Assignment 3 - Apache Spark

Important Notes

Submission Guidelines and Marks

This assignment is worth 4 marks in total. Complete all parts of A and B, for a total of **2 marks**. Submit two Python Jupyter notebooks, one for each section. Submit with an .ipynb extension for each notebook. Submit the notebook with the output included. Do not submit a PDF file, a .py file, a screenshot, etc. Please write code in English.

To pass this assignment, you need to get the mark for Section A+B.

The marks are awarded for submitting correct and complete code to both sections. Completing Section A (or B) only gives **no marks.**

For example, if the question asks you to count something, the code should do that and the result should be printed out in the notebook. Poorly formatted code or code with many spelling mistakes may be penalized. Do not submit blocks of commented-out code, it makes it difficult for us to know what to mark. We must be able to run the code on the cluster without modifying it.

Section C and D are **each worth 1 mark**, for a total of 2 further marks counting toward higher grades. For these sections, submit a single PDF file with your written answers. Label your answers clearly with the question numbers. The document must be well-formatted. Poor spelling, grammar, and punctuation may be penalized.

In all, you should submit 2 Jupyter notebooks, and 1 PDF file.

These marks are a guideline, where there are mistakes in some sections. We'll make a judgment about the overall mark. I strongly recommend that you attempt all questions. **Work individually for all sections of this assignment**.

Configuring your VM and connecting to the Spark Cluster

We will be using a shared Spark cluster for this assignment. You will deploy your application to this cluster, where it will occupy resources until terminated. We start by opening the Web GUI for the Spark cluster (which has already been setup):

1. Look in the list of VMs and find the public (ie floating) IP of the VM named `spark-master`
2. From your laptop, connect to the Spark web GUI running on this machine on port 8080:
   http://sparkmasterpublicip:8080
3. This port is open to the whole Internet, so it is protected with credentials:
   `Username: de1-spark-user Password: 2vegano1micon8`
4. Explore the UI. Observe:
   a. The total memory and cores (and how much are currently used).
   b. The details for each machine in the cluster.

    c. The list of running applications, and the resources used by each. Later on, If you can see there are insufficient resources, you can request on Slack and we can deploy more machines to the cluster.
    d. The 'kill' button that can be used to terminate your application.
5. You need to use this page to check the resource usage of your own applications, whether they have started/stopped successfully, and so on. Keep a tab open whenever you are working with Spark!

To actually run an application, you need to create a VM to host a Python notebook for your Spark code.

1. Create a "**small**" instance, with the base image "**Ubuntu 22.04 - 2023.01.07**".
2. Add your VM to the security group "**spark-cluster-clients-for-student-machines**"
3. Connect to the VM as you normally do. Optional, but recommended, is to setup port forwarding for your connection. The Spark 'driver' application(s) running on your VM will serve HTTP GUIs on port 4040 and successive ports. We can also setup forwarding to the HDFS web GUI running on the Spark cluster (because it does not have a password). For that, you will need the **private** IP of the Spark master node:

```
ssh ubuntu@yourvmpublicip -i mykey.pem \
      -L 4040:localhost:4040 \
      -L 4041:localhost:4041 \
      -L 4042:localhost:4042 \
      -L 4043:localhost:4043 \
      -L 4044:localhost:4044 \
      -L 9870:privateipofsparkmasternode:9870
```

   You should now be able to browse the HDFS web GUI from your laptop at:
   http://localhost:9870
   Note that you can "Browse the filesystem" (under Utilities).
   The other forwarding rules won't point to anything until we start a Spark application.
   These rules only apply for the duration of the SSH connection, and need to be specified each time you connect.
4. On the VM, run: `apt update -y`
5. Install the following packages: `python3-pip net-tools openjdk-11-jdk-headless`
6. There is no DNS facility in Uppmax for VMs to resolve eachother's hostnames. We use the hosts file as a workaround. Run the following code in the shell:

```
for i in {1..4};
do
 for j in {1..255};
 do
     echo "192.168.$i.$j host-192-168-$i-$j" | sudo tee -a /etc/hosts
 done
done
```

   Also, set the hostname of the your VM according to this scheme:
   `sudo hostname host-$(hostname -I | awk '{$1=$1};1' | sed 's/\./-/'g)`
   Verify it is set correctly:
   `hostname`
   And set it again on future logins:
   `echo "sudo hostname host-$(hostname -I | awk '{$1=$1};1' | sed 's/\./-/'g)" | sudo tee -a /home/ubuntu/.profile`

7. Set the JAVA_HOME environment variable (now and for the future):

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
echo JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64 | sudo tee -a /etc/environment
```

8. The version of PySpark must exactly match the cluster:

