lineage → ability of a RDD to create the earlier stage.

1) map → apply the business logic on each element in the source collection.

2) filter → applies a business logic on every element in the collection and if it returns a value, it considers it and if it does not return a value, it skips it.

3) flatMap( some value) → It applies map and then brings the results in to a single collection.

===================> Post Morning Break

4) groupBy → This will expect a parameter in the function and grouping is done on all elements in the source RDD based on this parameter. This happens with Scalar RDDs and we can customize it.

5) groupByKey → groups the similar keys. Will happen only Paired RDDs.

1. get a key, LIST[values]

Core 1

(ABC,1) (ABC,1) (ABC,1)

Mapping

How many elements will have to go through the network - 3

Is it going to be merged as one- NO

Shuffling → ABC,[1,1,1]

Aggregation → ABC,[3] → Note there would no Aggregation in groupByKey.

Combiner → It perform the aggregation at the mapper level itself so that individual similar elements can go in one go.

Core 1 Core 2

(ABC,1) (ABC,1) (ABC,1) (ABC,1) (ABC,1)

(ABC,[1,1,1]) (ABC,[1,1])

Combing → Will this happen at both the cores. YES. Combiner always works at the core level.

Mapping How many elements will have to go through the network - based on the number of cores that we have.

Shuffling phase → Bring together all similar values together

(ABC,[1,1,1,1,1])

Reducing Phasee → (ABC,5)

6) reduceByKey → group the similar keys. It does 2 things

a) get a key,LIST[values]

b) perform the aggregation on the LIST

7) mapPartitons → Performs the aggregation at the partition level and not at the element level like map function.

<https://www.datacamp.com/community/tutorials/python-iterator-tutorial>

8) sample

The first parameter is whether the same element can be taken or not

The second parameter is the probability of whether the same element will be present in the answer or not

The third parameter is the randomized number so that we will get the same result with the same number

9) union → simply bring together all the partitions together. Narrow. Duplicates are not removed.

10) join → Note that this will return a paired RDD for every value that is in the parent RDD for every value that is in the Child RDD.

11) distinct → Note this is going to be a wide transformations.

12) coalesce → reduce the number of partitions. Note: To increase the number of partitions, we will have to call a transformation called repartition. Note that this is a optimization operator.

Where to use coalesce. → If we see skewed data then the best way to re-arrange that is via coalesce.

=====================> Post Afternoon Lunch

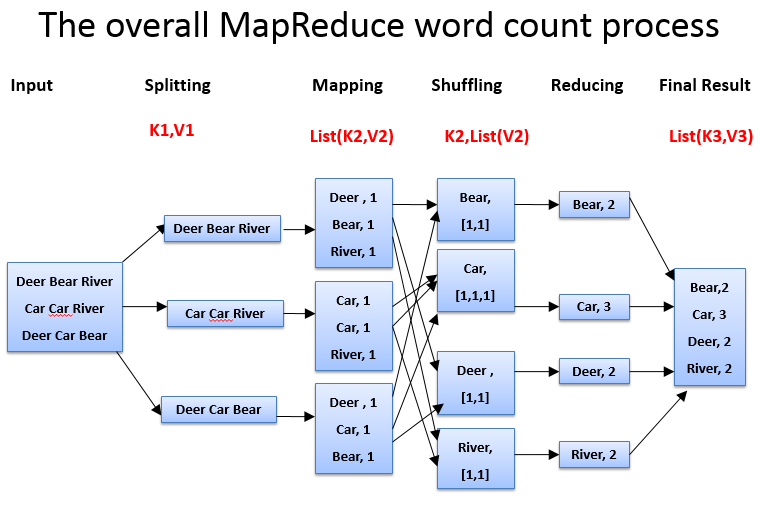
13) keyBy → Primary purpose of keyBy is based on a function or a business logic convert the scalar RDD in to a pairRDD.

14) partitionBy → To customize the partitions in to a specific list of partitions.

