BIG DATA ANALYTICS

LAB ASSIGNMENT 5

Mahesh Pachare

FINAL YEAR B.TECH IT
191080054

Aim: To set up Spark Multinode cluster and write wordcount program in scala and execute on a spark cluster.

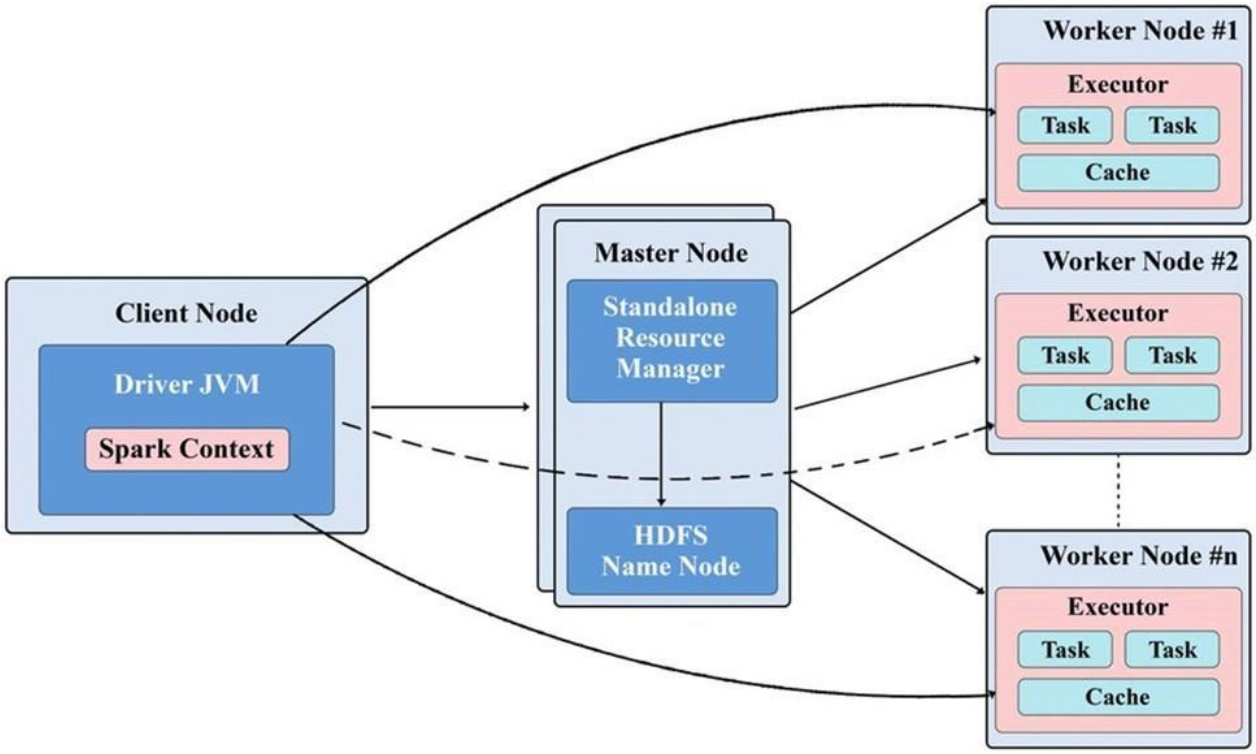
Theory:

Spark is a distributed computing framework that allows the processing of large datasets across a cluster of computers. Multi-Node Cluster refers to a cluster of computers connected through a network that enables parallel processing of large datasets. Scala is a programming language used to develop applications on the Spark platform. It is a statically typed language that combines object-oriented and functional programming concepts.

