EECS E6893 Big Data Analytics
Spark Introduction Tutorial

Apurva Patel, amp2365@columbia.edu

Agenda

- Functional programming in Python
  - Lambda
- Crash course in Spark (PySpark)
  - RDD
  - Useful RDD operations
    - Actions
    - Transformations
  - Example: Word count

Functional programming in Python

Lambda expression

- Creating small, one-time, anonymous function objects in Python
- Syntax: `lambda arguments: expression`
  - Any number of arguments
  - Single expression
- Could be used together with *map, filter, reduce*
- Example:
  - Add:

```
add = lambda x, y : x + y
```

Represents same thing

```
def add (x, y):
    return x + y
```

```
type(add) = <type 'function'>
```

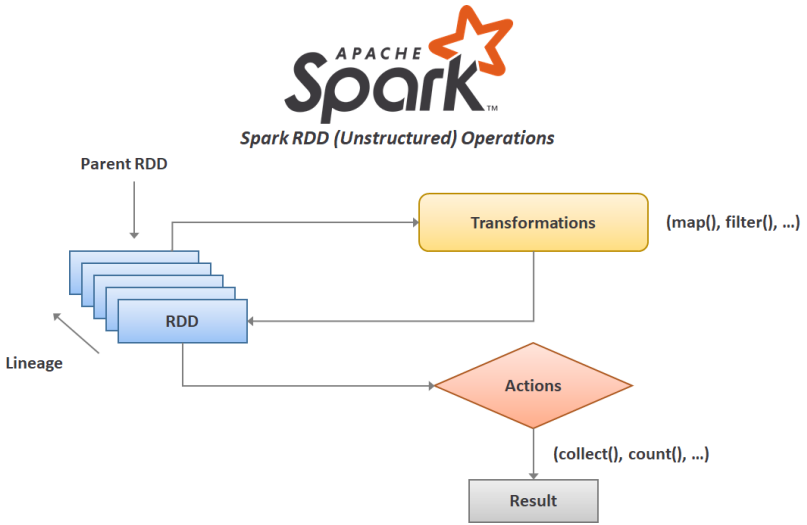
```
add(2,3)
```

Crash course in Spark

Resilient Distributed Datasets (RDD)

Main abstraction provided by Spark

- a collection of elements
- partitioned across the nodes of the cluster
- can be operated on in parallel



Transformations - creating a new RDD dataset on top of already existing one with the last transformation (returns a new RDD representing the results)

Actions -return a value to the driver program after running a computation on the dataset (return final result)

Refer this for more explanation: https://tomaztsql.wordpress.com/2021/12/08/advent-of-2021-day-8-creating-rdd-files/

Resilient Distributed Datasets (RDD)

- Spark is RDD-centric
- RDDs are immutable
- RDDs can be cached in memory
- RDDs are computed lazily (Time complex)
- RDDs know who their parents are (knows the source of inheritence)
- RDDs automatically recover from failures (might require some permissions)

Useful RDD Actions

- take(n): return the first n elements in the RDD as an array.
- collect(): return all elements of the RDD as an array. Use with caution.
- count(): return the number of elements in the RDD as an int.
- saveAsTextFile(‘path/to/dir’): save the RDD to files in a directory. Will create the directory if it doesn’t exist and will fail if it does.
- foreach(func): execute the function against every element in the RDD, but don’t keep any results.

Note: *func* is a lambda expression: due to this feature we cannot define a function in spark and have to use lambda function

Useful RDD transformations

map(*func*)

- Apply a function to every element of an RDD and return a new result RDD

```
data = ["Apple,Amy", "Butter,Bob", "Cheese,Chucky"]
data = sc.parallelize(data)
```

```
# map
data.map(lambda line: line.split(',')).take(3)
```

```
[['Apple', 'Amy'], ['Butter', 'Bob'], ['Cheese', 'Chucky']]
```

```
data.map(lambda line: line.lower()).take(3)
```

```
['apple,amy', 'butter,bob', 'cheese,chucky']
```

Note for map()

- Applies a given function to each element of a collection.
- Returns a new collection with the same number of elements as the original.
- Each element in the result is the direct output of the mapping function.

flatmap(*func*)

- Similar to *map()*, yet flatten by removing the outermost container

```
# flatMap
data.flatMap(lambda line: line.split(',')).take(6)
```

```
['Apple', 'Amy', 'Butter', 'Bob', 'Cheese', 'Chucky']
```

Note for flatMap():

- Also applies a given function to each element of a collection.
- The mapping function returns a collection for each element.
- flatMap() then "flattens" these resulting collections into a single collection.
- The final result may have a different number of elements than the original.

mapValues(*func*)

- Apply an operation to the value of every element of an RDD and return a new result RDD
- **Only works with pair RDDs (key,value)**

```
pair_data = [('Apple', 'Amy'), ('Butter', 'Bob'), ('Cheese', 'Chucky')]
pair_data = sc.parallelize(pair_data)
```

```
# mapValues()
# each pair: (key, value)
pair_data.mapValues(lambda name: name.lower()).take(3)
```

```
[('Apple', 'amy'), ('Butter', 'bob'), ('Cheese', 'chucky')]
```

flatMapValues(*func*)