```
pip3 install pyspark==3.2.3
pip3 install notebook
```

9. Start your notebook server, login (to its web GUI) and create a blank Python3 notebook.

Running Your Spark Application

For your notebook, you will need to know the private (192.168.x.x) IP address of the Spark master node, which will be visible in the list of instances. Note that the Spark master is also the Hadoop namenode, and you will need to specify **port 9000** in the `hdfs://` URI in your code to open datasets.

For this assignment, we will be sharing a Spark cluster. We can learn about the complexity of working with a shared cluster, and it allows us to more efficiently share resources within the group. It is very important that you follow these instructions to ensure that your application does not occupy excessive resources in the cluster (which prevents other students from running their application).

1. Run one app at a time.
2. Put your name in the name of your application (so you can tell it apart from other students').
3. Kill your application when you have finished with it.
4. Don't interfere with any of the virtual machines in the cluster.
5. Outside of scheduled labs, when the cluster is quieter, you can increase `spark.cores.max`, but keep an eye on other people using the system.
6. Always start your application with dynamic allocation enabled, autoscaling, and a cap on CPU cores (other examples you will see online will not have these settings):

```
from pyspark.sql import SparkSession

spark_session = SparkSession.builder\
        .master("spark://privateipofsparkmaster:7077") \
        .appName("yourname_yourapplicationname")\
        .config("spark.dynamicAllocation.enabled", True)\
        .config("spark.dynamicAllocation.shuffleTracking.enabled",True)\
        .config("spark.shuffle.service.enabled", True)\
        .config("spark.dynamicAllocation.executorIdleTimeout","30s")\
        .config("spark.cores.max", 4)\
        .getOrCreate()
```

There will be technical problems from time to time with the cluster, which we will try to fix as quickly as possible.

In your Notebook, start a spark application by running a cell based on the template code above. Check in the Spark master web GUI for details of your application.

In a new cell, run a toy example with the RDD API (should take a few seconds):

```
def add(a, b):
    # associative and commutative!
    return a + b

rdd = spark_session.sparkContext.parallelize(range(10**7))

result = rdd.filter(lambda x: x % 2 == 0)\
            .map(lambda x: x ** 2)\
            .reduce(add)

print(result)
```

Think: why does our reduce function need to be associative and commutative?

If you setup the port forwarding over SSH, you can use the Web GUI of your driver application (from your laptop) to access a lot of diagnostic information about the running application. Take a good look around!

http://localhost:4040

In a new cell, terminate your application:

```
spark_session.stop()
```

Open the Spark master GUI and confirm the application is terminated.

Learning Outcomes

Learning Outcome	*Assessed*
Use Apache Spark appropriately for interactive analysis of large datasets, using a range of Map/Reduce operations	Section A, B
Deploy Apache Spark to the Cloud	Prerequisite.
Describe key components in the Spark architecture/execution model, use this understanding to develop applications effectively, and avoid common performance pitfalls	Section C, D
Relate key ideas in the implementation of Spark to concepts in the theory of distributed computing and big data.	Section C

Learning Resources

You will need to use the lecture material, Spark documentation and code examples, to answer the questions. Remember the ‘cheat sheet’ handout is intended to help you to navigate the Spark documentation - keep it next to you when you are coding! Unless otherwise stated, you must use Spark for all the calculations (not e.g. Excel or Linux tools!).

Debugging

As we discussed in the lecture, Spark will run our Python code remotely on our cluster, and collect results (or errors!) back to our client application. You can expect error messages to be less helpful than with regular Python.

I recommend you use `my_rdd.take(10)` or `my_dataframe.show()` to check the contents of your RDD at each step. Build up working code by adding one step at a time. Convince yourself of your own understanding before moving on.

Web resources like stackoverflow are useful in that they can help you discover the names of functions (especially where you know what you want but don't know what it's called). If you use snippets from the web, be sure to take the time to fully understand it, and modify it so it does what you need. Use Python 3 for this assignment.

Part A - Working with the RDD API

For this section, we'll use the PySpark RDD API. Use the existing cluster, and a notebook on your own client machine, which you must deploy yourself.

A parallel corpus is a document, or collection of documents, in multiple languages, where sentence boundaries are aligned between those boundaries. We'll work with a parallel corpus of transcripts from the European Parliament http://www.statmt.org/europarl/

Choose a language: I have loaded corpora for all languages into the HDFS cluster, in the directory 'europarl'. Germanic languages like Swedish are likely to work better. For each language there is a pair of aligned files, one in English, and one in the other language.

Question A.1

A.1.1 Read the English transcripts with Spark, and count the number of lines.
A.1.2 Do the same with the other language (so that you have a separate lineage of RDDs for each).
A.1.3 Verify that the line counts are the same for the two languages.
A.1.4 Count the number of partitions.

Question A.2

A.2.1 Pre-process the text from both RDDs by doing the following:

- Lowercase the text
- Tokenize the text (split on space)

Hint: define a function to run in your driver application to avoid writing this code twice.

A.2.2 Inspect 10 entries from each of your RDDs to verify your pre-processing.
A.2.3 Verify that the line counts still match after the pre-processing.

Question 1.A.3

A.3.1 Use Spark to compute the 10 most frequently according words in the English language corpus. Repeat for the other language.
A.3.2 Verify that your results are reasonable.

Question A.4

A.4.1 Use this parallel corpus to mine some translations in the form of word pairs, for the two languages. Do this by pairing words found on short lines with the same number of words respectively. We (incorrectly) assume the words stay in the same order when translated.

Follow this approach. Work with the pair of RDDs you created in question A.2.
Hint: make a new pair of RDDs for each step, sv_1, en_1, sv_2, en_2, ...

1. Key the lines by their line number (hint: ZipWithIndex()).
2. Swap the key and value - so that the line number is the key.
3. Join the two RDDs together according to the line number key, so you have pairs of matching lines.
4. Filter to exclude line pairs that have an empty/missing “corresponding” sentence.
5. Filter to leave only pairs of sentences with a small number of words per sentence, this should give a more reliable translation (you can experiment).
6. Filter to leave only pairs of sentences with the same number of words in each sentence.
7. For each sentence pair, map so that you pair each (in order) word in the two sentences. We no longer need the line numbers. (hint: use python’s built in zip() function)
8. Use reduce to count the number of occurrences of the word-translation-pairs.
9. Print some of the most frequently occurring pairs of words.

Do your translations seem reasonable? Use a dictionary to check a few (don’t worry, you won’t be marked down for incorrect translations!).

Section B - Working with DataFrames and SQL

For this section, we’ll use the PySpark DataFrames/SQL API. Use the existing cluster, and a notebook on your own client machine, which you must deploy yourself.

We’ll work with a large dataset in CSV format. Our dataset is the Los Angeles Parking Citations (https://www.kaggle.com/cityofLA/los-angeles-parking-citations). The dataset has been pre-loaded into the HDFS cluster.

B.1 Load the CSV file from HDFS, and call show() to verify the data is loaded correctly.

B.2 Print the schema for the DataFrame.

B.3 Count the number of rows in the CSV file.

B.4 Count the number of partitions in the underlying RDD.

B.5 Drop the columns VIN, Latitude and Longitude.

B.6 Find the maximum fine amount. How many fines have this amount?
You need to convert the ‘fine amount’ column to a float to do this correctly.

B.7 Show the top 20 most frequent vehicle makes, and their frequencies.

B.8 Let's expand some abbreviations in the color column. Create a *User Defined Function* to create a new column, 'color long', mapping the original colors to their corresponding values in the dictionary below. If there is no key matching the original color, use the original color.