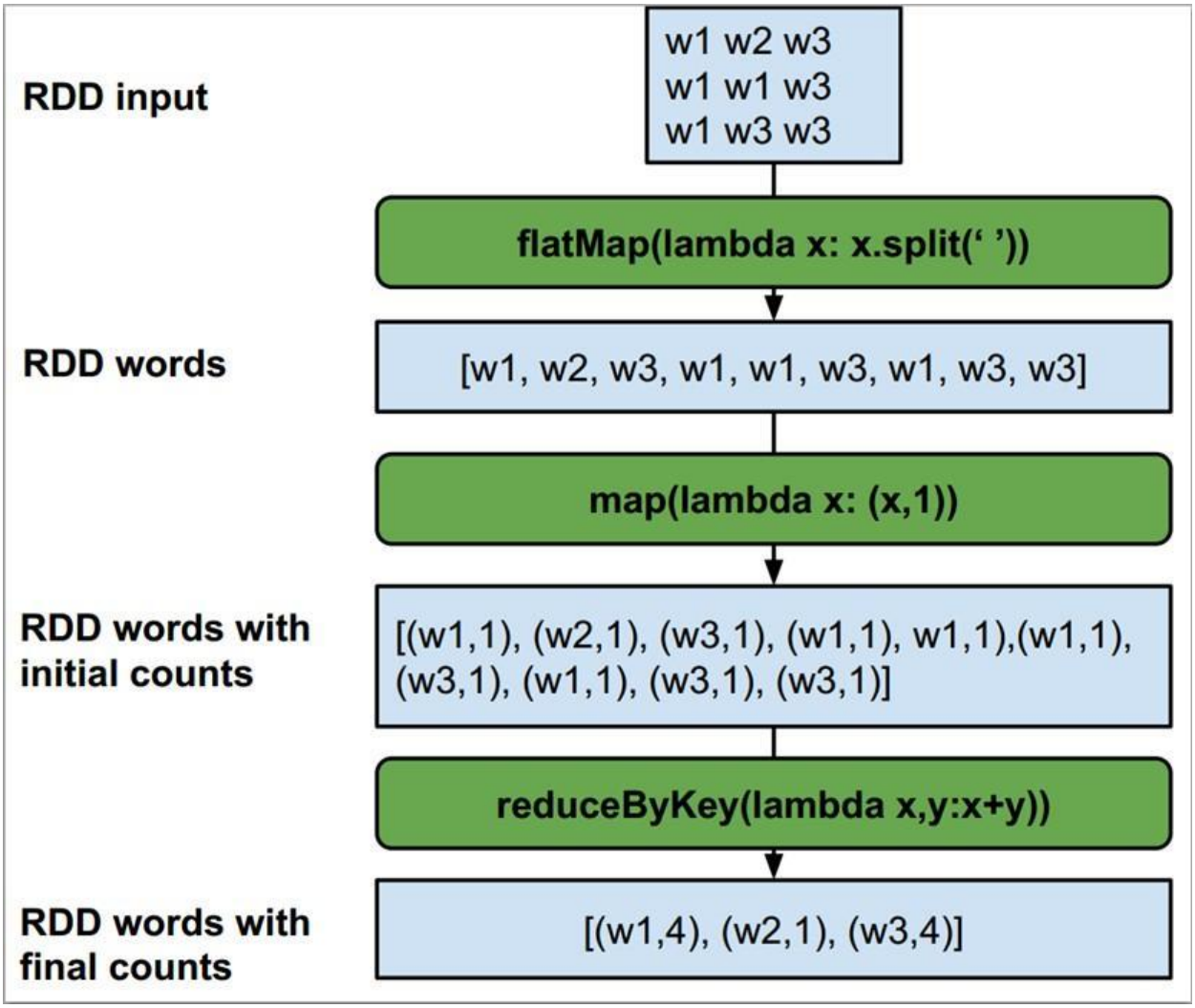
To set up a Spark Multi-Node Cluster, one needs to have a set of machines connected to a network and have Spark installed on each machine. The nodes in the cluster can communicate with each other using a messaging system called Apache Zookeeper. As you can see from the diagram, a Spark cluster consists of several nodes. Each node can be either a worker node or a master node. The worker nodes are responsible for processing data, while the master node is responsible for managing the cluster and coordinating the work among the worker nodes.

When a job is submitted to the Spark cluster, the master node divides the workload into smaller tasks and assigns them to the worker nodes. The worker nodes then process the data in parallel, and the results are returned to the master node for aggregation.

