Task 03: Spark Practice

For this task, you need to get Spark up and running. Installing it with hadoop-built-in and single node is the easiest. But I will encourage you to go through a multiple cluster installation so that you understand how things are connected to each other. All these steps are covered in Task 0 so choose whichever one you want and proceed with it.

Task 0a: Connectivity

In case you want to connect to the university servers already configured with hadoop and spark, you will need to provide your credentials via the following details through ssh.

username:	bda-XXP-YYYY (replace XXP-YYYY. e.g bda-21P-0001)
Password:	whät you configured for hadoop
IP:	121.52.146.108 (connect via ssh / putty)

Task 0b: Installation (Single Node with Bundled Hadoop)

If you want to work with this mode, then you can download spark from http://spark.apache.org/downloads.html. Select “Pre-built for Apache Hadoop 3.3 and later” which gives you spark in the form of pre-compiled binaries. The first step is to extract the archive file.

```
tar xvzf spark-3.5.1-bin-hadoop3.tgz -C ~/spark
```

which will extract the files to spark folder in your home directory. The spark folder will contain another folder of spark-3.5.1-bin-hadoop3. You can move its content up one level to spark, or any other location you want. The rest of this walk-through assumes you move all contents to spark folder. Whichever location is finalized, add it to your environment variables by editing the .bashrc file as:

```
nano ~/.bashrc
```

and add to the end of the file the following:

```
export SPARK_HOME=~/spark

export PATH=$PATH:$SPARK_HOME
```

```
source ~/.bashrc
```

You can then directly run python spark as:

```
bin/pyspark
```

You can view status of various spark jobs submitted by accessing the URL: http://localhost:4040

Task 0c: Linking with Jupyter

To link with jupyter, add the following additional exports to .bashrc file:

```
export PYSPARK_PYTHON=/usr/bin/python3

export PYSPARK_DRIVER_PYTHON=jupyter

export PYSPARK_DRIVER_PYTHON_OPTS='notebook --no-browser --port=8889'
```

Now, when you run bin/pyspark, it will show you the following prompt

```
[I 2024-03-07 10:03:51.433 ServerApp] Serving notebooks from local directory: /home/omar
[I 2024-03-07 10:03:51.433 ServerApp] Jupyter Server 2.12.5 is running at:
[I 2024-03-07 10:03:51.433 ServerApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).

[C 2024-03-07 10:03:51.435 ServerApp]

    To access the server, open this file in a browser:
        file:///home/omar/.local/share/jupyter/runtime/jpserver-56146-open.html

    Or copy and paste one of these URLs:
        http://localhost:8889/tree?token=d1e0d5b8d22110ae51e3e7f4361ade2c02aa2374c79cdb71
        http://127.0.0.1:8889/tree?token=d1e0d5b8d22110ae51e3e7f4361ade2c02aa2374c79cdb71
```

Copy paste the URL (with full token) to your browser and you will be able to run spark through Jupyter.

Task 0d: Installation (Multiple Nodes with Existing Hadoop)

To install spark standalone (with an existing hadoop installation), visit http://spark.apache.org/downloads.html and select “Pre-built for user Specified Apache Hadoop”. Then, we proceed with extraction:

```
tar xvzf spark-3.5.1-bin-without-hadoop.tgz -C /opt
```

You can also create a soft link (short-cut) to this folder as:

```
ln -sf /opt/spark-3.5.1-bin-without-hadoop /opt/spark
```

Next, we provide some configuration settings in the bashrc file as:

```
nano ~/.bashrc
```

The following configuration can be given:

```
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop

export SPARK_HOME=/opt/spark

export SPARK_DIST_CLASSPATH=$(hadoop classpath)

export PATH=$PATH:$SPARK_HOME/bin

export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
```

Note that you have specified the environment variable for Hadoop so the following command must be able to give you its location:

```
echo $HADOOP_HOME
```

If it is blank, then you need to specify the location in the bashrc file as well.

```
export HADOOP_HOME=/opt/hadoop
```

Then, specify some configuration settings in the spark-env.sh file as:

```
nano conf/spark-env.sh
```

When opened, the following configuration can be given:

```
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
```

Note from above that hadoop must be on your paths. If it isn’t, then you may need to use the following instead:

```
export SPARK_DIST_CLASSPATH=$(/opt/hadoop/bin/hadoop classpath)
```

Then, specify the Spark Master by creating/editing the following file:

```
nano conf/spark-defaults.conf
```

Then, you can specify the one-liner below to link Spark with the Hadoop Yarn Scheduler:

```
spark.master    yarn
```

The instructions carried so far would have been done on your Master server. Carry out the same steps for each worker node that you intend to add to your cluster.

Once the worker nodes have been added to the cluster, you will now need to specify to spark master the list of all nodes. For this, create/edit the file /opt/spark/conf/worker and ensure that it contains all worker machine hostnames.

```
vim /opt/spark/conf/worker

worker1hostname
worker2hostname
```

If you want the master node to act as a worker node as well, you can add it to the above list. The IP addresses of all machines along with host names need to be already defined in your /etc/hosts files.

The last step is then to launch the spark cluster. For this, on the master node, run the following:

```
sbin/start-master.sh
```

You can also launch the workers directly from the master node by running:

```
sbin/start-workers.sh
```

Note that to start a specific worker, you can use the file sbin/start-worker.sh as well. Alternatively to these separate commands, you can also launch:

```
sbin/start-all.sh
```

To stop the services, you will need to run either of:

```
sbin/stop-all.sh
sbin/stop-master.sh
sbin/stop-workers.sh
```

You can verify that the services (both hadoop and spark) are running by running jps:

```
jps
```

which should show you something similar to the following (note: I have underlined the spark services):

```
28113 DataNode
28290 SecondaryNameNode
27394 Master
32547 Worker
28499 ResourceManager
27971 NameNode
33660 Jps
28607 NodeManager
```

Task 1: Exploring the RDD Space

We start with creating a spark context by running the following:

```
from pyspark import SparkConf, SparkContext
```

```
conf = SparkConf().setMaster("local").setAppName("My App")
sc = SparkContext (conf = conf)
```

In case the spark context is already created (e.g. by Jupyter), you can use the following to fetch the existing context.

```
sc = SparkContext.getOrCreate();
```

You can get more details about the spark context and a log of activities by running sc directly as below:

```
sc
```

The above will point you to the Spark UI where you can assess the environment and statistics related to your installation, spark jobs, etc.

Create a simple RDD object from an array by running:

```
x = [1,2,3,4,5,6,7,8,9,10,11,12];
xRDD = sc.parallelize(x);
```

Here, sc.parallelize creates a partitioned collection of elements for x. You can view its types by running:

```
print(type(x));
print(type(xRDD));
```

To view the number of partitions used in the xRDD object along with its composition, run:

```
xRDD.getNumPartitions()
print(xRDD.glom().collect())
```

If you want more threads with less work, you need to increase the number of partitions. However, a safe figure is 2-3x the number of cores on a machine. If you want less number of threads with more work, you will need to decrease the number of partitions. To change the partition count of an RDD object, you can pass on the number of desired partitions directly when creating the array, or use the RDD's repartition() method. Following are code for creating 5 partitions in both scenarios:

```
xRDD5 = sc.parallelize(x, 5);
xRDD5 = xRDD.repartition(5);
```

To view the actual execution order of workload, you can set an iterator on each partition as below:

```
def f(iterator):
    for x in iterator:
        print(x)
    yield None
```

Then, you can iterate partition by partition using:

```
xRDD.foreachPartition(f)
```

Let's experiment a little bit by splitting the xRDD into odd and even numbers, and then creating their union:

```
xRDDEven = xRDD.filter(lambda y: (y % 2 == 0) in x)
xRDDOdd  = xRDD.filter(lambda y: (y % 2 == 1) in x)

xRDDUnion = xRDDEven.union(xRDDOdd)
```

Now answer the following questions:

Question 1. What is the URL of Spark UI installed on your machine? (Note: See output of sc)

Question 2. What is the class type xRDD object and find the URL of the API documentation fo this

class type.

Question 3. From the URL you have found, find what the function glom() does.

Question 4. What is the difference in output of print(xRDD), print(xRDD.collect()), and print(xRDD.glom().collect())

Question 5. Parallelize an array of size 50. What is the default number of partitions used for this array? Think about what this default number represents.

Question 6. For the array size of 50, repartition it to 7 partitions. Check the composition of the partitions and repot the workload on each thread.

Question 7. Comment on the execution order of partitions.

Question 8. Open the Spark UI and report the Job ID, Job description, and duration (in seconds) of the most time consuming job carried out so far.

Question 9. From the Spark UI, present the Directed Acyclic Graph visualization for your foreachPartition() code.

Question 10. You will notice the introduction of concept of stage with previous question. Explain what it is and how it is numbered.

Question 11. From the Spark UI, report the Directed Acyclic Graph visualization for the Union() code. Report which partition is executed on which machine.

Submission

Send in your code/jupyter notebooks and answer to your questions to get graded for this task.