BIG DATA HADOOP AND SPARK DEVLOPMENT

ASSIGNMENT 18

Table of Contents:

BIG DATA HADOOPAND SPARK DEVELOPMENT

1. Introduction

In this assignment, the given tasks are performed and Output of the tasks are recorded in the form of Screenshots.

2. Objective

This Assignment consolidates the deeper understanding of the Session – 18 INTRODUCTION TO SPARK

3. Problem Statement

- ### 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
  - find the total elements in the list
  - calculate the average of the numbers in the list
  - find the sum of all the even numbers in the list
  - find the total number of elements in the list divisible by both 5 and 3

- ### Task 2

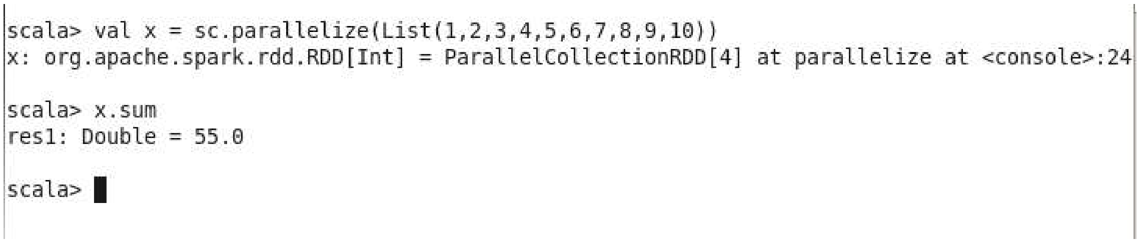
  1) Pen down the limitations of MapReduce.
  2) What is RDD? Explain few features of RDD?
  3) List down few Spark RDD operations and explain each of them.

Expected Output

- Task 1

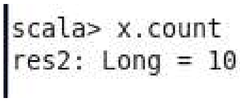
Given a list of numbers - List[Int] (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

- find the sum of all numbers



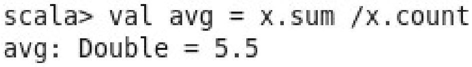
```
scala> val x = sc.parallelize(List(1,2,3,4,5,6,7,8,9,10))
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize at <console>:24

scala> x.sum
res1: Double = 55.0

scala> █
```

- find the total elements in the list



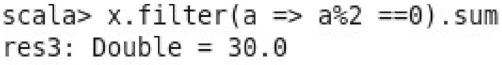
```
scala> x.count
res2: Long = 10
```

- calculate the average of the numbers in the list



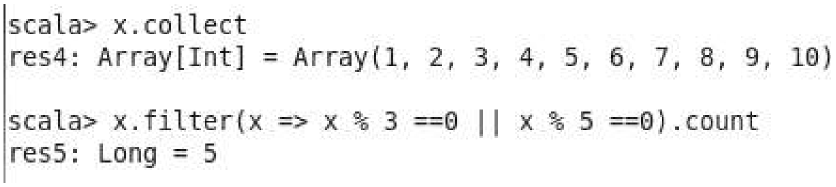
```
scala> val avg = x.sum /x.count
avg: Double = 5.5
```

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



```
scala> x.filter(a => a%2 ==0).sum
res3: Double = 30.0
```

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



```
scala> x.collect
res4: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> x.filter(x => x % 3 ==0 || x % 5 ==0).count
res5: Long = 5
```

- Task 2

1) Pen down the limitations of MapReduce.

All the major limitations are listed as below.