Next, let's take a look at a diagram of a Multi-Node Cluster:



A Multi Node Cluster consists of several nodes connected through a network. Each node can be a physical machine or a virtual machine. In a Multi-Node Cluster, the nodes communicate with each other using a messaging system called Apache Zookeeper. Zookeeper ensures that the nodes are aware of each other's status and can coordinate their actions.



The Wordcount program is a popular example used to demonstrate the capabilities of Spark. The program reads text data from a file and counts the occurrence of each word in the file. The program is written in Scala, and it utilizes the Spark API to distribute the workload across the nodes in the cluster.

To execute the Wordcount program on the Spark cluster, the program must first be compiled and packaged into a JAR file. The JAR file is then submitted to the Spark cluster using the spark-submit command. The Spark driver program then

distributes the workload across the nodes in the cluster, and the results are returned to the driver program for aggregation.

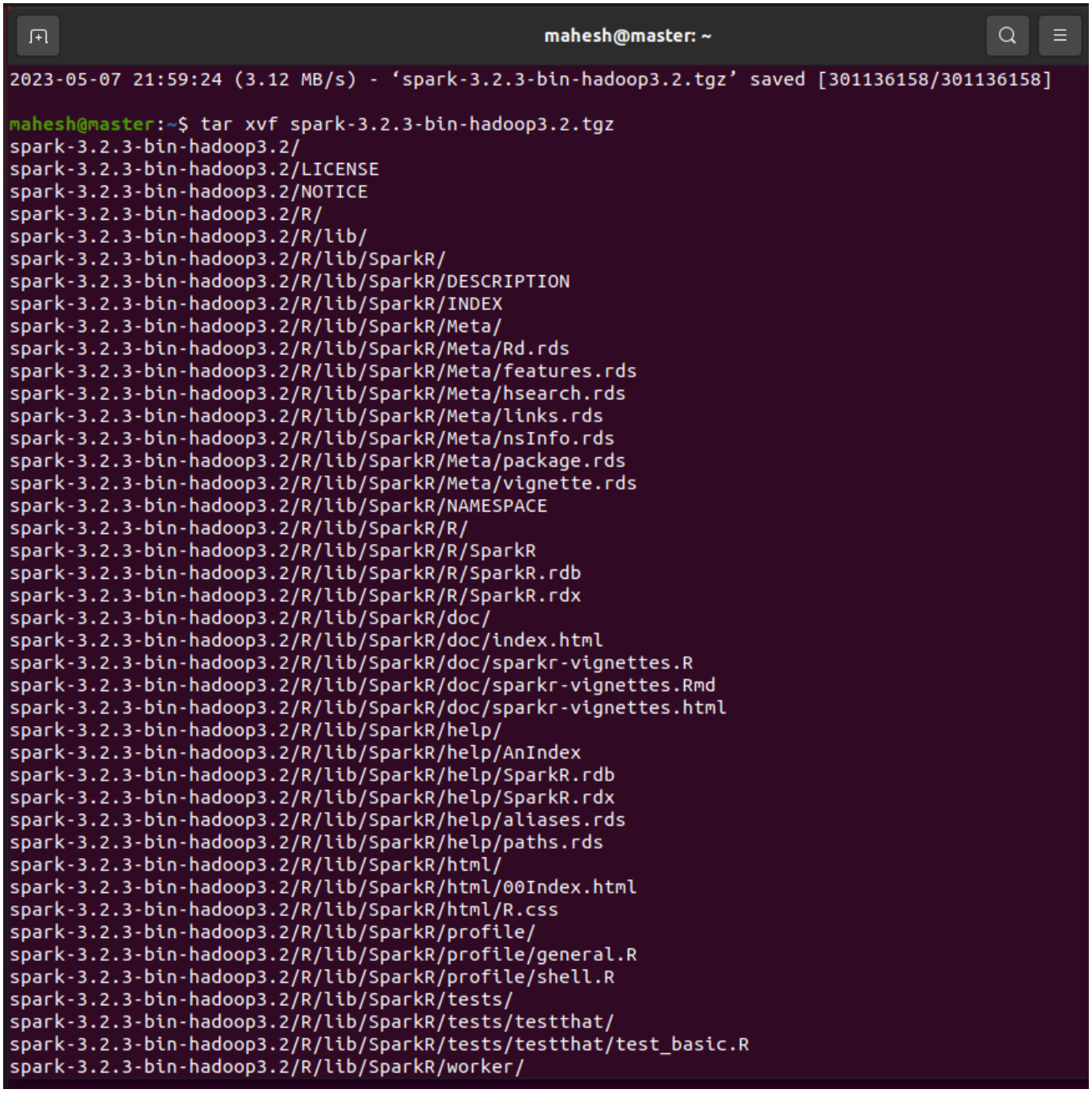
Overall, the expected theory for this experiment involves understanding the concepts of Spark, Multi Node Cluster, and Scala programming language and using these concepts to set up a Spark cluster and execute the Wordcount program.