- Pass each value in the (K, V) pair RDD through a *flatMap* function without changing the keys

```
# flatMapValues()
pair_data.flatMapValues(lambda name: name.lower()).take(6)
```

```
[('Apple', 'a'),
 ('Apple', 'm'),
 ('Apple', 'y'),
 ('Butter', 'b'),
 ('Butter', 'o'),
 ('Butter', 'b')]
```

filter(*func*)

- Return a new RDD by selecting the elements which func returns true

```
# filter
data = sc.parallelize([1, 2, 3, 4, 5])
data.filter(lambda x: x % 2 != 0).take(3)
```

```
[1, 3, 5]
```

groupByKey()

- When called on a RDD of (K, V) pairs, returns a new RDD of (K, Iterable<V>) pairs

```
# groupByKey()
data = sc.parallelize([('A', 1), ('A', 2), ('B', 3), ('C', 4)])
print(data.groupByKey().take(1))

for pair in data.groupByKey().take(1):
    print(pair[0], [n for n in pair[1]])
```

```
[('A', <pyspark.resultiterable.ResultIterable object at 0x7f0b00a85290>)]
('A', [1, 2])
```

reduceByKey(*func*)

- Combine elements of an RDD by key and then apply a *reduce func* to pairs of values until only a single value remains
- reduce function *func* must be of type (V,V) => V

```
# reduceByKey()
data = sc.parallelize([('A', 1), ('A', 2), ('B', 3), ('C', 4)])
data.reduceByKey(lambda v1, v2: v1 + v2).take(1)
```

```
[('A', 3)]
```

sortBy(*func*)

- Sort an RDD according to a sorting *func* and return the results in a new RDD

```
# sortBy()
data = sc.parallelize([('A', 99), ('B', 3), ('C', 4)])

print(data.sortBy(lambda pair: pair[1]).take(4))
print(data.sortBy(lambda pair: -pair[1]).take(4))
print(data.sortBy(lambda pair: pair[0]).take(4))
```

```
[('B', 3), ('C', 4), ('A', 99)]
[('A', 99), ('C', 4), ('B', 3)]
[('A', 99), ('B', 3), ('C', 4)]
```

sortBy(*func*)

1. Data creation:
data = sc.parallelize([('A', 99), ('B', 3), ('C', 4)])
This creates a Resilient Distributed Dataset (RDD) with three tuples, each containing a letter and a number.

2. Sorting operations:
The code then performs three different sorting operations on this data:

a. `data.sortBy(lambda pair: pair[1]).take(4)`
- Sorts the data based on the second element of each tuple (the number).
- Results in `[('B', 3), ('C', 4), ('A', 99)]` (ascending order of numbers).

b. `data.sortBy(lambda pair: -pair[1]).take(4)`
- Sorts based on the negative of the second element, effectively reversing the order.
- Results in `[('A', 99), ('C', 4), ('B', 3)]` (descending order of numbers).

c. `data.sortBy(lambda pair: pair[0]).take(4)`
- Sorts based on the first element of each tuple (the letter).
- Results in `[('A', 99), ('B', 3), ('C', 4)]` (alphabetical order of letters).

3. The `take(4)` method:
- This returns the first 4 elements of the sorted RDD. In this case, since there are only 3 elements, it returns all of them.

4. Lambda functions:
- These are used to specify the sorting key. `pair[1]` refers to the number, `pair[0]` to the letter.

sortByKey()

- Sort an RDD according to the ordering of the keys and return the results in a new RDD.

```
# sortByKey()
data = sc.parallelize([('A', 99), ('B', 3), ('C', 4)])
data.sortByKey().take(3)
```

```
[('A', 99), ('B', 3), ('C', 4)]
```

substract()

- Return a new RDD that contains all the elements from the original RDD that do not appear in a target RDD.

```
# substract
data1 = sc.parallelize(['Apple,Amy', 'Butter,Bob', 'Cheese,Chucky'])
data2 = sc.parallelize(['Wendy', 'McDonald,Ronald', 'Cheese,Chucky'])
data1.subtract(data2).take(3)
```

```
['Butter,Bob', 'Apple,Amy']
```

Example: word count in Spark

```
import pyspark
import sys

if len(sys.argv) != 3:
  raise Exception("Exactly 2 arguments are required: <inputUri> <outputUri>")

inputUri=sys.argv[1]
outputUri=sys.argv[2]

sc = pyspark.SparkContext()
lines = sc.textFile(sys.argv[1])
words = lines.flatMap(lambda line: line.split())
wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda count1,
count2: count1 + count2)
wordCounts.saveAsTextFile(sys.argv[2])
```

https://cloud.google.com/dataproc/docs/tutorials/gcs-connector-spark-tutorial#python

Word count in Spark:create RDD & read file into RDD (1)