15) zip → This will take partition wise elements from the first RDD and pair it with the Second RDD. Note: The number of elements in both RDDs should be the same.

16) reduce action → Typically is the final step in our computation where we get a answer and we cannot call collect method on this.

Traditional Map Reduce Word Count.



============================> Small POC

New thing here is that we are going to use our Hadoop environment [ YARN process ] in this example.

What happens when we execute something in YARN mode →

Resource Manager

Node Manager

1 2 3 4 5 6 7 8 9 10

128 72

AM

1. submit the command for execution [ client system ].It will go to the Resource Manager.
2. In the YARN Mode, the RM will talk to any available container and will start the ApplicationMaster in that container. This can be started on any Node in the Cluster.
3. The AM will talk to the NN to get the IPs of the Nodes where the data is and then the processing will start.

Tell me what is an Application Master → Is always the first container which is started and this is responsible for managing the life cycle of the applications.

Exercise: Work with the small POC [1st part to check the top 3 airlines with most projects ]

carrierRDD → (carrierInfo, COUNT)

Swap the values

name = name.map( lambda (carrierInfo,COUNT) : (COUNT,carrierInfo))

Call the SortByKey

Started with Small Project :-

a) Find out top 3 Airlines with Most Flights:-

flightRdd=sc.textFile("/input/flights.csv").map(lambda line: line.split(","))

carrierRdd = flightRdd.map(lambda line: (line[5],1))

ReducedRdd = carrierRdd.reduceByKey(lambda a,b: a+b)

carriersSorted = ReducedRdd.map(lambda (a,b): (b,a)).

FinalAnswer = carriersSorted.sortByKey(ascending=False).take(3)

FinalAnswer

Way of swapping in jupyter notebook:- map(lambda a: (a[1], a[0]))

Result:- [(87, u'NW'), (85, u'WN'), (62, u'AA')]

Scala Code:-

val flightRDD = sc.textFile("/poc/flights.csv",2).map(\_.split(",")).keyBy(\_(5)).mapValues(flightName => 1)

val cntFlight = flightRDD.reduceByKey(\_ + \_)

val revResults = cntFlight.map(f => (f.\_2, f.\_1))

val sortedResults = revResults.sortByKey(false)

sortedResults.take(3)

=================> Post Evening Tea →

We are finished with Spark Core and we will now move on to Spark SQL

Spark SQL → Way to interact with data which is in the SQL format.

→ sqlContext.sql (SQL Query)

→ DataFrame

Connect to various Data sources

HDFS

Local FS

NoSQL → HBase and Cassandra

S3

The name DataFrame came from

1. R → DataFrame
2. Python → Pandas

Note everything in SQL will be in the form of ROW object.

File Format supported for SQL Style Access :-

textFile

JSON → column headers are a part of the actual data

avro

RC

ORC

Parquet →

Note: sqlContext is readily available only in the REPL mode.

sqlContext → create table

HDFS → actual table created → /user/hive/warehouse/customer

Row Format Delimited → means record separator which by default is ,

If we need to specify a different record separator --> lines terminated by ‘|’

External v/s Internal Table → Internal Table.

Fields terminated by ‘,’ → Column Separator

Stored as textFile → This is the file Format. [ we have given the various types of file formats earlier ]

===============>

sqlContext HDFS

create table a new table gets created.

Storing the data → Local means linux and for HDFS we should remove the word local.

sqlContext.sql("CREATE TABLE IF NOT EXISTS customer (id INT, name STRING, city STRING, state STRING, zipcode STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored as textfile")

Then copy the customer.txt in to the lab/data location

sqlContext.sql("LOAD DATA LOCAL INPATH '/home/notroot/lab/data/customer.txt' INTO TABLE customer")

val result = sqlContext.sql("FROM customer SELECT id, name, city")

val result1 = sqlContext.table("customer").select ("id","name").head(2)

val dfs = sqlContext.read.json(“/input/employee.json")

Note: Before running the baby\_name we should execute this in the .bashrc

export JAVA\_TOOL\_OPTIONS="-Dhttps.protocols=TLSv1.2"

