Big Data Fundamentals with PySpark (Programming in PySpark RDD’s)

RDDs from Parallelized collections

Resilient Distributed Dataset (RDD) is the basic abstraction in Spark. It is an immutable distributed collection of objects. Since RDD is a fundamental and backbone data type in Spark, it is important that you understand how to create it. In this exercise, you'll create your first RDD in PySpark from a collection of words.

Remember you already have a SparkContext `sc` available in your workspace.

```python
# Create an RDD from a list of words
RDD = sc.parallelize(["Spark", "is", "a", "framework", "for", "Big Data processing"])

# Print out the type of the created object
print("The type of RDD is", type(RDD))
```

RDDs from External Datasets

PySpark can easily create RDDs from files that are stored in external storage devices such as HDFS (Hadoop Distributed File System), Amazon S3 buckets, etc. However, the most common method of creating RDD's is from files stored in your local file system. This method takes a file path and reads it as a collection of lines. In this exercise, you'll create an RDD from the file path (`file_path`) with the file name `README.md` which is already available in your workspace.

Remember you already have a SparkContext `sc` available in your workspace.

```python
# Print the file_path
print("The file_path is", file_path)

# Create a fileRDD from file_path
fileRDD = sc.textFile(file_path)

# Check the type of fileRDD
print("The file type of fileRDD is", type(fileRDD))
```

Partitions in your data

SparkContext's `textFile()` method takes an optional second argument called `minPartitions` for specifying the minimum number of partitions. In this exercise, you'll create an RDD named `fileRDD_part` with 5 partitions and then compare that with `fileRDD` that you created in the previous exercise. Refer to the "Understanding Partition" slide in video 2.1 to know the methods for creating and getting the number of partitions in an RDD.

Remember, you already have a SparkContext `sc`, `file_path` and `fileRDD` available in your workspace.

```python
# Check the number of partitions in fileRDD
print("Number of partitions in fileRDD is", fileRDD.getNumPartitions())

# Create a fileRDD_part from file_path with 5 partitions
fileRDD_part = sc.textFile(file_path, minPartitions = 5)

# Check the number of partitions in fileRDD_part
print("Number of partitions in fileRDD_part is", fileRDD_part.getNumPartitions())
```

Map and Collect

The main method by which you can manipulate data in PySpark is using `map()`. The `map()` transformation takes in a function and applies it to each element in the RDD. It can be used to do any number of things, from fetching the website associated with each URL in our collection to just squaring the numbers. In this simple exercise, you'll use `map()` transformation to cube each number of the `numbRDD` RDD that you created earlier. Next, you'll return all the elements to a variable and finally print the output.

Remember, you already have a SparkContext `sc`, and `numbRDD` available in your workspace.

```python
# Create map() transformation to cube numbers
cubedRDD = numbRDD.map(lambda x: x**3)

# Collect the results
numbers_all = cubedRDD.collect()

# Print the numbers from numbers_all
for numb in numbers_all:
    print(numb)
```

Filter and Count

The RDD transformation `filter()` returns a new RDD containing only the elements that satisfy a particular function. It is useful for filtering large datasets based on a keyword. For this exercise, you'll filter out lines containing keyword `Spark` from `fileRDD` RDD which consists of lines of text from the `README.md` file. Next, you'll count the total number of lines containing the keyword `Spark` and finally print the first 4 lines of the filtered RDD.

Remember, you already have a SparkContext `sc`, `file_path` and `fileRDD` available in your workspace.

```python
# Filter the fileRDD to select lines with Spark keyword
fileRDD_filter = fileRDD.filter(lambda line: 'Spark' in line)

# How many lines are there in fileRDD?
print("The total number of lines with the keyword Spark is", fileRDD_filter.count())

# Print the first four lines of fileRDD
for line in fileRDD_filter.take(4):
  print(line)
```

ReduceByKey and Collect

One of the most popular pair RDD transformations is `reduceByKey()` which operates on key, value (k,v) pairs and merges the values for each key. In this exercise, you'll first create a pair RDD from a list of tuples, then combine the values with the same key and finally print out the result.

Remember, you already have a SparkContext `sc` available in your workspace.

```python
# Create PairRDD Rdd with key value pairs
Rdd = sc.parallelize([(1,2), (3, 4), (3, 6), (4, 5)])

# Apply reduceByKey() operation on Rdd
Rdd_Reduced = Rdd.reduceByKey(lambda x, y: x + y)

# Iterate over the result and print the output
for num in Rdd_Reduced.collect():
  print("Key {} has {} Counts".format(num[0], num[1]))
```

SortByKey and Collect

Many times it is useful to sort the pair RDD based on the key (for example word count which you'll see later in the chapter). In this exercise, you'll sort the pair RDD `Rdd_Reduced` that you created in the previous exercise into descending order and print the final output.