Implementation:

1) Download the compatible version of Sparks with Hadoop 3.2.x.

```
mahesh@master:~$ wget https://archive.apache.org/dist/spark/spark-3.2.3/spark-3.2.3-bin-hadoop3.2.tgz
--2023-05-07 21:57:50--  https://archive.apache.org/dist/spark/spark-3.2.3/spark-3.2.3-bin-hadoop3.2.tgz
Resolving archive.apache.org (archive.apache.org)... 138.201.131.134, 2a01:4f8:172:2ec5::2
Connecting to archive.apache.org (archive.apache.org)|138.201.131.134|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 301136158 (287M) [application/x-gzip]
Saving to: ‘spark-3.2.3-bin-hadoop3.2.tgz’

spark-3.2.3-bin-hadoop3.2.t 100%[=====================================>] 287.19M  3.12MB/s    in 92s

2023-05-07 21:59:24 (3.12 MB/s) - ‘spark-3.2.3-bin-hadoop3.2.tgz’ saved [301136158/301136158]

mahesh@master:~$
```

2) Unzipping the tar file to get spark files in spark-3.2.3-bin-hadoop3.2 folder.

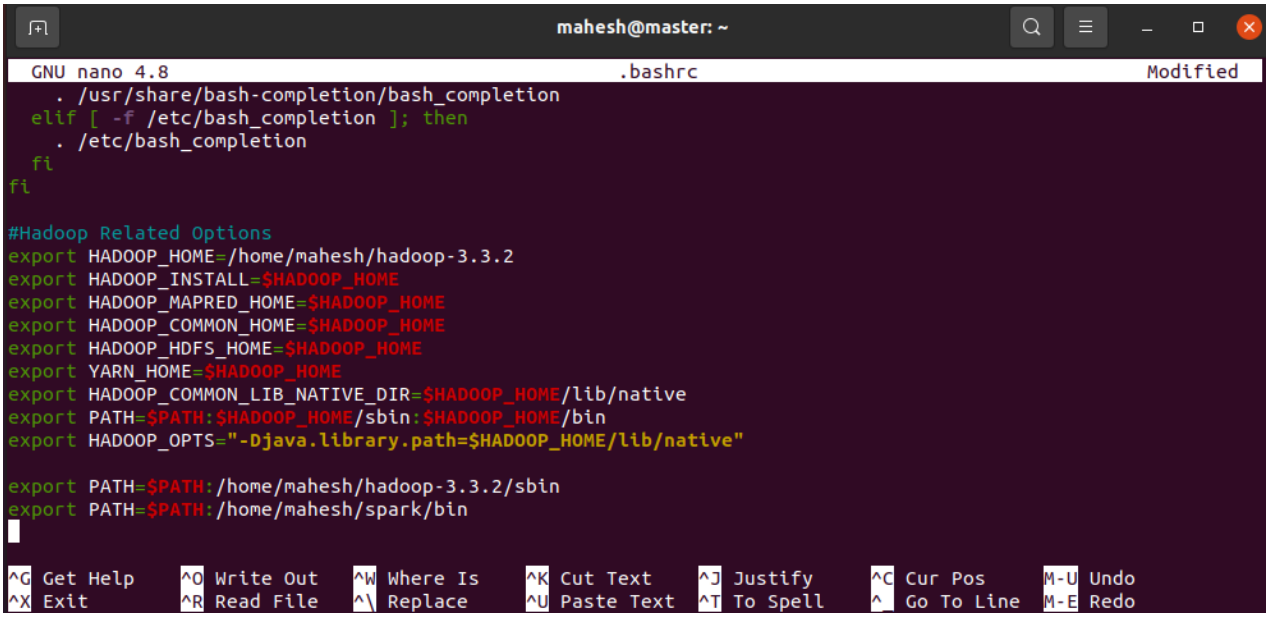


3) Renaming the spark folder from “spark-3.2.3-bin-hadoop3.2.tgz” to “spark”.

4) Download and unzip scala

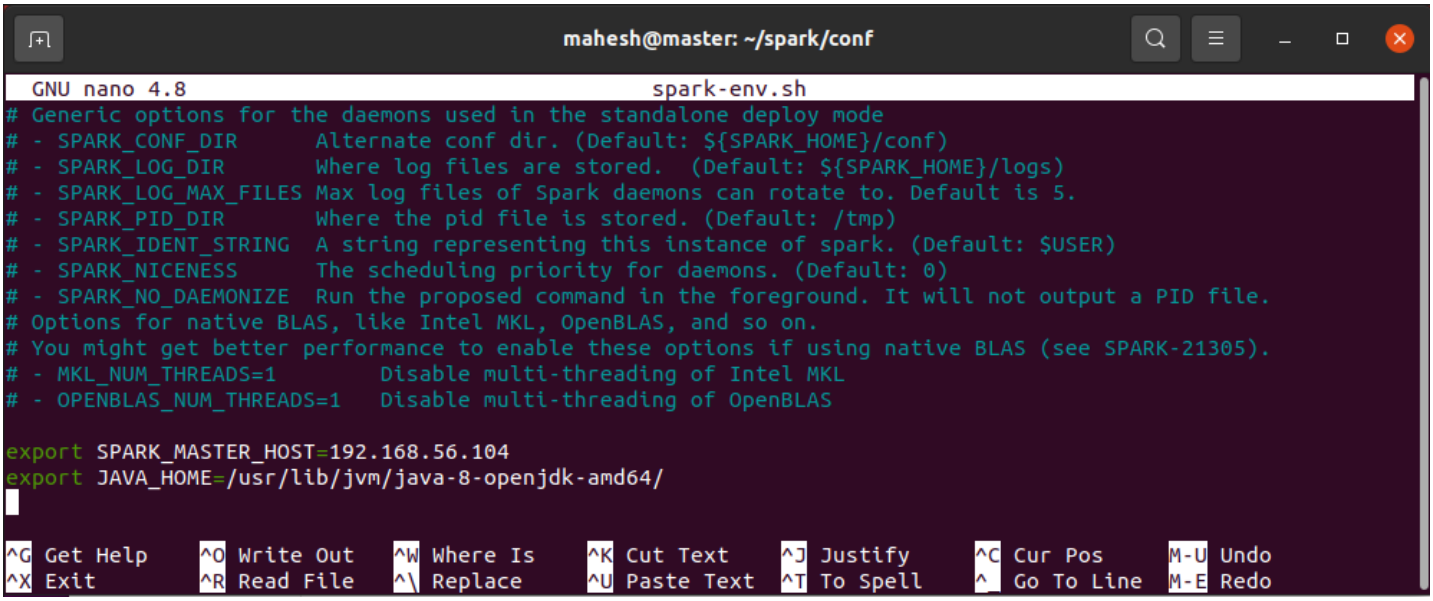
```
mahesh@master:~$ sudo wget https://downloads.lightbend.com/scala/2.11.0/scala-2.11.0.tgz
--2023-05-08 10:17:05--  https://downloads.lightbend.com/scala/2.11.0/scala-2.11.0.tgz
Resolving downloads.lightbend.com (downloads.lightbend.com)... 18.66.41.115, 18.66.41.108, 18.66.41.70, ...
