UNIVERSITY OF TARTU
Institute of Computer Science

Cloud Computing 2022/23 spring

Practice 7 - Introduction to Apache Spark

In this Practice session, you will start working with the Apache Spark framework in Python. You will learn how to set it up on your Laptop without having to install or configure Spark and learn how to modify Spark RDD applications using the PyCharm Python IDE. We will improve how the WordCount application splits lines of text into words and will change how the data is grouped between Map and Reduce tasks.

References

- Spark programming guide: https://spark.apache.org/docs/latest/programming-guide.html
- Spark Python API - https://spark.apache.org/docs/latest/api/python/index.html
- Spark Python RDD functions: https://spark.apache.org/docs/latest/api/python/reference/pyspark.html#rdd-apis
- Spark Python examples: https://github.com/apache/spark/tree/master/examples/src/main/python

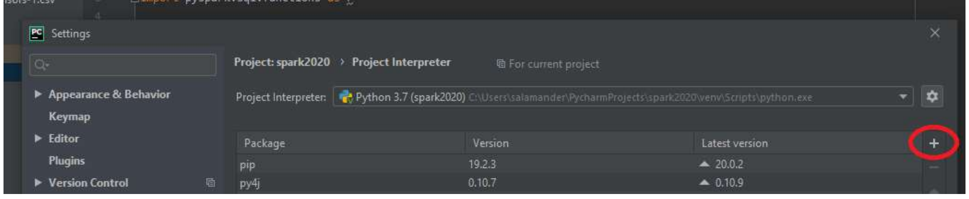
Exercise 7.1. Configuring PyCharm IDE for Spark Python

We will use a Python PyCharm IDE to simplify working with Spark Python RDD scripts.

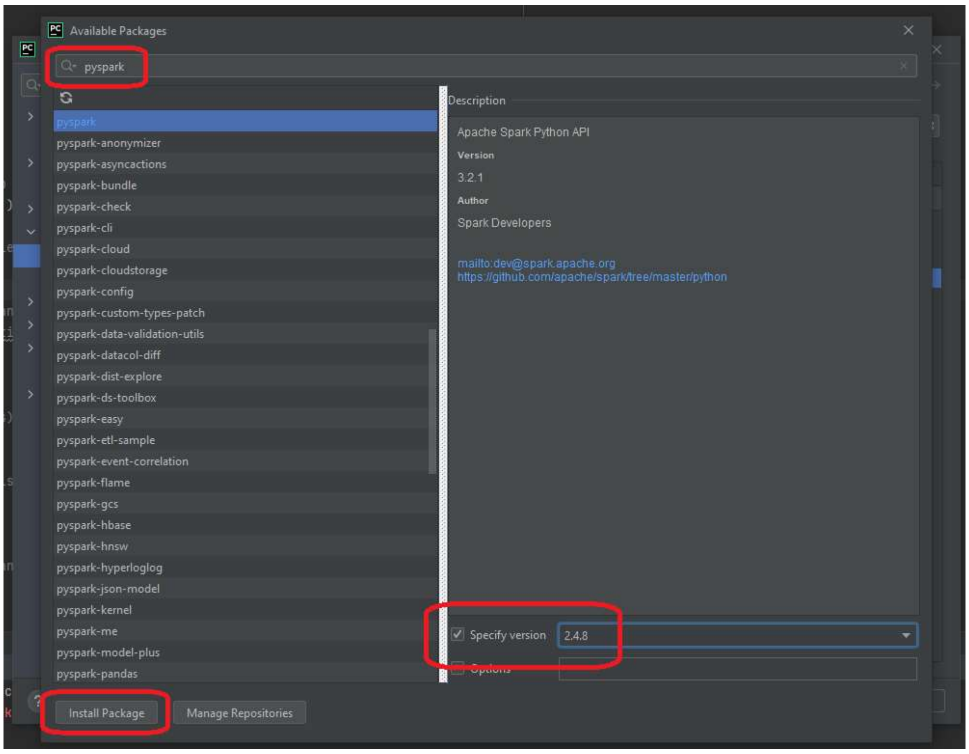
It is also ok to use other approaches to run Spark Python scripts. You could also follow the command line and virtualenv approach if PyCharm does not work nicely in your computer.

