ASSIGNMENT 45.1 – BIG DATA & SPARK

Mar 2018 batch - Student: K. Anandaranga

---

1. Task 1

Given a list of numbers - List[Int] (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

- find the sum of all numbers

```
scala> val list = List(1,2,3,4,5,6,7,8,9,10)
list: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> list.reduceLeft(_+_)          // sum of all the numbers
res37: Int = 55
```

- find the total elements in the list

```
scala> list.length
res42: Int = 10
```

- calculate the average of the numbers in the list

```
scala> ((list.reduce(_+_)) / (list.length))        // average of numbers in list
res38: Int = 5
```

- find the sum of all the even numbers in the list

```
scala> list.filter(_% 2 == 0).reduce (_+_)        // sum of all even numbers in the list
res39: Int = 30
```

- find the total number of elements in the list divisible by both 5 and 3

```
scala> list.filter(x => x%3 ==0 && x%5 ==0).length // total of all elements divisible by both 5 and 3
res40: Int = 0
```

2. Task 2

1) Pen down the limitations of MapReduce.
2) What is RDD? Explain few features of RDD?
3) List down few Spark RDD operations and explain each of them.

Solution:

Limitations of MapReduce

- Cannot handle interactive processing
- Cannot perform real-time (stream) processing
- Cannot do iterative (delta) processing
- Cannot do in-memory processing or cacheing
- Cannot perform graph processing
- It has high latency which may be unsuitable in various applications
- It is not easy to use since it requires complex code for each and every operation
- Most often cannot handle complex machine-learning type algorithms
- With too many keys, sorting may take a very long time

RDD and its features

RDD stands for "Resilient Distributed Dataset" and is a fundamental data structure of Apache Spark.

- **In-memory computation:** stores intermediate results in RAM instead of disk
- **Lazy evaluations:** do not compute right away, but keep track of required transformations and compute only at the right time
- **Fault tolerance:** track data lineage and rebuild lost data automatically upon failure
- **Immutability:** data is safe to share across processes
- **Partitioning:** each partition is a logical division of data
- **Persistence:** user can state which RDD they would want to reuse
- **Coarse-grained operations:** achieved by map or filter or group-by operations
- **Location stickiness:** can define placement preference to compute partitions

A few Spark RDD operations

RDD in Apache Spark supports 2 types of operations **– (a) Transformation, (b) Action**

RDD Transformations are functions that take an RDD as input and produce one or many RDDs as output via lazy operations.

- **Narrow Transformations:** Data is from a single partition, and is result of map, filter, Flatmap, MapPartition, Sample and Union functions

- **Wide or Shuffle Transformations:** Data may live in many partitions of the parent RDD and use functions such as Intersection, Distinct, ReduceByKey, GroupByKey, Join, Cartesian, Repartition and Coalesce

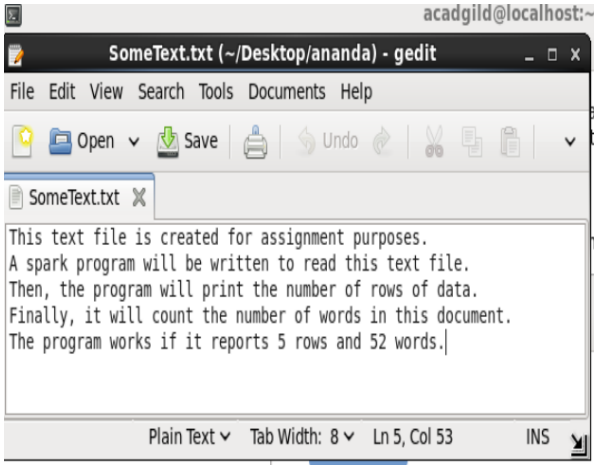
RDD Actions returns final result of RDD computations.

- It triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system.
- Actions produce non-RDD values.
- First(), take(), reduce(), collect(), the count() is some of the Actions in Spark.

3. Task 3

$spark-shell

- **Write a program to read a text file and print the number of rows of data in the document.**




```
scala> val lines = sc.textFile("file:///home/acadgild/Desktop/ananda/SomeText.txt") // read the text file
lines: org.apache.spark.rdd.RDD[String] = file:///home/acadgild/Desktop/ananda/SomeText.txt MapPartitionsRDD[228] at textFile
 at <console>:27

scala> lines.foreach(println)  // see what is in the text file
This text file is created for assignment purposes.
A spark program will be written to read this text file.
Then, the program will print the number of rows of data.
Finally, it will count the number of words in this document.
The program works if it reports 5 rows and 52 words.

scala> lines.count() // count of number of rows
res47: Long = 5

scala> lines.first() // print the first row
res48: String = This text file is created for assignment purposes.
```

- **Write a program to read a text file and print the number of words in the document.**

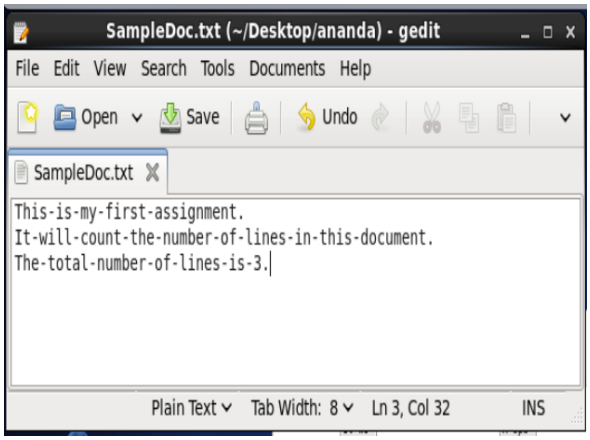
```
scala> val textlines = sc.textFile("file:///home/acadgild/Desktop/ananda/SomeText.txt") // read the text file
textlines: org.apache.spark.rdd.RDD[String] = file:///home/acadgild/Desktop/ananda/SomeText.txt MapPartitionsRDD[230] at text
File at <console>:27

scala> val words = textlines.flatMap(line => line.split(" ")) // splitting based on space character
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[231] at flatMap at <console>:28

scala> words.count // total word count
res49: Long = 52
```

- **We have a document where the word separator is -, instead of space. Write a spark code, to obtain the count of the total number of words present in the document.**

Sample document :
This-is-my-first-assignment.
It-will-count-the-number-of-lines-in-this-document.
The-total-number-of-lines-is-3




```
scala> val textlines = sc.textFile("file:///home/acadgild/Desktop/ananda/SampleDoc.txt") // read the text file
textlines: org.apache.spark.rdd.RDD[String] = file:///home/acadgild/Desktop/ananda/SampleDoc.txt MapPartitionsRDD[233] at tex
tFile at <console>:27

scala> val words = textlines.flatMap(line => line.split("-")) // splitting based on - character
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[234] at flatMap at <console>:28

scala> words.count // total word count
res50: Long = 22

scala> textlines.count // total line count
res51: Long = 3
```