Connecting to downloads.lightbend.com (downloads.lightbend.com)|18.66.41.115|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26007395 (25M) [application/octet-stream]
Saving to: ‘scala-2.11.0.tgz’

scala-2.11.0.tgz                    55%[=====================================>                                ]  13.64M  2.
68MB/s    eta 1scala-2.11.0.tgz                 55%[=====================================>                          ]  13
.81M  2.76MB/s  scala-2.11.0.tgz                58%[======================================>                      ]  14.61M
2.89MB/s    scala-2.11.0.tgz              60%[=======================================>                    ]  15.12M  2.98MB/s
    scala-2.11.0.tgz              63%[========================================>                   ]  15.84M  3.10MB/s    eta 12s
scalascala-2.11.0.tgz               71scalascalascalascala-2.11scalascala-2.11.0.tgz                95%[==============
==========================================>    ]  23.56M  3.84MB/s    eta 3sscala-2.11.0.tgz                100%[=======
=====================================================>]  24.80M  3.82MB/s    in 14s

2023-05-08 10:17:20 (1.76 MB/s) - ‘scala-2.11.0.tgz’ saved [26007395/26007395]

mahesh@master:~$ tar -xvf scala-2.11.0.tgz
scala-2.11.0/
scala-2.11.0/man/
scala-2.11.0/man/man1/
scala-2.11.0/man/man1/scaladoc.1
```