Create RDD

```
sc = pyspark.SparkContext()
```

represents the connection to a Spark cluster

```
!pip install nltk==3.6.5
!pip install regex==2021.10.8
```

```
Requirement already satisfied: nltk==3.6.5 in /opt/conda/miniconda3/lib/python3.8/site-packages (3.6.5)
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/miniconda3/lib/python3.8/site-packages (from nltk==3.6.5) (2021.10.8)
Requirement already satisfied: tqdm in /opt/conda/miniconda3/lib/python3.8/site-packages (from nltk==3.6.5) (4.64.0)
Requirement already satisfied: joblib in /opt/conda/miniconda3/lib/python3.8/site-packages (from nltk==3.6.5) (1.1.0)
Requirement already satisfied: click in /opt/conda/miniconda3/lib/python3.8/site-packages (from nltk==3.6.5) (7.1.2)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

Read file into RDD

```
text_file = sc.textFile("gs://big_data_ta/data/shakes.txt")
text_file.take(10)
```

```
[u"***The Project Gutenberg's Etext of Shakespeare's First Folio***",
 u'**********The Tragedie of Macbeth**********',
 u'',
 u'This is our 3rd edition of most of these plays.  See the index.',
 u'',
 u'',
 u'Copyright laws are changing all over the world, be sure to check',
 u'the copyright laws for your country before posting these files!!',
 u'',
 u'Please take a look at the important information in this header.']
```

Word count in Spark: split into words (2)

```
words = text_file.flatMap(lambda line: line.split(" ")).filter(lambda x: x != '')
words.take(10)
```

```
[u'***The',
 u'Project',
 u"Gutenberg's",
 u'Etext',
 u'of',
 u"Shakespeare's",
 u'First',
 u'Folio***',
 u'**********The',
 u'Tragedie']
```

Word count in Spark: form (k, v) pairs (3)

```
word_pairs = words.map(lambda x: (x, 1))
word_pairs.take(10)
```

```
[(u'***The', 1),
 (u'Project', 1),
 (u"Gutenberg's", 1),
 (u'Etext', 1),
 (u'of', 1),
 (u"Shakespeare's", 1),
 (u'First', 1),
 (u'Folio***', 1),
 (u'**********The', 1),
 (u'Tragedie', 1)]
```

Word count in Spark: reduce by aggregating (4)

```
word_pairs.reduceByKey(lambda a, b: a + b).take(10)
```

```
[(u'bidding', 1),
 (u'Lead', 1),
 (u'hart,', 1),
 (u'ever!', 1),
 (u'wracke,', 2),
 (u'protest', 1),
 (u'Barke', 1),
 (u'hate', 2),
 (u"knoll'd", 1),
 (u'grace,', 1)]
```

```
word_pairs.reduceByKey(lambda a, b: a + b).sortBy(lambda pair: -pair[1]).take(10)
```

```
[(u'the', 620),
 (u'and', 427),
 (u'of', 396),
 (u'to', 367),
 (u'I', 326),
 (u'a', 256),
 (u'you', 193),
 (u'in', 190),
 (u'is', 185),
 (u'my', 170)]
```

Summarized explanation:

1. Import statements:
   - `import pyspark`: Imports the PySpark library.
   - `import sys`: Imports the sys module for handling command-line arguments.

2. Argument checking:
   - Checks if exactly two command-line arguments are provided (input and output URIs).
   - Raises an exception if the number of arguments is incorrect.

3. Input and output URI assignment:
   - `inputUri = sys.argv[1]`: Assigns the first argument as the input file path.
   - `outputUri = sys.argv[2]`: Assigns the second argument as the output file path.

4. SparkContext creation:
   - `sc = pyspark.SparkContext()`: Creates a SparkContext, which is the entry point for Spark functionality.

5. Word count logic:
   - `lines = sc.textFile(inputUri)`: Reads the input file as an RDD of lines.
   - `words = lines.flatMap(lambda line: line.split())`: Splits each line into words.
   - `wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda count1, count2: count1 + count2)`:
     - Maps each word to a key-value pair (word, 1).
     - Reduces by key (word) to sum up the counts.

6. Result saving:
   - `wordCounts.saveAsTextFile(outputUri)`: Saves the resulting word counts to the specified output URI.

This program demonstrates key PySpark concepts:
- RDD operations: `textFile()`, `flatMap()`, `map()`, `reduceByKey()`
- Lambda functions for data transformation
- Distributed data processing (implicit in Spark's operations)

To run this program in industry, you would typically use the `spark-submit` command, providing the input and output file paths as arguments.

Next week tutorial

- Spark Dataframe and Spark SQL
- Spark MLlib
- HW1

References

- GCP Cloud Shell
  - https://cloud.google.com/shell/docs/quickstart
- Python functional programming
  - https://book.pythontips.com/en/latest/map_filter.html
  - https://medium.com/better-programming/lambda-map-and-filter-in-python-4935f248593
- Spark
  - RDD programming guide: https://spark.apache.org/docs/latest/rdd-programming-guide.html
  - Spark paper: https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/Zaharia.pdf
  - RDD paper: https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf