

Exercise:

1. Install spark and pyspark (ubuntu only)
2. Run a spark shell and test the installation.
3. Run the wordcount program that you did using hadoop using pyspark.
4. Use the movielens dataset available in the LMS theory page and try to find out for each movie, how are the ratings distributed.

Link:

<https://medium.com/@agusmahari/pyspark-step-by-step-guide-to-installing-pyspark-on-linux-bb8af96ea5e8>

1. Install spark and pyspark (ubuntu only)

- PySpark, a Python library for Apache Spark
- Below steps is to install Apache Spark and using pip install the PySpark library

Step 1: Install Java

```
java -version
```

```
shruthimohan@EDITH:~$ java -version
openjdk version "11.0.24" 2024-07-16
OpenJDK Runtime Environment (build 11.0.24+8-post-Ubuntu-1ubuntu320.04)
OpenJDK 64-Bit Server VM (build 11.0.24+8-post-Ubuntu-1ubuntu320.04, mixed mode, sharing)
```

Step 2. Install Apache Spark

```
wget
```

<https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz>

```
hadoop@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC: /hadoop$ wget https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
--2024-10-08 09:58:40-- https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
Resolving archive.apache.org (archive.apache.org)... 65.108.204.189, 2a01:4f8:1a:a084::2
Connecting to archive.apache.org (archive.apache.org)[65.108.204.189]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 300965906 (287M) [application/x-gzip]
Saving to: 'spark-3.2.0-bin-hadoop3.2.tgz'

spark-3.2.0-bin-hadoop3.2.tgz 100%[=====] 287.02M 1.50MB/s in 7m 4s
2024-10-08 09:57:45 (693 KB/s) - 'spark-3.2.0-bin-hadoop3.2.tgz' saved [300965906/300965906]
```

extracted file: `tar -xvzf spark-3.2.0-bin-hadoop3.2.tgz`

```
hadoop@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$ tar -xvzf spark-3.2.0-bin-hadoop3.2.tgz
spark-3.2.0-bin-hadoop3.2/
spark-3.2.0-bin-hadoop3.2/NOTICE
spark-3.2.0-bin-hadoop3.2/kubernetes/
spark-3.2.0-bin-hadoop3.2/kubernetes/tests/
spark-3.2.0-bin-hadoop3.2/kubernetes/tests/python_executable_check.py
spark-3.2.0-bin-hadoop3.2/kubernetes/tests/autoscale.py
spark-3.2.0-bin-hadoop3.2/kubernetes/tests/worker_memory_check.py
spark-3.2.0-bin-hadoop3.2/kubernetes/tests/py_container_checks.py
spark-3.2.0-bin-hadoop3.2/kubernetes/tests/decommissioning.py
spark-3.2.0-bin-hadoop3.2/kubernetes/tests/pyfiles.py
spark-3.2.0-bin-hadoop3.2/kubernetes/tests/decommissioning_cleanup.py
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/decom.sh
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/entrypoint.sh
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/R/
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/R/Dockerfile
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/python/
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/python/Dockerfile
spark-3.2.0-bin-hadoop3.2/kubernetes/dockerfiles/spark/Dockerfile
spark-3.2.0-bin-hadoop3.2/jars/
spark-3.2.0-bin-hadoop3.2/jars/RoaringBitmap-0.9.0.jar
spark-3.2.0-bin-hadoop3.2/jars/spark-graphx_2.12-3.2.0.jar
spark-3.2.0-bin-hadoop3.2/jars/kubernetes-model-metrics-5.4.1.jar
spark-3.2.0-bin-hadoop3.2/jars/okhttp-3.12.12.jar
spark-3.2.0-bin-hadoop3.2/jars/hive-shims-common-2.3.9.jar
spark-3.2.0-bin-hadoop3.2/jars/spire_2.12-0.17.0.jar
spark-3.2.0-bin-hadoop3.2/jars/logging-interceptor-3.12.12.jar
spark-3.2.0-bin-hadoop3.2/jars/hive-serde-2.3.9.jar
spark-3.2.0-bin-hadoop3.2/jars/zookeeper-3.6.2.jar
spark-3.2.0-bin-hadoop3.2/jars/kryo-shaded-4.0.2.jar
spark-3.2.0-bin-hadoop3.2/jars/hive-jdbc-2.3.9.jar
spark-3.2.0-bin-hadoop3.2/jars/commons-logging-1.1.3.jar
spark-3.2.0-bin-hadoop3.2/jars/commons-compiler-3.0.16.jar
spark-3.2.0-bin-hadoop3.2/jars/jdo-api-3.0.1.jar
spark-3.2.0-bin-hadoop3.2/jars/JLargeArrays-1.5.jar
spark-3.2.0-bin-hadoop3.2/jars/metrics-jvm-4.2.0.jar
spark-3.2.0-bin-hadoop3.2/jars/httpcore-4.4.14.jar
spark-3.2.0-bin-hadoop3.2/jars/commons-codec-1.15.jar
spark-3.2.0-bin-hadoop3.2/jars/jcl-over-slf4j-1.7.30.jar
spark-3.2.0-bin-hadoop3.2/jars/spark-repl_2.12-3.2.0.jar
spark-3.2.0-bin-hadoop3.2/jars/stax-api-1.0.1.jar
spark-3.2.0-bin-hadoop3.2/jars/hive-storage-api-2.7.2.jar
spark-3.2.0-bin-hadoop3.2/jars/zookeeper-jute-3.6.2.jar
spark-3.2.0-bin-hadoop3.2/jars/spark-sketch_2.12-3.2.0.jar
spark-3.2.0-bin-hadoop3.2/jars/kubernetes-model-policy-5.4.1.jar
spark-3.2.0-bin-hadoop3.2/jars/jline-2.14.6.jar
spark-3.2.0-bin-hadoop3.2/jars/jackson-dataformat-yaml-2.12.3.jar
spark-3.2.0-bin-hadoop3.2/jars/kubernetes-model-coordination-5.4.1.jar
spark-3.2.0-bin-hadoop3.2/jars/dropwizard-metrics-hadoop-metrics2-reporter-0.1.2.jar
```

Move the extracted folder to the /opt directory: `sudo mv`

`spark-3.2.0-bin-hadoop3.2 /opt/spark`

```
hadoop@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$ sudo mv spark-3.2.0-bin-hadoop3.2 /opt/spark
[sudo] password for hadoop:
hadoop@snuce-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$
```

Step 3. Set Up Environment Variables

To set permanent environment variables for a single user click on home in terminal

```
sudo nano ~/.bashrc
```

```

GNU nano 4.8 /home/shruthimohan/.bashrc
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc.
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion" # This loads nvm bash_completion
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

```

```

hadoop@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$ sudo nano ~/.bashrc
hadoop@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$ source ~/.bashrc
hadoop@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$

```

Step 4. Install PySpark

Install PySpark using pip: `pip install pyspark`

```

hadoop@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$ pip install pyspark
Defaulting to user installation because normal site-packages is not writeable
Collecting pyspark
  Downloading pyspark-3.5.3.tar.gz (317.3 MB)
    317.3/317.3 MB 4.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.7 (from pyspark)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl.metadata (1.5 kB)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
    200.5/200.5 kB 6.2 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.3-py2.py3-none-any.whl size=317840630 sha256=ad620f0845e816c2194eab84cefa892c78d7517543bf64b213d4dd2f29b33e6
  Stored in directory: /home/hadoop/.cache/pip/wheels/97/f5/c0/947e2c0942b361ffe58651f36bd7f13772675b3863fd63d1b1
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.7 pyspark-3.5.3
hadoop@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$

```

Step 5. Verify PySpark Installation: `pyspark`

```

hadoop@snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~/hadoop$ pyspark
Python 3.11.7 (main, Dec 15 2023, 18:12:31) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
24/10/08 10:12:25 WARN Utils: Your hostname, snucse-HP-Pro-Tower-400-G9-PCI-Desktop-PC resolves to a loopback address: 127.0.1.1; using 10.23.22.170 instead (on interface eno1)
24/10/08 10:12:25 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).
24/10/08 10:12:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to

  ____      _
 / ___|  __| | | |
 \___ \  | | | | | |
  ___) | | | | | | |
 |_____|_|_|_|_|_|_|

 version 3.2.0

Using Python version 3.11.7 (main, Dec 15 2023 18:12:31)
Spark context Web UI available at http://10.23.22.170:4040
Spark context available as 'sc' (master = local[*], app id = local-1728362546168).
SparkSession available as 'spark'.
>>>

```

Run pyspark:

3. Run the wordcount program that you did using hadoop using pyspark.

Step 1: cd word_count (directory created in prev ex)

```
GNU nano 4.8                                     file
Hi
hello
ganesh
shrumo
bagiya
cynddia
harini
suruthi
saanji
krishna
sholini
sholini
krishna
saanji
saanji
bagiya
bagiya
bagiya
bagiya
ganesh
shrumo
```

Step 2: create a SparkSession and sparkContext

```
>>> # Create SparkSession and sparkcontext
parkSession
spark = SparkSession.builder\
    >>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder\
...     .master("local")\
...     .appName('Firstprogram')\
...     .getOrCreate()
ontext>>> sc=spark.sparkContext
```

Step 3: Read the input file as RDD and provide transformations to calculate the count of each word in our file.

```
>>> text_file = sc.textFile("file1.txt")
>>> counts = text_file.flatMap(lambda line: line.split(" ")) \
...     .map(lambda word: (word, ...)) \
...     .map(lambda word: (word, 1)) \
...     .reduceByKey(lambda x, y: x + y)
```

Step 4: Output

```
>>> # Printing each word with its respective count
output = counts.collect()
for (word, count) in output:
    print(word, count)
[Stage 0: >] (0 + 2) / 2]
>>> for (word, count) in output:
...     print("%s: %i" % (word, count))
...
Hi: 1
ganesh: 2
bagiya: 9
saanji: 3
krishna: 2
hello: 1
shrumo: 2
cynddia: 4
harini: 4
suruthi: 4
sholini: 4
```

NOTES:

1. `SparkSession` is the entry point for creating a Spark application.
2. `.master("local")`: Specifies that the Spark application should run in "local" mode, meaning it will use local threads instead of connecting to a Spark cluster
3. `sc = spark.sparkContext`: Gets the Spark context (`sc`) from the `SparkSession`. `sc` is the main entry point for Spark's RDD-based API, providing access to core Spark functions like data loading, transformations, and actions.
4. RDD= Resilient Distributed Dataset
5. `flatMap(lambda line: line.split(" "))`: Applies a function to each line in `text_file`, splitting it into individual words. `flatMap` flattens the result so each word becomes a separate element in the RDD.

Program done in jupyter notebook:

```
[7]: import pyspark

# Create SparkSession and sparkcontext
from pyspark.sql import SparkSession
spark = SparkSession.builder\
    .master("local")\
    .appName('wordcount')\
    .getOrCreate()
sc=spark.sparkContext

[8]: # Read the input file and Calculating words count
text_file = sc.textFile("word_count/file1.txt")
counts = text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda x, y: x + y)

[10]: output = counts.collect()
for (word, count) in output:
    print('{}\t{}'.format(word, count))

Hi      1
hello   1
ganesh  2
shrumo  2
bagiya  9
cynddia 4
harini  4
suruthi 4
saanji  3
krishna 2
sholini 4

[11]: # Stopping Spark-Session and Spark context
sc.stop()
spark.stop()
```

4. Use the movielens dataset available and try to find out for each movie, how are the ratings distributed.

Code:

```
from pyspark.sql import SparkSession
```

```
from pyspark.sql.functions import col, count
```

```
# Initialize Spark session
```

```

spark = SparkSession.builder \

    .appName("MovieLensRatingsDistribution") \

    .getOrCreate()

# Load dataset

data_path = "movielens.txt"

movies_df = spark.read.csv(data_path, sep='\t', inferSchema=True) \

    .toDF("user_id", "movie_id", "rating", "timestamp")

# Calculate the distribution of ratings for each movie

rating_distribution = movies_df.groupBy("movie_id", "rating") \

    .agg(count("rating").alias("rating_count")) \

    .orderBy("movie_id", "rating")

# Show the results

rating_distribution.show()

```

```

# Show the results
rating_distribution.show()

```

```

+-----+-----+-----+
|movie_id|rating|rating_count|
+-----+-----+-----+
|      1|      1|           8|
|      1|      2|          27|
|      1|      3|          96|
|      1|      4|         202|
|      1|      5|         119|
|      2|      1|           8|
|      2|      2|          17|
|      2|      3|          55|
|      2|      4|          42|
|      2|      5|           9|
|      3|      1|          11|
|      3|      2|          20|
|      3|      3|          25|
|      3|      4|          23|
|      3|      5|          11|
|      4|      1|           6|
|      4|      2|          24|
|      4|      3|          57|
|      4|      4|          93|
|      4|      5|          29|
+-----+-----+-----+
only showing top 20 rows

```