- NB! Make sure your computer has Python installed.
  - In Linux and Mac:
    - 3.7, 3.8, 3.9, 3.10 should all work
  - In Windows:
    - Suggested Python version: 3.7
      - NB! Do not use Python 3.8!
    - NB! You will also need Java version 8 (v 1.8)
  - If you need help setting up JAVA v8 into Windows system path: https://www.geeksforgeeks.org/how-to-set-java-path-in-windows-and-linux/
  - It would be best to check that JAVA_HOME is not set (better option) up or refers to Java 8 path.

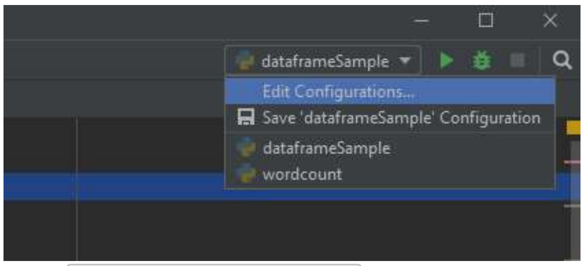
- Download and install the **community edition** of PyCharm Python IDE.
  - https://www.jetbrains.com/pycharm/download
  - You can also use the Professional version, you will be able to use your university email to activate the Student subscription.
- Open PyCharm and create a new Python project.
  - Create a new VirtualEnv.
    - NB! Make sure the Python version is the same as the default Python of your computer (Either 2.* or 3.*)
- After creating a new project, we need to import required libraries
  - Open the PyCharm settings (`File->Settings`)
  - Open Project Interpreter settings `Project: Name -> Project Interpreter`
  - Click the Plus button on the upper right side of the interpreter settings window


Settings
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Project: spark2020 › Project Interpreter
For current project
Project Interpreter: Python 3.7 (spark2020) C:\Users\salamander\PycharmProjects\spark2020\venv\Scripts\python.exe
Package
Version
Latest version
pip 19.2.3 20.0.2
py4j 0.10.7 0.10.9

- Write the name of the python package you want to install
  - The package we want to install are: `pyspark`
  - In Windows:
    - Make sure to install Spark version 2.4.8!!
  - In Linux and Mac:
    - Can install latest Spark version 3.X
- Click on the **Install Package** button on the bottom left side of the window.



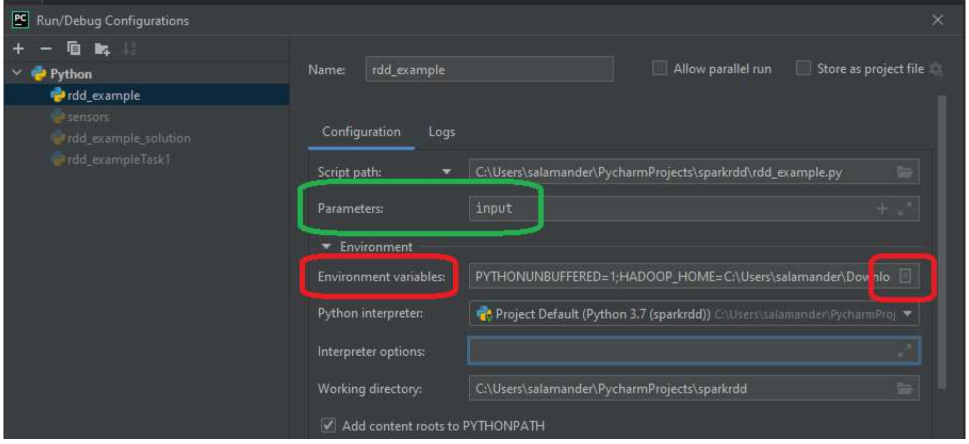
- Download the following Python Spark WordCount example [spark_rdd_example.py]() file and move it inside your PySpark project.
- Create a new folder named **input** inside your project. We will put all the input files for the WordCount application there.
  - Download 5 random books from Gutenberg in **Plain Text (UTF-8)** format:
    - [http://www.gutenberg.org/ebooks/search/?sort_order=random]()
  - Move the downloaded text files into the input folder.
- Try to run the spark_rdd_example.py (You will get an error about the missing arguments, but it will generate run configuration for it)
- Modify the run configuration of the spark_rdd_example.py script.



