Computer Science & Information Systems

Big Data Systems – Spark Lab Sheet:1

Spark Shell Operations

1. Objective

Students should be able to

A. Get familiarity with the Apache Spark system
B. Get hands-on experience with Spark Shell commands

This lab sheet provides a quick introduction to using Spark. This exercise will introduce the API through Spark's interactive shell (in Python), then next labs will show how to write applications in Python.

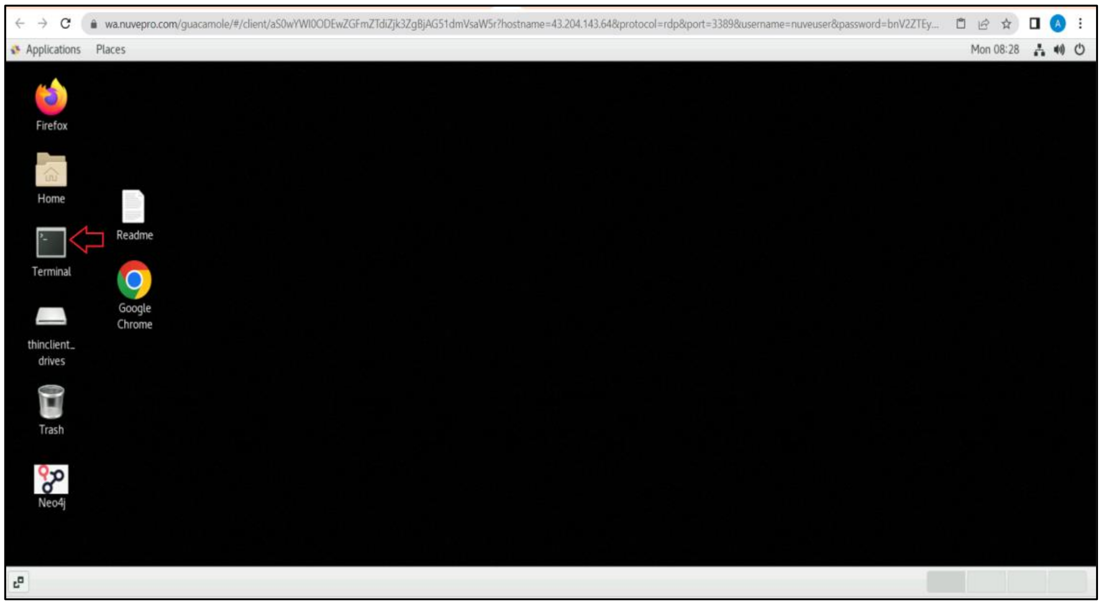
Note that, before Spark 2.0, the main programming interface of Spark was the Resilient Distributed Dataset (RDD). After Spark 2.0, RDDs are replaced by Dataset, which is strongly-typed like an RDD, but with richer optimizations under the hood. The RDD interface is still supported, and one can get a more detailed reference at the RDD programming guide. It is highly recommended to switch to use Dataset, which has better performance than RDD, but as RDDs are basic blocks of Spark, we will be getting familiarity with them in this lab sheet.

At a high level, every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster. The main abstraction Spark provides is a resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to persist an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.