• Issue with small files
- Hadoop is not suited for small data. (HDFS) Hadoop distributed file system lacks the ability to efficiently support the random reading of small files because of its high capacity design.
• Slow Processing speed
- MapReduce process large data sets. There are tasks that need to be performed: Map and Reduce and, MapReduce requires a lot of time to perform these tasks thereby increasing latency. Data is distributed and processed over the cluster in MapReduce which increases the time and reduces processing speed.
• Support for Batch Processing only
- Hadoop supports batch processing only, it does not process streamed data, and hence overall performance is slower. MapReduce framework of Hadoop does not leverage the memory of the Hadoop cluster to the maximum.
• No Real-time data processing
- Apache Hadoop is designed for batch processing, that means it take a huge amount of data in input, process it and produce the result. Although batch processing is very efficient for processing a high volume of data, but depending on the size of the data being processed and computational power of the system, an output can be delayed significantly. Hadoop is not suitable for Real-time data processing.
• No Delta Iteration
- Hadoop is not so efficient for iterative processing, as Hadoop does not support cyclic data flow(i.e. a chain of stages in which each output of the previous stage is the input to the next stage).
• Latency
- In Hadoop, MapReduce framework is comparatively slower, since it is designed to support different format, structure and huge volume of data. In MapReduce, Map takes a set of data and converts it into another set of data, where individual element are broken down into key value pair and Reduce takes the output from the map as input and process further and MapReduce requires a lot of time to perform these tasks thereby increasing latency.
• Not easy to use
- In Hadoop, MapReduce developers need to hand code for each and every operation which makes it very difficult to work. MapReduce has no interactive mode, but adding one such as hive and pig makes working with MapReduce a little easier for adopters.
• Security
- Hadoop can be challenging in managing the complex application. If the user doesn't know how to enable platform who is managing the platform, your data could be at huge risk. At storage and network levels, Hadoop is missing encryption, which is a major point of concern. Hadoop supports Kerberos authentication, which is hard to manage. HDFS supports access control lists (ACLs) and a traditional file permissions model. However, third party vendors have enabled an organization to leverage Active Directory Kerberos and LDAP for authentication.
• No Abstraction
- Hadoop does not have any type of abstraction so MapReduce developers need to hand code for each and every operation which makes it very difficult to work.

- Vulnerable by Nature

- Hadoop is entirely written in java, a language most widely used, hence java been most heavily exploited by cyber criminals and as a result, implicated in numerous security breaches.

- No Caching

- Hadoop is not efficient for caching. In Hadoop, MapReduce cannot cache the intermediate data in memory for a further requirement which diminishes the performance of Hadoop.

- Lengthy line of code

- Hadoop has 1,20,000 line of code, the number of lines produces the number of bugs and it will take more time to execute the program.

- Uncertainty

- Hadoop only ensures that data job is complete, but it’s unable to guarantee when the job will be complete.

2) What is RDD? Explain few features of RDD?

An RDD in Spark is simply an automatic immutable distributed collection of objects. Each RDD is split into multiple partitions (), which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user defined classes. Features of RDD:

- RDD is a part of Spark
- RDDs understand data in terms of partitions.
- RDDs don’t store data.
- RDD just understands data, from HDFS or local machine (or on the fly)
- They are also immutable.

3) List down few Spark RDD operations and explain each of them.

RDD's perform two types of operations
– Transformations and Actions.

Transformations:

Any function that returns an RDD is a transformation, elaborating it further we can say that Transformation is functions which create a new data set from an existing one by passing each data set element through a function and returns a new RDD representing the results.

All transformations in Spark are lazy. They do not compute their results right away. Instead, they just remember the transformations applied to some base data set (e.g. a file). The transformations are only computed when an action requires a result that needs to be returned to the driver program. In the below example shown in the screenshot, we have applied the transformation using filter function. Now let us see some transformations like map, flatmap, filter which are commonly used.

Map: Map will take each row as input and return an RDD for the row. Below is the sample demonstration of the above scenario.
Flat map: flatMap will take an iterable data as input and returns the RDD as the contents of the iterator.

Filter: filter returns an RDD which meets the filter condition. Below is the sample demonstration of the above scenario.
ReduceByKey: reduceByKey takes a pair of key and value pairs and combines all the values for each unique key.

Actions:
Collect: collect is used to return all the elements in the RDD.
Count: count is used to return the number of elements in the RDD.
CountByValue: countByValue is used to count the number of occurrences of the elements in the RDD.
Reduce: Below is the sample demonstration of the above scenario where we have taken the total number of medals column in the dataset and loaded into the RDD map_test1. On this RDD we are performing reduce operation.

Finally, we have got that there is a total of 9529 medals declared as the winners in Olympic.
Take: take will display the number of records we explicitly specify.