- The example script requires one input parameter: `location of the input folder`
  - Specify the name of the previously created input folder under program `Parameters` of the Run configuration of the example script.

Additional tasks if you are using Windows (Otherwise skip this section)

- Create a new empty folder named `hadoop` somewhere in your computer and a subfolder `bin` inside the hadoop folder.
- Download windows pre-built Hadoop utilities which are required for running Hadoop in Windows and are used also by Apache Spark for filesystem-related operations.
  - Newer Hadoop libraries require additional native Windows libraries to be built which are not distributed with the Hadoop binaries by default. It is possible to build them from the Hadoop source code, but we will download pre-built versions instead to save time.
  - Download `hadoop-2.8.1.zip` from [GitHub repository]()
  - Unpack the container, scan it with anti virus and copy its content (only files) into the `hadoop/bin` folder we downloaded in the previous step of this task.
- If you get an error about `MSVCR100.dll` then you may have to download and install **Microsoft Visual C++ 2010 Redistributable Package** (It should be 64-bit version if you're using 64bit OS)
- Set up the HADOOP_HOME environment variable inside the Run Configuration. It should link to the Hadoop folder you previously created downloaded!



Run/Debug Configurations

- Python
  - rdd_example
  - sensors
  - rdd_example_solution
  - rdd_exampleTask1

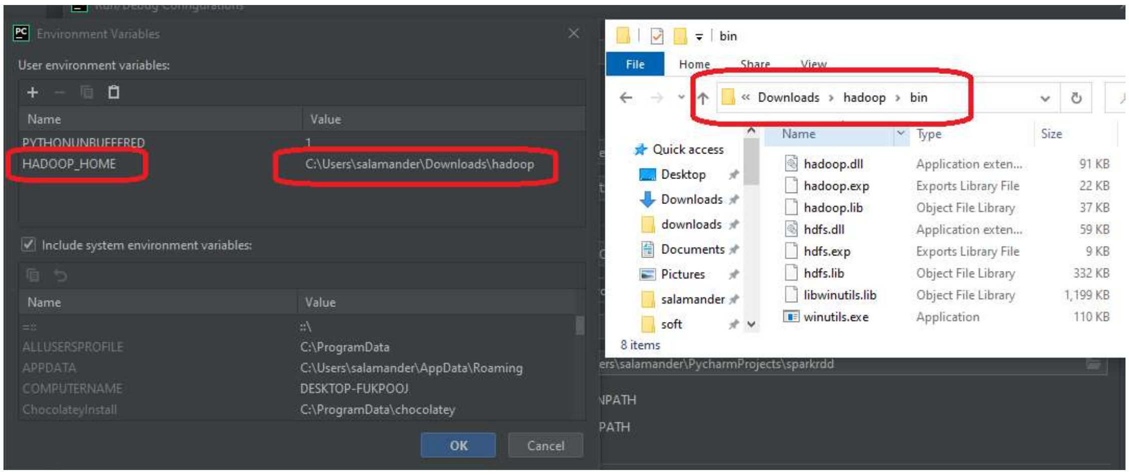
Name: rdd_example   ☐ Allow parallel run   ☐ Store as project file

Configuration | Logs

Script path:	C:\Users\salamander\PycharmProjects\sparkrdd\rdd_example.py
Parameters:	input
Environment variables:	PYTHONUNBUFFERED=1;HADOOP_HOME=C:\Users\salamander\Downlo
Python interpreter:	Project Default (Python 3.7 (sparkrdd)) C:\Users\salamander\PycharmProj
Interpreter options:	
Working directory:	C:\Users\salamander\PycharmProjects\sparkrdd

☑ Add content roots to PYTHONPATH

- This Hadoop folder must include the winutils.exe and other files we copied inside the bin subfolder! NB! HADOOP_HOME must refer to hadoop folder, NOT the bin subfolder inside the hadoop folder!