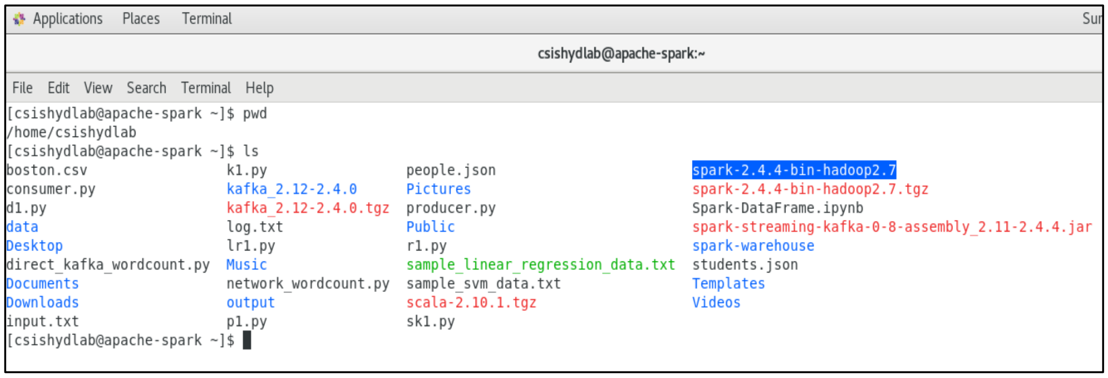
2. Steps to be performed

Data Preparations -

a) Open the terminal by clicking on icon on desktop


wa.nuvepro.com/guacamole/#/client/aS0wYWI0ODEwZGFmZTdiZjk3ZgBjAG51dmVsaW5r?hostname=43.204.143.64&protocol=rdp&port=3389&username=nuveuser&password=bnV2ZTEy...
Applications
Places
Mon 08:28
Firefox
Home
Readme
Terminal
Google Chrome
thinclient_drives
Trash
Neo4j

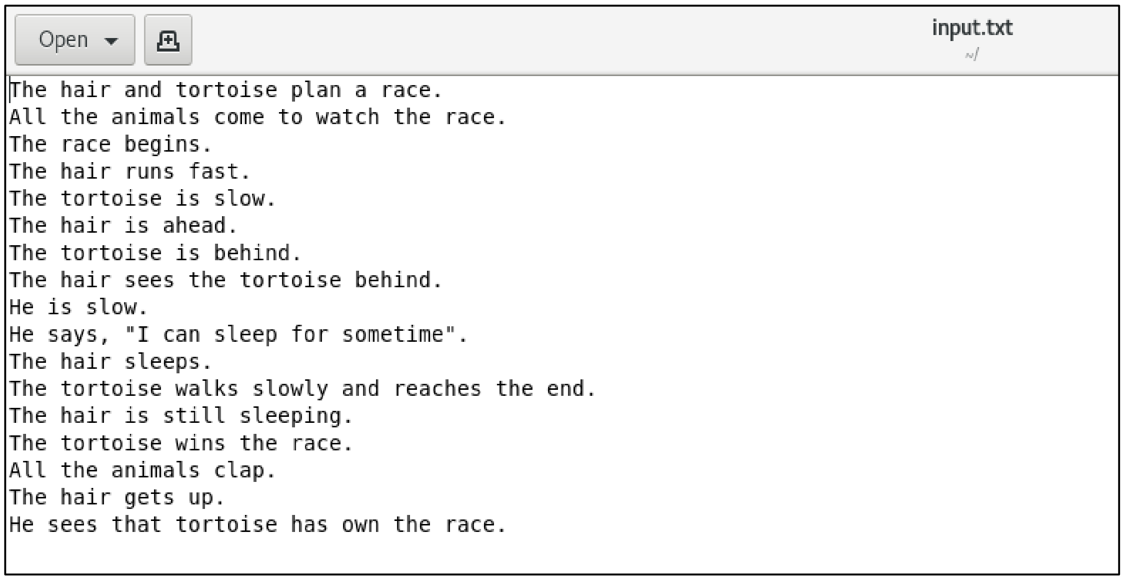
b) Look at the current directory and also file listings in it. It must have a spark installation directory present in it. Commands like pwd, ls can be used for it.



```
[csishydlab@apache-spark ~]$ pwd
/home/csishydlab
[csishydlab@apache-spark ~]$ ls
boston.csv              k1.py                people.json                      spark-2.4.4-bin-hadoop2.7
consumer.py             kafka_2.12-2.4.0     Pictures                         spark-2.4.4-bin-hadoop2.7.tgz
d1.py                   kafka_2.12-2.4.0.tgz producer.py                      Spark-DataFrame.ipynb
data                    log.txt              Public                           spark-streaming-kafka-0-8-assembly_2.11-2.4.4.jar
Desktop                 lr1.py               r1.py                            spark-warehouse
direct_kafka_wordcount.py Music              sample_linear_regression_data.txt students.json
Documents               network_wordcount.py sample_svm_data.txt              Templates
Downloads               output               scala-2.10.1.tgz                 Videos
input.txt               p1.py                sk1.py
[csishydlab@apache-spark ~]$
```

c) Prepare the input text file using any file editor. Copy and paste the content present in the attached input.txt file in this file.

