# Experiment - 14.

- Problem Statement - Write a simple program in scala using apache spark framework.
- Pre-requisite - Basics of java.
- Objective: Students must be able to write a simple program using scala language.
- THEORY -

1] Apache Spark -

a) Modes of deployment -

(1) Standalone mode in Apache Spark.

- Spark is deployed on the top of Hadoop distributed file system (HDFS). For computation, spark and mapreduce run in parallel for the spark jobs submitted to cluster.

(2). Hadoop YARN / Modes -

- Apache spark runs on Mesos or YARN (Yet another resource navigator one of key features of second generation Hadoop.) without any root access or pre-installation.

It integrates spark on top Hadoop stack that is already present on system.

(3). SIMR (Spark in Map reduce). -

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

2] Getting started with apache Stadalone mode of deployment -

Step 1: Verify if java is installed.

Java is pre-requisite software for running spark applications. Use the following command to verify if java is installed.

$ java --version.

If java is not installed, install it first.

Step 2: Verify if Scala is installed.

As apache spark is used through Scala language Scala should be installed. to proceed with installing spark cluster in Standalone mode.

If not installed, install Scala.

- To install Scala -

```
$ sudo apt-get install scala.
```

- Verify installation.

```
$ scala -version.
```

Step 3: Download and install Apache spark.

Download the latest version of Apache Spark (Pre-built according to your hadoop version) from link - spark.apache.org/downloads.html.

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

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

Move spark downloaded files from downloads folder to your local system where you can run spark applications. Use -

```
$ sudo su -
Password:
# cd /home/ubuntu/Downloads/
# mv spark-1.6.1-bin-hadoop2.6 /usr/local/spark
# exit.
```

Setup environment variable –

$ source ~/.bashrc.
export PATH = $ PATH : /usr/local/spark/bin.

Setting path variables will locate the spark executables.

Step 4 : Verify spark installation –

$ spark-shell.

- In case of successful installation, it will install apache spark easily. and open in Scala.
- In other words for this, we just have to place the compiled version of apache spark applications on each node of spark cluster after java and Scala are installed.

* Operating or deploying a spark cluster manually :-

- The most common way to launch spark applications on the cluster is to use the shell command spark-submit.
- When using the spark submit shell command the spark application need not be configured particularly for each cluster as the spark-submit shell script uses the cluster managers through a single interface

- Spark submit script has several flags that help control the resources used by your Apache spark application.

- Spark submit flags dynamically supply configurations to spark context object.

- Program Information -

1. The first step is to explicitly import the required spark classes into your spark program -

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark._
```

2. Creating a spark context -

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.

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

- Word Count = This is name of application.
- local = This parameter denotes the master URL to connect 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 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 Spark Context created in previous step.

```
val input = sc.textfile ("input.txt");
```

- RDD - Transformation -

```
Val count = input.flatMap (line => line.split(" ")).
map (word => (word, 1)).reduceByKey (_+_).
```

- Run the spark application -

```
scala> val inputfile = sc.textfile ("input.txt").
scala> counts.saveAsTextfile ("output").
```

* The output will be uploaded - in the output file.

- **Conclusion:** This we successfully completed program using Scala and apache spark.

* * * * * * *