Environment Variables

User environment variables:

Name	Value
PYTHONUNBUFFERED	1
HADOOP_HOME	C:\Users\salamander\Downloads\hadoop

☑ Include system environment variables:

Name	Value
=::	::\
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\salamander\AppData\Roaming
COMPUTERNAME	DESKTOP-FUKPOOJ
ChocolateyInstall	C:\ProgramData\chocolatey

OK Cancel

bin

« Downloads › hadoop › bin

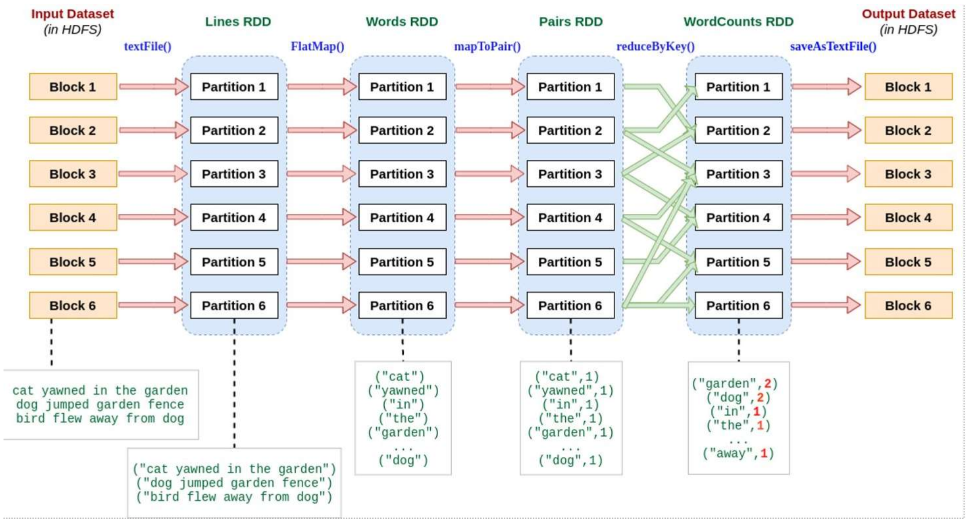
Name	Type	Size
hadoop.dll	Application exten...	91 KB
hadoop.exp	Exports Library File	22 KB
hadoop.lib	Object File Library	37 KB
hdfs.dll	Application exten...	59 KB
hdfs.exp	Exports Library File	9 KB
hdfs.lib	Object File Library	332 KB
libwinutils.lib	Object File Library	1,199 KB
winutils.exe	Application	110 KB

8 items

Exercise 7.2. Learning the Python Spark WordCount example

- Familiarize yourself with the Python Spark WordCount example, the script contains the Word Count example from the lecture and comments that explain what each of the steps of code are doing.
- Run the Python script.
  - This example prints the results out directly to the console

Illustration of how the Spark WordCount example works from the lecture:



Exercise 7.3. Improving the example Word Count application

One of the main problems with the simple MapReduce WordCount example is that it does not remove punctuation marks and other special characters and as a result, it counts words like `cat` and `cat.` separately.

1. Improve how the text lines are split into words.
   - Replace the lambda function `lambda line: line.split(' ')` applied inside the flatMap RDD operation by a custom function, which removes punctuation marks (.,:;# etc.) and it should also output only clear lowercase words.
   - NB! If you create your own custom function, remember that function applied inside the flatMap operation needs to return a list of values (words in our case).
   - Output is expected to be the same as in was in the example WordCount, the only difference is that the output should no longer have capital letters and words should not start or end with punctuation marks.

Deliverable: save the current script as the solution of the exercise. Also save a copy of the script output as a text file (not screenshot).

- The output records should contain: `word, count`
- For example:

```
are 763
page 19
symbol 1
replaced 6
chapter 5
```

Exercise 7.4. Extending the Word Count application

In the basic Word Count example, all unique words were grouped together and their Sum was computed. In this exercise, we will change the WordCount to calculate the count of words for each input file (book) separately by changing how the data is grouped. Then, in the next exercise, we will modify the application to only output top 5 words (based on the highest count) for each input file.