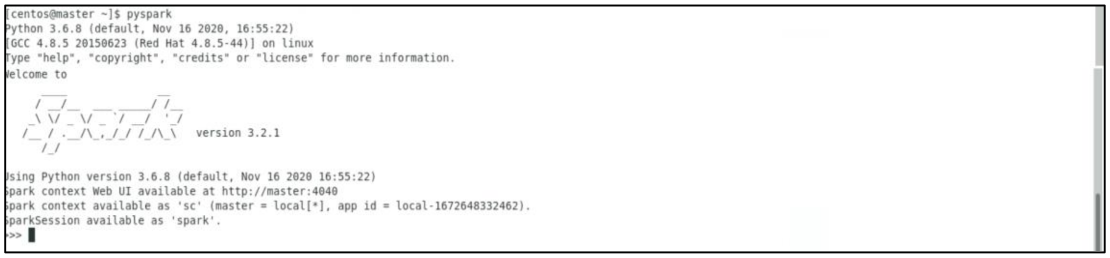
```
[centos@master ~]$ gedit input.txt
```



```
The hair and tortoise plan a race.
All the animals come to watch the race.
The race begins.
The hair runs fast.
The tortoise is slow.
The hair is ahead.
The tortoise is behind.
The hair sees the tortoise behind.
He is slow.
He says, "I can sleep for sometime".
The hair sleeps.
The tortoise walks slowly and reaches the end.
The hair is still sleeping.
The tortoise wins the race.
All the animals clap.
The hair gets up.
He sees that tortoise has own the race.
```

Using the Shell

a) Enter the Spark shell using the 'pyspark' .



```
[centos@master ~]$ pyspark
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.2.1
      /_/

Using Python version 3.6.8 (default, Nov 16 2020 16:55:22)
Spark context Web UI available at http://master:4040
Spark context available as 'sc' (master = local[*], app id = local-1672648332462).
SparkSession available as 'spark'.
>>> 
```

In the PySpark shell, a special interpreter-aware SparkContext is already created, in the variable called sc. Making your own SparkContext will not work. You can set which master the context connects to using the --master argument, and you can add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to --py-files. You can also add dependencies (e.g. Spark Packages) to your shell session by supplying a comma-separated list of Maven coordinates to the --packages argument. Any additional repositories where dependencies might exist (e.g. Sonatype) can be passed to the --repositories argument. Any Python dependencies a Spark package has (listed in the requirements.txt of that package) must be manually installed using pip when necessary.

Using the Resilient Distributed Datasets (RDDs)

Spark revolves around the concept of a resilient distributed dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.

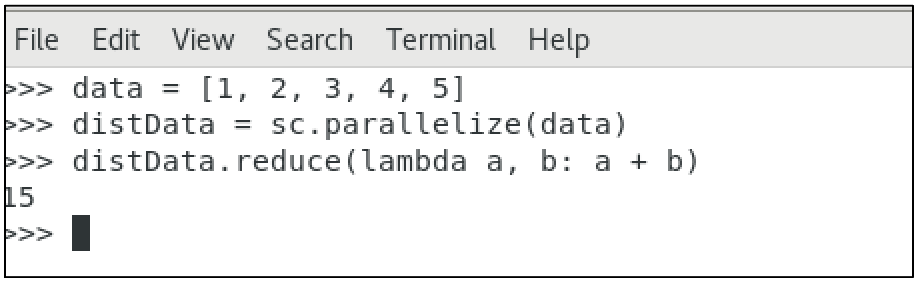
b) Parallelized Collections

Parallelized collections are created by calling SparkContext's parallelize method on an existing iterable or collection in your driver program. The elements of the collection are copied to form a distributed dataset that can be operated on in parallel. For example, here is how to create a parallelized collection holding the numbers 1 to 5:

```
>>>data = [1, 2, 3, 4, 5]
>>>distData = sc.parallelize(data)
```