```
COLORS = {
'AL':'Aluminum', 'AM':'Amber', 'BG':'Beige', 'BK':'Black', 'BL':'Blue',
'BN':'Brown', 'BR':'Brown', 'BZ':'Bronze', 'CH':'Charcoal', 'DK':'Dark',
'GD':'Gold', 'GO':'Gold', 'GN':'Green', 'GY':'Gray', 'GT':'Granite',
'IV':'Ivory', 'LT':'Light', 'OL':'Olive', 'OR':'Orange', 'MR':'Maroon',
'PK':'Pink', 'RD':'Red', 'RE':'Red', 'SI':'Silver', 'SL':'Silver',
'SM':'Smoke', 'TN':'Tan', 'VT':'Violet', 'WT':'White', 'WH':'White',
'YL':'Yellow', 'YE':'Yellow', 'UN':'Unknown'
}
```

B.9 Using this new column, what's the most frequent colour value for Toyotas (TOYT)?

Section C - Concepts in Apache Spark and Distributed Computing

Please answer in full sentence(s), per question, as appropriate. Use the handout to help you, while you may also need to search for extra materials. Marks will be awarded based on the correct use of terminology for Spark and distributed computing concepts. Do not need to submit code for these questions, but you might want to experiment with your code to help you answer them.

C.1 What is RDD (Explain the Resilient and Distributed)? How does Spark work with RDD?

C.2 Your colleague is trying to understand her Spark code by adding a print statement inside her split_line(..) function, as shown in this code snippet:

```
def split_line(line):
    print('splitting line...')
    return line.split(' ')

lines = spark_context.textFile("hdfs://host:9000/i-have-a-dream.txt")
print(lines.flatMap(split_line).take(10))
```

When she runs this code in her notebook, she sees the following output:

```
['I', 'am', 'happy', 'to', 'join', 'with', 'you', 'today', 'in', 'what']
```

But, she doesn't see the "splitting line…" output in her notebook. Why not?

C.3 Calling .collect() on a large dataset may cause driver application to run out of memory
Explain why.

C.4 Are partitions mutable or immutable? Why is this advantageous?

C.5 What is the difference between DataFrame and RDD? Explain their advantages/shortages

Section D - Essay Questions

“A colleague has mentioned her Spark application has poor performance, what is your advice?”

List 4 clear recommendations, answer in full sentences. I suggest 2 or 3 sentences for each recommendation.

Marks will be awarded based on the correct use of Spark terminology. Do not submit code for this question (but you might want to mention API methods, for example). You do not need to include references, but if you feel this supports your answer, please include a short, well-formatted bibliography.