1. Extract the name of the file for each input record.
   - To include the file path inside the RDD, we need to change how the input RDDs are created.
   - Let's replace the line:
     - `lines = sc.textFile(input_folder)`
   - with
     - `lines = spark.read.text(input_folder).select(input_file_name(), "value").rdd.map(tuple)`
   - This will use the `input_file_name()` function to generate a new column into the RDD, where the value is the file name (of the input).
   - You will also need to import the `input_file_name()` function: `from pyspark.sql.functions import input_file_name`
   - Now the resulting RDD contains tuples, which have 2 elements: `(file_path, line_from_the_input_files)`
   - However, now the previous code no longer works properly as the structure inside the RDD's have changed.
2. Let's fix the WordCount code to work with expanded tuple.
   - The main thing we need to do is to change how the lines are split into words, and what is the key that the data is grouped by.
     - Previously, data was only grouped by the `word` value. Now we need to group data by both `word` and `file_name`.
   - Fixing the splitting stage:
     - Now that the RDD contains filename and line columns, we want to change the code in a way that the line is split into word and file_name is not affected.
     - Something to keep in mind, when dealing Spark RDD's that contain tuples of two elements:
       - In Spark, first value of the tuple is considered a key
       - Second value is considered a value.
       - If there are more than 2 elements in the RDD, all are considered values.
     - Because the linesRDD contains file path as key, we can use the `flatMapValues` operation instead of `flatMap` operation to apply the splitting operation only on the second element of the tuple: the line value.
       - Replace the `flatMap` with `flatMapValues` operation
       - This should mean that you do not need to modify how the splitting is applied. (Other than changing the Spark flatMap operation)
   - Fixing the map stage:
     - We should also update the next operation: `pairs = words.map(lambda word: (word, 1))`
       - Instead of word, the input to the map is now tuple of `(file_name, word)`.
       - We want the resulting structure to be `((file_name, word), 1)`, where
         - `(file_name, word)` is the **key** - meaning data is grouped by both file name and word.
         - `1` is the **value** that will be summed together
       - However, we do not actually need to change the lambda function, as it will simply move the existing tuple (which now consists of `(file_name, word)` instead of `(word)`) into key and assigns 1 as a value - creating a nested tuple. And creates the same result we wanted as a result.
   - The rest of the code can also stay the same.

- The output records should contain: `(file_name, word), count`
- For example

```
('43774-0.txt', 'images'): 3
('43774-0.txt', 'generously'): 2
('43774-0.txt', 'inconsistent'): 1
('43774-0.txt', 'spelling'): 8
('43774-0.txt', 'document'): 16
```

Exercise 7.5. Replacing the count with top 5

Let's now extend the previous solution to compute and output only top 5 words (with the highest count) for each file.

1. Extend the previous WordCount application to compute the top 5 word with the highest count for each book.
   - We can start from the state when we already have computed count for each unique file and word.
   - Add a new Spark `map()` operation to restructure the RDD containing `((file_name, word), count)` into: `(file_name, (word, count))` - leaving the filename as the key and a tuple of file_name and word as a value.
     - You can either use a lambda function or define a custom python function to be used inside the map operation.
     - When using a lambda functions that get tuples as input, You can address tuple elements just like list elements.
       - Example: `tuple[0]` - to get first element of a tuple.
       - Example, when dealing with nested tuples: `lambda tuple: tuple[0][1] + tuple[1]`
   - Use the `groupByKey()` operation on the output RDD of the previous step to create a new RDD which has been grouped by the file name.
   - Apply `mapValues(top5)` Spark operation on the grouped RDD to apply a user defined function `top5(records)` on each group (unique file_name).

    - Implement the `top5(records)` function, which gets an iterable list of `(word, count)` tuples (`records`) as input and outputs a list of `(word, count)` tuples, where the count is highest
        - Effectively, the task is to order the tuples by the count values, and output a list of 5 tuples with the highest count values.
        - Hint: Some of the Python sorting features allow sorting a list of tuples based on specific tuple column index
2. Save the resulting RDD into a text file in plain-text format using `saveAsTextFile` Spark action.
    - This will likely result in multiple text files, one for each of the parallel Spark processes/threads working in the background. It may also happen that few of the files are empty, but there should be one record for each of the input files total inside the output files.