Once created, the distributed dataset (distData) can be operated on in parallel. For example, we can call distData.reduce(lambda a, b: a + b) to add up the elements of the list.

```
>>>distData.reduce(lambda a, b: a + b)
```



```
>>> data = [1, 2, 3, 4, 5]
>>> distData = sc.parallelize(data)
>>> distData.reduce(lambda a, b: a + b)
15
>>> 
```

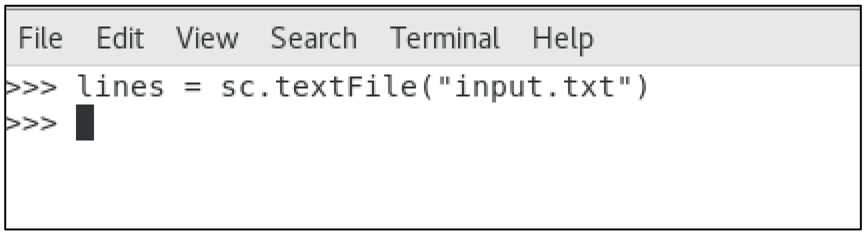
c) Using External Datasets

PySpark can create distributed datasets from any storage source supported by Hadoop, including your local file system, HDFS, Cassandra, HBase, Amazon S3, etc. Spark supports text files, SequenceFiles, and any other Hadoop InputFormat.

Text file RDDs can be created using SparkContext's textFile method. This method takes an URI for the file (either a local path on the machine, or a hdfs://, s3a://, etc URI) and reads it as a collection of lines. Here is an example invocation:

```
>>> lines = sc.textFile("input.txt")
```

Once created, distFile can be acted on by dataset operations.



```
>>> lines = sc.textFile("input.txt")
>>>
```