==================>

# Day 3

Stuff done on Day 1

1. Started with Hadoop 2.7.2 and Spark 1.6.2 in our image
2. What is Apache Spark - Attributes
3. Architecture of Spark
4. What happens when we start REPL - spark-shell
5. Analysis of Word Count using Scala - REPL
6. Analysis of Word Count using Python - REPL
7. Analysis of Word Count using Java - spark-submit
8. Word Count using eclipse in Scala - spark-submit
9. Job History Server of Spark
10. Remove the additional lines in Spark
11. Jupyter Notebook

Stuff done on Day 2

1. Started with Jupyter notebook and saw the Transformations and Actions -- map, flatMap, filter, groupBy, groupByKey, reduceByKey, mapPartitions, sample, union, join, distinct, coalesce, keyBy, reduceBy, zip and also the reduce action.
2. Small POC → Top 3 airlines having the most flights with --master yarn
3. Spark SQL → Traditional hive style → creating a table, loading data in to the table, retrieve the data
4. Example using JSON and we also changed the shuffle partitions to 4
5. Example of csv using a specific library from spark-packages.org

Day 3 → Stuff

1. Word Count example using pyspark in a batch mode
2. Continue with our Spark SQL → working with PySpark
3. Working with DF
4. Spark Streaming
5. Parquet data format
6. MySQL - take data and load it in a dataframe
7. Accumulator and Broadcast variables
8. Zeppelin Install

==============================================>

file → WordCount.py

spark-submit WordCount.py

SparkConf → This is the first class which we will have to load

SparkContext → pass the SparkConf as a parameter.

Continue with our SparkConf →

What is the RDDs and DataFrames [ Only for SQL Style Processing ] ?

In RDDs we will have to compute manually using the language which we prefer and in DataFrames we will use SQLContext.sql or DF API to work with our analysis.

If you are having unstructured data we will be using SparkStreaming.

Does Spark support Real Time Processing → Near Real time Processing

If you want real time processing → Apache Storm with Kafka

Madhukar Pathak → Data Mantra → Catalyst

<https://www.youtube.com/watch?v=TCWOJ6EJprY>

<https://www.youtube.com/watch?v=bqs1HKlvXzQ>

<https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html>

Exercise:- Working with the WordCount in Spark SQL

<https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html>

Note: From Spark 2.0 onwards, In Java we will not have DataFrame but a DataSet API.

================> Post Morning Tea

Exercise: Create a DataFrame from a RDD. → Using Scala

POJO → Plain Old Java Object

DataSet Basic Concepts

https://data-flair.training/blogs/apache-spark-dataset-tutorial/

--------------->

<https://databricks.com/session/optimizing-apache-spark-sql-joins> → Good Video Tutorial on various types of Spark SQL Joins.

Cost Based Optimizer.

<https://databricks.com/blog/2017/08/31/cost-based-optimizer-in-apache-spark-2-2.html>

Creating a DataFrame from a RDD → Scala

Intel Virtualization → Win 8 → <https://www.youtube.com/watch?v=3irpIFya_lk>

What came first in Spark / Hadoop → Create table or the Data came first → Data

So we cannot do Validations in Spark / Hadoop.

---------------------------------->

Exercise → PySpark → Checking the customer table created earlier

Exercise → PySpark → toDF and createDataFrame().

Exercise → Working with the DF concepts

→ Working with UDFs.

Now we will close our current shell and start with our jupyter notebook in which we will see the various functions of DF.

To save the DF to a Hive Table.

1. df.saveAsTextFile(“/input/result”)
2. In Hive create a table and point to the result directory.

----------------------------------> Post Afternoon Lunch

To Persist a RDD we can call → df1.rdd.saveAsTextFile("/input/Test")

Also remember to call coalesce to 1 before that so that we can get only 1 output file.

We will talk about Spark Streaming → Non-Real Time. If we need real time → Apache Storm + Kafka.