Remember, you already have a SparkContext `sc` and `Rdd_Reduced` available in your workspace.

```python
# Sort the reduced RDD with the key by descending order
Rdd_Reduced_Sort = Rdd_Reduced.sortByKey(ascending=False)

# Iterate over the result and print the output
for num in Rdd_Reduced_Sort.collect():
  print("Key {} has {} Counts".format(num[0], num[1]))
```

Counting By Keys

For many datasets, it is important to count the number of keys in a key/value dataset. For example, counting the number of countries where the product was sold or to show the most popular baby names. In this simple exercise, you'll use the `Rdd` pair RDD that you created earlier and count the number of unique keys in that pair RDD.

Remember, you already have a SparkContext `sc` and `Rdd` available in your workspace.

```python
# Transform the rdd with countByKey()
total = Rdd.countByKey()

# What is the type of total?
print("The type of total is", type(total))

# Iterate over the total and print the output
for k, v in total.items():
  print("key", k, "has", v, "counts")
```

Create a base RDD and transform it

The volume of unstructured data (log lines, images, binary files) in existence is growing dramatically, and PySpark is an excellent framework for analyzing this type of data through RDDs. In this 3 part exercise, you will write code that calculates the most common words from **Complete Works of William Shakespeare**.

Here are the brief steps for writing the word counting program:

- Create a base RDD from `Complete_Shakespeare.txt` file.
- Use RDD transformation to create a long list of words from each element of the base RDD.
- Remove stop words from your data.
- Create pair RDD where each element is a pair tuple of `('w', 1)`
- Group the elements of the pair RDD by key (word) and add up their values.
- Swap the keys (word) and values (counts) so that keys is count and value is the word.

- Finally, sort the RDD by descending order and print the 10 most frequent words and their frequencies.

In this first exercise, you'll create a base RDD from `Complete_Shakespeare.txt` file and transform it to create a long list of words.

Remember, you already have a SparkContext `sc` already available in your workspace. A `file_path` variable (which is the path to the `Complete_Shakespeare.txt` file) is also loaded for you.

```python
# Create a baseRDD from the file path
baseRDD = sc.textFile(file_path)

# Split the lines of baseRDD into words
splitRDD = baseRDD.flatMap(lambda x: x.split())

# Count the total number of words
print("Total number of words in splitRDD:", splitRDD.count())
```

Remove stop words and reduce the dataset

After splitting the lines in the file into a long list of words using `flatMap()` transformation, in the next step, you'll remove stop words from your data. Stop words are common words that are often uninteresting. For example "I", "the", "a" etc., are stop words. You can remove many obvious stop words with a list of your own. But for this exercise, you will just remove the stop words from a curated list `stop_words` provided to you in your environment.

After removing stop words, you'll next create a pair RDD where each element is a pair tuple `(k, v)` where `k` is the key and `v` is the value. In this example, pair RDD is composed of `(w, 1)` where `w` is for each word in the RDD and 1 is a number. Finally, you'll combine the values with the same key from the pair RDD using `reduceByKey()` operation

Remember you already have a SparkContext `sc` and `splitRDD` available in your workspace.

```python
# Convert the words in lower case and remove stop words from stop_words
splitRDD_no_stop = splitRDD.filter(lambda x: x.lower() not in stop_words)

# Create a tuple of the word and 1
splitRDD_no_stop_words = splitRDD_no_stop.map(lambda w: (w, 1))

# Count of the number of occurences of each word
resultRDD = splitRDD_no_stop_words.reduceByKey(lambda x, y: x + y)
```

Print word frequencies

After combining the values (counts) with the same key (word), you'll print the word frequencies using the `take(N)` action. You could have used the `collect()` action but as a best practice, it is not recommended as `collect()` returns all the elements from your RDD. You'll use `take(N)` instead, to return `N` elements from your RDD.

What if we want to return the top 10 words? For this first, you'll need to swap the key (word) and values (counts) so that keys is count and value is the word. After you swap the key and value in the tuple, you'll sort the pair RDD based on the key (count) and print the top 10 words in descending order.

You already have a SparkContext `sc` and `resultRDD` available in your workspace.

```python
# Display the first 10 words and their frequencies
for word in resultRDD.take(10):
    print(word)

# Swap the keys and values
resultRDD_swap = resultRDD.map(lambda x: (x[1], x[0]))

# Sort the keys in descending order
resultRDD_swap_sort = resultRDD_swap.sortByKey(ascending=False)

# Show the top 10 most frequent words and their frequencies
for word in resultRDD_swap_sort.take(10):
    print("{} has {} counts". format(word[1], word[0]))
```