5) Adding the spark bin folder into the path variable.



6) Copying the spark-env template to use it as spark-env.sh, file in the conf folder inside spark.

```
mahesh@master:~$ sudo nano ~/.bashrc
mahesh@master:~$ cd spark/conf
mahesh@master:~/spark/conf$ cp spark-env.sh.template spark-env.sh
mahesh@master:~/spark/conf$ sudo nano spark-env.sh
```


mahesh@master: ~/spark/conf
GNU nano 4.8 spark-env.sh
Generic options for the daemons used in the standalone deploy mode
- SPARK_CONF_DIR Alternate conf dir. (Default: ${SPARK_HOME}/conf)
- SPARK_LOG_DIR Where log files are stored. (Default: ${SPARK_HOME}/logs)
- SPARK_LOG_MAX_FILES Max log files of Spark daemons can rotate to. Default is 5.
- SPARK_PID_DIR Where the pid file is stored. (Default: /tmp)
- SPARK_IDENT_STRING A string representing this instance of spark. (Default: $USER)
- SPARK_NICENESS The scheduling priority for daemons. (Default: 0)
- SPARK_NO_DAEMONIZE Run the proposed command in the foreground. It will not output a PID file.
Options for native BLAS, like Intel MKL, OpenBLAS, and so on.
You might get better performance to enable these options if using native BLAS (see SPARK-21305).
- MKL_NUM_THREADS=1 Disable multi-threading of Intel MKL
- OPENBLAS_NUM_THREADS=1 Disable multi-threading of OpenBLAS
export SPARK_MASTER_HOST=192.168.56.104
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line M-E Redo

7) Copy the workers.template file in spark/conf and edit it to add the worker machines.