Since it is non-real time, we have to specify the duration when it would trigger.

In Spark Streaming we will have to save the results of a individual process or it is lost.

The duration in spark streaming is also very crucial.

Duration of Spark Streaming → 1 min

Time taken for Spark Streaming to finish its work → 2 min

We will end up in a eventual data loss here,

The name of spark streams is DStreams.

What are the 2 components we have in streaming?

1) Data Source → Typically coming in from a putty or a data source

2) Spark Streams → Consumer of that data

You will always have 2 separate windows for this purpose.

import java.awt.\*; Import only the AWT classes and not event.

import java.awt.event.\*; Here you will have separate packages.

Example → Python.

Most Imp: When we create a SparkContext, we should do it with 2 cores. Why? One will be the receiver and the second will be the processor. The way to start your REPL is with --master local[2]

First Example: Using Python

from pyspark.streaming import StreamingContext

ssc = StreamingContext(sc, 20)

inputDS = ssc.socketTextStream("localhost",9999)

wc = inputDS.flatMap(lambda line: line.split(" ")).map(lambda word: (word,1)).reduceByKey(lambda a,b: a+b)

wc.pprint()

Start with the producer and then call ssc.start() in the consumer

=============> 2nd Example of Spark Streaming - Scala

Streaming Application → Scala

Java Application → Java

Note: Create 2 separate projects in eclipse.

The steps are

1. create a new scala project and name it as FirstStreaming
2. Change the scala container to 2.10
3. Create a new package called com.jpmc
4. Drag and drop the FirstStreaming.scala in to this package.
5. Add the dependency files → spark assembly jar file
6. Change the package to com.jpmc

For the Java Project

1. create a new java project and name it as ClientApp
2. Create a package called com.jpmc
3. Drag and drop ClientApp.java inside the package

Now we have 2 separate project and we will create a jar for each one of them. But please note you will have to change the language before you create the jar file.

Java Project → Right Click on the project - Export - Java File → Give the name of jar file as ClientApp.jar

Now change the perspective from java to scala and you can create a jar file for FirstStreaming.jar

Move both the jar files to lab/programs directory and they execute the jar files. Ensure that you are in the lab/programs directory.

java -classpath ClientApp.jar com.jpmc.ClientApp

This will start the producer

spark-submit --class com.jpmc.FirstStreaming FirstStreaming.jar

This will start the consumer

val output = fs.create(new Path("hdfs://quickstart.cloudera:8020/user/cloudera/sample.txt"))  
val writer = new PrintWriter(output)

writer.close();

==========================> Post Evening Tea

The file format used internally by JP is parquet format which also compresses the file.

Exercise on Parquet file → ParquetExample.txt

We are seeing 2 examples for Parquet

1. We are using cars.json here
2. We are using txns file and seeing the difference between normal consumption and using via parquet format.

================>

MySQL → MySQL.txt → Follow the Steps

Accumulator - Broadcast Variables → https://spark.apache.org/docs/1.6.2/programming-guide.html#shared-variables

<https://www.cloudera.com/developers/get-started-with-hadoop-tutorial/exercise-1.html>

The data set used in the example is already present retail.db in your Day3 folder.

Associative → (a+b) +c = a+(b+c)

Commutative ⇒ (a+b) = (b+a)

Zeppelin → look at zeppling\_install.txt in the Day3 folder

learning.jpmorganchase.com → to do section → today’s training session and click and give the feedback.

To stop the services do the following :-

1)zeppelin-daemon.sh stop

2)stop-all.sh

3) notroot@ubuntu:~/lab/software/spark-1.6.2-bin-hadoop2.6/sbin$ ./stop-history-server.sh

4) notroot@ubuntu:~$ mr-jobhistory-daemon.sh stop historyserver

5) Check via jps that no daemons are running

6) stop the putty and winscp session

7) Go the VMWare Image → VM - Power - Power Off

Zip the image and take it with you → Note that you should not have a .lck folder in your location.

Name: Venkita Krishnan → Contact Nos: 9821422745