*TE (Comp)*

**

**Experiment:B2**

**Title :**

Write a simple program in SCALA using Apache Spark framework

**Aim :**

* **install spark**
* **Deploy your own Spark cluster in standalone mode.**
* **Running your first spark program : Spark word count application.**

**Prerequisites:**

Java , Apache Spark

**Objectives :**

To learn the concept of how to display summary statistics for each feature Available in the dataset

**Theory:**

Standalone Mode in Apache Spark

Spark is deployed on the top of Hadoop Distributed File System (HDFS). For computations, Spark and MapReduce run in parallel for the Spark jobs submitted to the cluster.

Hadoop YARN/ Mesos

Apache Spark runs on Mesos or YARN (Yet another Resource Navigator, one of the key features in the second-generation Hadoop) without any root-access or pre-installation. It integrates Spark on top Hadoop stack that is already present on the system.

SIMR (Spark in Map Reduce)

This is an add-on to the standalone deployment where Spark jobs can be launched by the user and they can use the spark shell without any administrative access.

Getting Started with Apache Spark Standalone Mode of Deployment

Step 1: Verify if Java is installed

Java is a pre-requisite software for running Spark Applications. Use the following command to verify if Java is installed -

$java –version

Verify if Spark is installed

As Apache Spark is used through Scala programming language, Scala should be installed to proceed with installing spark cluster in Standalone mode. Use the following command to check if Scala is installed -

$scala –version

n case Scala is already installed on your system, it will display the version details. In the above screenshot, Scala programming language is not installed on my system. Let’s install Scala first -

$sudo apt-get install scala

Step 3: Download and Install Apache Spark:

Download the latest version of Apache Spark (Pre-built according to your Hadoop version) from this link: Apache Spark Download Link

Check the presence of .tar.gz file in the downloads folder. To install spark, extract the tar file using the following command:

(In this spark tutorial, we are using spark-1.3.1-bin-hadoop2.6 version)

$ tar xvf spark-1.6.1-bin-hadoop2.6.tgz

Move the spark downloaded files from the downloads folder to your local system where you plan to run your spark applications. Use the commands:

$ sudo su –

Password:

# cd /home/user/Downloads/

# mv spark-1.6.1-bin-hadoop2.6 /usr/local/spark

# exit

Spark Configuration

Let’s setup the environment variable for Apache Spark -

$ source ~/.bashrc

export PATH = $PATH: /usr/local/spark/bin

We add the above line ~/.bashrc file and save it. Setting the PATH variable will locate the Spark executables in the location /usr/local/spark/bin.

Step 3: Verify Apache Spark installation

Verify the installation using the following command:

$spark-shell

In case the installation happened successfully, the above command will start Apache Spark in Scala.

In other words, for this, we just have to place the compiled version of Apache Spark applications on each node of the Spark cluster, after Java and Scala are installed.

Below is the source code for the Word Count program in Apache Spark –

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.\_

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.\_

object SparkWordCount {

   def main(args: Array[String]) {

    val sc = new SparkContext( "local", "Word Count", "/usr/local/spark", Nil, Map(), Map())

      val input = sc.textFile("input.txt")

      Val count = input.flatMap(line ⇒ line.split(" "))

      .map(word ⇒ (word, 1))

      .reduceByKey(\_ + \_)

      count.saveAsTextFile("outfile")

      System.out.println("OK");

   }

}

Linking with Apache Spark

The first step is to explicitly import the required spark classes into your Spark program which is done by adding the following lines -

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.\_

Creating a Spark Context Object

The next step is to create a Spark context object with the desired spark configuration that tells Apache Spark on how to access a cluster. The below line of code in the word count example does this -

val sc = new SparkContext( "local", "Word Count", "/usr/local/spark", Nil, Map(), Map())

“Word Count” – This is the name of the application that you want to run.

“local”- This parameter denotes the master URL to connect the spark application to.

/usr/local/spark- This parameter denotes the home directory of Apache Spark.

Map() – The first map specifies the environment whilst the second one specifies the variables to work nodes.\

Creating a Spark RDD

The next step in the Spark Word count example creates an input Spark RDD that reads the text file input.txt using the Spark Context created in the previous step-

val input = sc.textFile("input.txt")

Spark RDD Transformations in Wordcount Example

The below lines of spark application code transform the input RDD to count RDD -

Val count = input.flatMap (line ⇒ line. Split (" "))

.map (word ⇒ (word, 1))

.reduceByKey (\_ + \_)

In the above piece of code, flatMap () is used to tokenize the lines from input text file into words.

Map () method counts the frequency of each word.

reduceByKey () method counts the repetitions of word in the text file.

Running your First Spark Application

We will submit the word count example in Apache Spark using the Spark shell instead of running the word count program as a whole -

Let’s start Spark shell

$ Spark-shell

Let’s create a Spark RDD using the input file that we want to run our first Spark program on. You should specify the absolute path of the input file-

scala> val inputfile = sc.textFile ("input.txt")

On executing the above command, the following output is observed -

Now is the step to count the number of words -

Each line is split into words using flatMap RDD transformation. flatMap works applying a function that returns a sequence for each element in the list, and flattens the results into the original list.

Each word is read and key-value pairs are created for each one of them using map transformation. This will assign the value ‘1’ to each of the work-keys.

Finally the values of similar keys are added to get the final word count using reduceByKey function.

scala> val counts = inputfile. flatMap (line => line. Split (" ")).map (word => (word, 1)).reduceByKey (\_+\_)

The next step is to store the output in a text file and exit the spark shell.

scala>counts.saveAsTextFile ("output")

**Conclusion:**

Thus we have studied a simple program in SCALA using Apache Spark framework

**Questions:**

* What are some main features of Scala? ...
* Write some benefits of using Scala. ...
* Name some of the frameworks that Scala supports. ...
* What are case classes in Scala? ...
* Explain the term stream in Scala. ...
* What are tuples and what is their usage in Scala? ...
* What are different types of Scala variables?