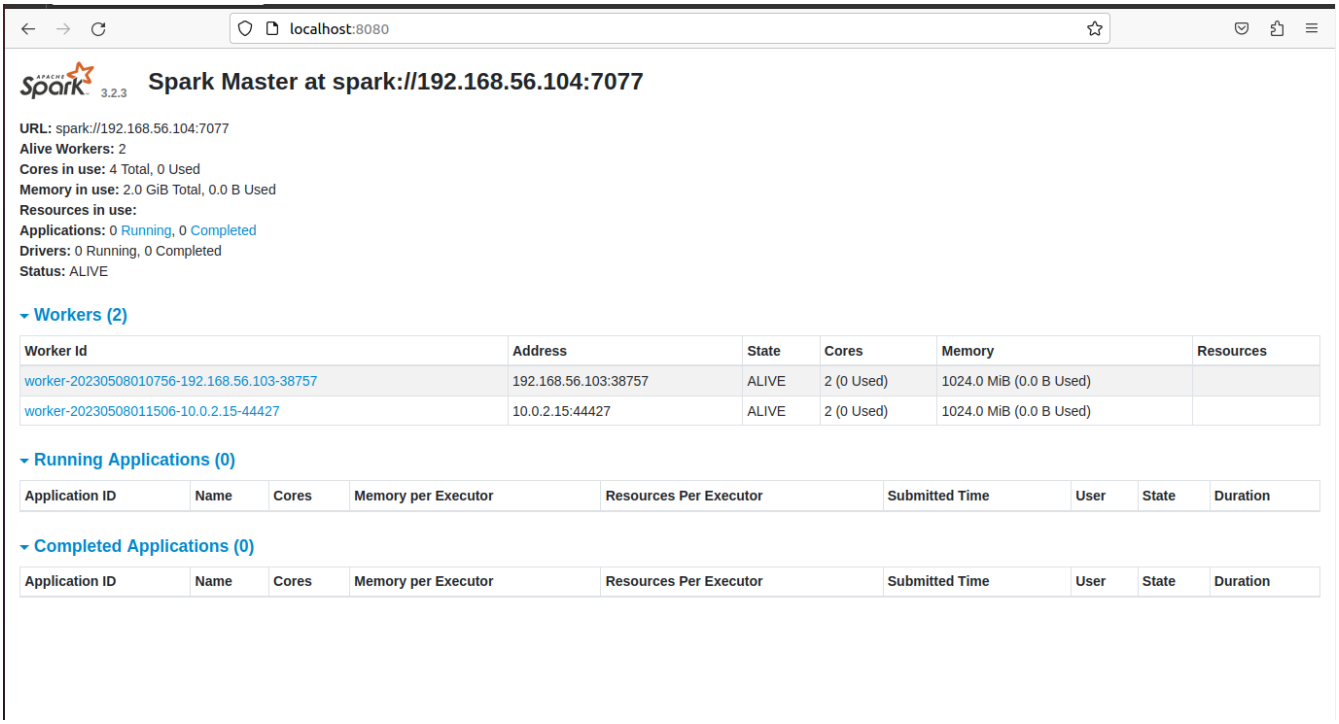
```
mahesh@master:~/spark/conf$ ls
fairscheduler.xml.template  metrics.properties.template   spark-env.sh           workers.template
log4j.properties.template   spark-defaults.conf.template  spark-env.sh.template
mahesh@master:~/spark/conf$ sudo cp workers.template workers
mahesh@master:~/spark/conf$ sudo nano workers
mahesh@master:~/spark/conf$
```

```
  GNU nano 4.8                                  workers                                  Modified
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

# A Spark Worker will be started on each of the machines listed below.
master
slave

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  M-U Undo
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line M-E Redo
```

8) Running the spark node using start-all.sh, command, and checking it in the master machine and the slave machine.

```
mahesh@master:~$ cd spark/sbin/
mahesh@master:~/spark/sbin$ ./start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /home/mahesh/spark/logs/sp
ark-mahesh-org.apache.spark.deploy.master.Master-1-master.out
slave: starting org.apache.spark.deploy.worker.Worker, logging to /home/mahesh/spark/
logs/spark-mahesh-org.apache.spark.deploy.worker.Worker-1-slave.out
master: starting org.apache.spark.deploy.worker.Worker, logging to /home/mahesh/spark
/logs/spark-mahesh-org.apache.spark.deploy.worker.Worker-1-master.out
mahesh@master:~/spark/sbin$ jps
12277 Worker
12153 Master
12347 Jps
mahesh@master:~/spark/sbin$
```

```
mahesh@slave:~$ jps
11324 Worker
11375 Jps
mahesh@slave:~$
```



9) Starting the hdfs file system of Hadoop using start-dfs.sh, command.

```
mahesh@master:~$ cd hadoop-3.3.2/
mahesh@master:~/hadoop-3.3.2$ start-dfs.sh
Starting namenodes on [master]
Starting datanodes
Starting secondary namenodes [master]
mahesh@master:~/hadoop-3.3.2$ jps
13204 SecondaryNameNode
12277 Worker
12153 Master
12875 NameNode
13373 Jps
13021 DataNode
mahesh@master:~/hadoop-3.3.2$
```

```
mahesh@slave:~$ jps
11509 DataNode
11324 Worker
11567 Jps
mahesh@slave:~$
```