d) Several operations can be performed on the text RDDs as follows

Provides the count of lines present in RDD

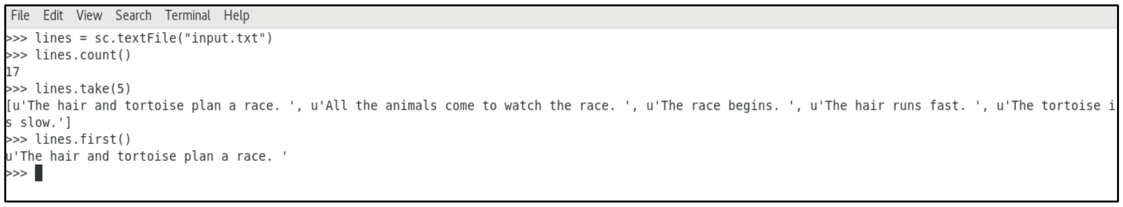
```
>>>lines.count()
```

Read the first 5 items of RDD

```
>>>lines.take(5)
```

Read the first item of RDD

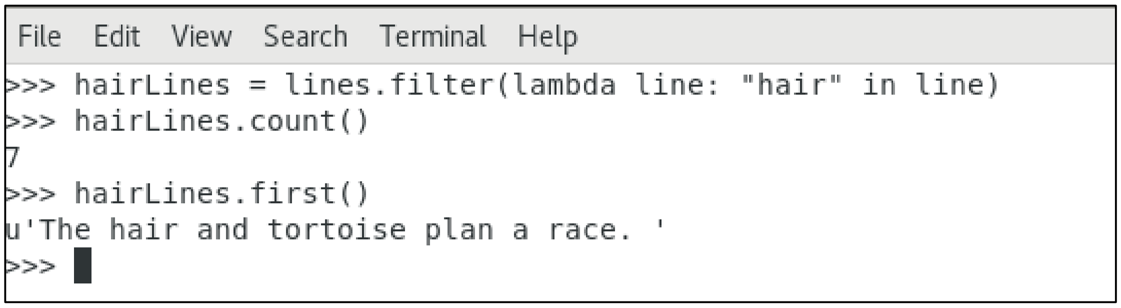
```
>>>lines.first()
```



```
>>> lines = sc.textFile("input.txt")
>>> lines.count()
17
>>> lines.take(5)
[u'The hair and tortoise plan a race. ', u'All the animals come to watch the race. ', u'The race begins. ', u'The hair runs fast. ', u'The tortoise i
s slow.']
>>> lines.first()
u'The hair and tortoise plan a race. '
>>>
```

e) We can extract the lines containing the exact words as follows

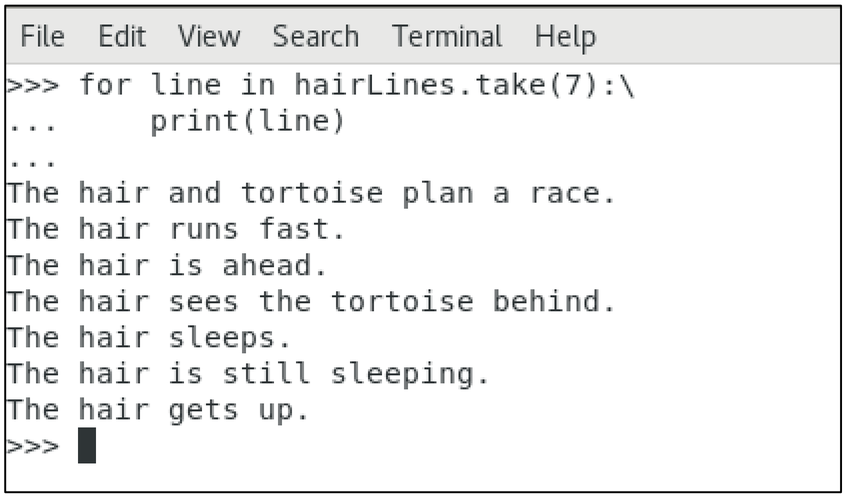
```
>>>hairLines = lines.filter(lambda line: "hair" in line)
>>>hairLines.count()
>>>hairLines.first()
```



```
>>> hairLines = lines.filter(lambda line: "hair" in line)
>>> hairLines.count()
7
>>> hairLines.first()
u'The hair and tortoise plan a race. '
>>>
```

f) One can iterate over each item in the RDD as follows

```
>>> for line in hairLines.take(7):\
...     print(line)
...
```



```
>>> for line in hairLines.take(7):\
...     print(line)
...
The hair and tortoise plan a race.
The hair runs fast.
The hair is ahead.
The hair sees the tortoise behind.
The hair sleeps.
The hair is still sleeping.
The hair gets up.
>>>
```