Deliverable: save the current script as the solution of the exercise. Also, the text file(s) of the saved RDD will be needed later in the delievrables.

- The output records should contain: `file, [(word, count)]`
- For example

```
('pg31806.txt', [ ('the', 2679), ('and', 1497), ('to', 1351), ('of', 1022)])
('pg32067.txt', [('the', 611), ('to', 261), ('of', 247), ('and', 223)])
```

Bonus exercise: Extending the Word Count application to compute the frequency of Words in the documents

The goal of thee bonus tasks it to extend the WordCount application to compute the frequency of all words in each of the files (book) in the input folder.

- Frequency of a word `w` in a file `f`: how frequently the word w occurs in the file `f` in comparison to the total number of words in file `f`.
    - Example: word `cat` occurs in file "`file1.txt` 100 times. `file1.txt` contains 40000 words. The frequency of the word `cat"` in file `file1.txt` is 1/400=0.0025.
- All the computations must be done "inside" the Spark RDD API.
    - Hint: It is possible to compute different results as separate RDD's and join two RDD's into a single one using the Spark RDD join operations.
    - You are not allowed to use the Spark DataFrame API (we will take a look at that in the next lab)

Deliverables:

- Python script from Exercise 7.3
- Python script from Exercise 7.5 (only the final version, no need to save intermediate versions of the script separately)
- Output of the Python scripts from Exercises 7.3, 7.5

Task	Lab 7 - spark
Current submission	ZIP (17:24 24.03.2023) If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
File	Choose File No file chosen
Comment	

Submit

Potential issues and solutions

- **NB!**You are experiencing "java.lang.UnsatisfiedLinkError: org.apache.hadoop.io.nativeio.NativeIO$Windows.access(.....)":
    - Please try one of these two:
        1. Try downgrading to Spark 2.4.8 in your Python Project
            - Open project settings, where you added "pyspark", and choose "Specify Version" after opening the pyspark package details
        2. Use OS Environment variables instead of PyCharm run configuration of HADOOP_HOME. Since we will use set them at OS level, remove these from PyCharm run conf.
            - Set a new Windows Environment Variable HADOOP_HOME with the correct directory path for hadoop.
            - After the above steps, restart PyCharm and try running the project again
        3. Check Java v8 is set up properly.
            - Check that JAVA_HOME is not set up (**ok**) or refers to Java 8 path.
                - It is best to avoid spaces in JAVA_HOME path.
                    - You can replace `C:\Program Files\` with `C:\PROGRA~1\` and `C:\Program Files (x86)\` with `C:\PROGRA~2\` in the java path
            - Run the command `java -version` in Pycharm terminal. And check that you have the correct version.
                - If you get an error that java command is not found, then you have to set up Windows path environment variable. Guide for setting up JAVA into Windows system path: https://www.geeksforgeeks.org/how-to-set-java-path-in-windows-and-linux/
            - Append `/bin;` to your Windows PATH variable
- NB! You may see errors like:
    - `"ERROR:root:Exception while sending the command."`

    - `"ConnectionRefusedError: [WinError 10061] No connection could be made because the target machine actively refused it"`
    - These errors are not related to the Spark code but seem to be an issue about properly closing the Spark session.
    - You can ignore these errors as long as all the output is shown properly in the output.
- Be careful with the code indentation in Python script. Python is very strict when it comes to mixing tabs and spaces.
    - Check indentation manual for Python: https://docs.python.org/2.0/ref/indentation.html
- It is suggested to use Java 8 as the default Java in your computer.
- Be careful with the code indentation in Python script. Python is very strict when it comes to mixing tabs and spaces.
    - Check indentation manual for Python: https://docs.python.org/2.0/ref/indentation.html
- **Python in worker has different version 2.7 then that in driver 3.6**
    - Check that Your Pycharm project Python interpreter matches your computer's default interpreter. Spark will launch python worker processes in the background and there will be a conflict if worker processes use different Python versions.
- In Windows, avoid spaces in user and folder names.