10) Adding input files in HDFS and starting spark shell

```
mahesh@master:~$ cd exp5/
mahesh@master:~/exp5$ hdfs dfs -mkdir /exp5
mahesh@master:~/exp5$ hdfs dfs -mkdir /exp5/Input
mahesh@master:~/exp5$ hdfs dfs -ls /exp5
Found 1 items
drwxr-xr-x   - mahesh supergroup          0 2023-05-08 10:33 /exp5/Input
mahesh@master:~/exp5$ sudo nano data.txt
mahesh@master:~/exp5$ hdfs dfs -put data.txt /exp5/Input
mahesh@master:~/exp5$ hdfs dfs -ls /exp5/Input
Found 1 items
-rw-r--r--   2 mahesh supergroup        224 2023-05-08 10:35 /exp5/Input/data.txt
mahesh@master:~/exp5$
```

```
mahesh@master:~$ cd spark/
mahesh@master:~/spark$ spark-shell
23/05/08 10:39:55 WARN Utils: Your hostname, master resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instea
terface enp0s3)
23/05/08 10:39:55 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/05/08 10:40:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
 where applicable
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1683522605222).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.2.3
      /_/

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 1.8.0_362)
Type in expressions to have them evaluated.
Type :help for more information.
```

11) Now in scala spark shell write scala code snippets for WordCount as follows.

```
scala> val text = sc.textFile("hdfs://master:9000/exp5/Input/data.txt")
text: org.apache.spark.rdd.RDD[String] = hdfs://master:9000/exp5/Input/data.txt MapPartitionsRDD[1] at t
extFile at <console>:23

scala> text.collect
res1: Array[String] = Array(My name is father's name is Mr. Dinkar Pachare. My father is an teacher and
we live together in Chandrapur. My father always motivates me to work hard and achieve my goal. I'm luck
y to have my dad. and I'm very proud of him.)

scala> val counts = text.flatMap(line => line.split(" "))
counts: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:23

scala> counts.collect ;
res2: Array[String] = Array(My, name, is, father's, name, is, Mr., Dinkar, Pachare., My, father, is, an,
 teacher, and, we, live, together, in, Chandrapur., My, father, always, motivates, me, to, work, hard, a
nd, achieve, my, goal., I'm, lucky, to, have, my, dad., and, I'm, very, proud, of, him.)

scala> val mapf = counts.map(word => (word,1))
mapf: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:23

scala> mapf.collect ;
res3: Array[(String, Int)] = Array((My,1), (name,1), (is,1), (father's,1), (name,1), (is,1), (Mr.,1), (D
inkar,1), (Pachare.,1), (My,1), (father,1), (is,1), (an,1), (teacher,1), (and,1), (we,1), (live,1), (tog
ether,1), (in,1), (Chandrapur.,1), (My,1), (father,1), (always,1), (motivates,1), (me,1), (to,1), (work,
1), (hard,1), (and,1), (achieve,1), (my,1), (goal.,1), (I'm,1), (lucky,1), (to,1), (have,1), (my,1), (da
d.,1), (and,1), (I'm,1), (very,1), (proud,1), (of,1), (him.,1))

scala> val reducef = mapf.reduceByKey(_+_);
reducef: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:23

scala> reducef.collect ;
res4: Array[(String, Int)] = Array((motivates,1), (is,3), (have,1), (teacher,1), (live,1), (we,1), (Pach
are.,1), (very,1), (lucky,1), (proud,1), (father,2), (together,1), (Chandrapur.,1), (my,2), (My,3), (him
.,1), (me,1), (father's,1), (Mr.,1), (hard,1), (dad.,1), (name,2), (work,1), (to,2), (in,1), (of,1), (Di
nkar,1), (I'm,2), (always,1), (achieve,1), (an,1), (and,3), (goal.,1))

scala>
```

Conclusion: Hence in this experiment we have installed and set up the spark multi-node cluster on two virtual machines having Ubuntu in a virtual box. Then we have executed a word count program in the scala programming language by generating RDD from a data store in the hdfs file system.