l) **Performing RDD Operations**

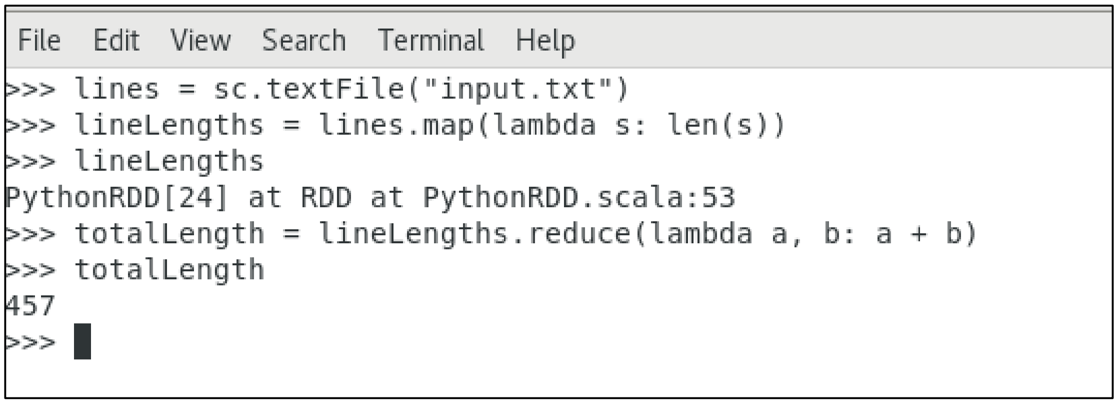
RDDs support two types of operations: transformations, which create a new dataset from an existing one, and actions, which return a value to the driver program after running a computation on the dataset. For example, map is a transformation that passes each dataset element through a function and returns a new RDD representing the results. On the other hand, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel reduceByKey that returns a distributed dataset).

All transformations in Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently. For example, we can realize that a dataset created through map will be used in a reduce and return only the result of the reduce to the driver, rather than the larger mapped dataset.

By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also persist an RDD in memory using the persist (or cache) method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

Just see how the following code counts the characters present in the all lines of given input file.

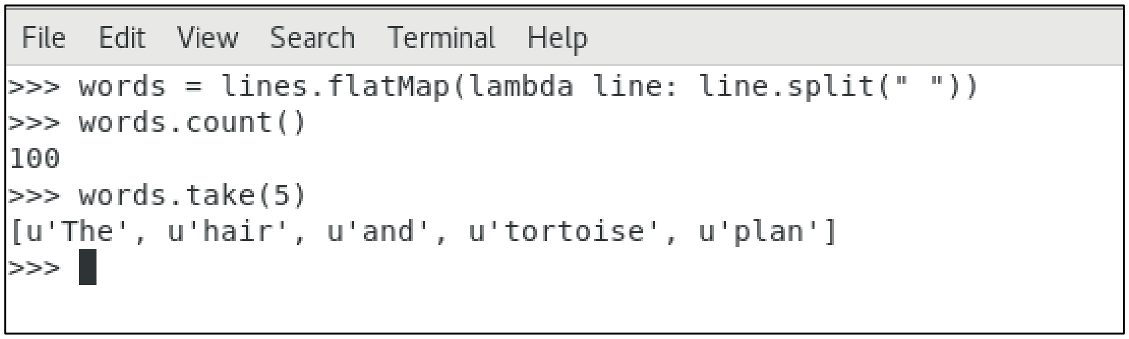
```
>>> lines = sc.textFile("input.txt")
>>> lineLengths = lines.map(lambda s: len(s))
>>> lineLengths
>>> totalLength = lineLengths.reduce(lambda a, b: a + b)
>>> totalLength
```



```
>>> lines = sc.textFile("input.txt")
>>> lineLengths = lines.map(lambda s: len(s))
>>> lineLengths
PythonRDD[24] at RDD at PythonRDD.scala:53
>>> totalLength = lineLengths.reduce(lambda a, b: a + b)
>>> totalLength
457
>>>
```

g) Let’s try out a simple word count program using the RDDs

```
>>>words = lines.flatMap(lambda line: line.split(" "))
>>>words.count()
>>>words.take(5)
```



```
>>> words = lines.flatMap(lambda line: line.split(" "))
>>> words.count()
100
>>> words.take(5)
[u'The', u'hair', u'and', u'tortoise', u'plan']
>>>
```

h) Exit the shell using the exit() command.

```
>>>exit()
```

3. Outputs/Results

- Students should be able to see the usage of spark shell for the RDD creations and transformations.

4. Observations

- Students carefully needs to observe the syntax the python commands used for RDD creation, transformations.

References

A. Spark Documentation
B. RDD Programming Guide