

Session 18:
INTRODUCTION TO SPARK
Assignment 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
- 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

Program:

Created a new Spark program below,

```
Task1.scala Task2.scala pom.xml SimpleSpark.scala *Assignment18.scala
package com.hadoop.spark.demoproject
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

object Assignment18 {
  def main(args: Array[String]): Unit = {
    /*
     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
    */
    val conf = new SparkConf().setAppName("HelloSpark").setMaster("local")
    val sc = new SparkContext(conf)

    val numbers : List[Int] = List(1,2,3,4,5,6,7,8,9,10)
    val distData = sc.parallelize(numbers)

    val sum = distData.reduce(_+_ ) // Find sum of all numbers
    val elementCount = distData.count() // Find the total elements in the list
    val average = sum / elementCount // Calculate the average of numbers in the list
    val evenElements = distData.filter(element => element % 2 == 0) // Filter the even number elements and creates new RDD "evenElements"
    val sumOfEvenElements = evenElements.reduce(_+_ ) // Calculate the sum of even elements
    val divisibleElements = distData.filter(element => (element%5 == 0) && (element%3 == 0)).count() // Find the total number of elements in the list divisible by both 5 and 3


    println("-----")
    println("SUM OF ALL NUMBERS : " + sum)
    println("TOTAL ELEMENTS IN THE LIST : " + elementCount)
    println("AVERAGE OF ELEMENTS IN THE LIST : " + average)
    println("SUM OF EVEN NUMBERS IN THE LIST : " + sumOfEvenElements)
    println("TOTAL ELEMENTS DIVISIBLE BY BOTH 5 AND 3 : " + divisibleElements)
    println("-----")
  }
}
```

Output:

```
19/01/13 20:45:40 INFO TaskSetManager: Starting task 0.0 in stage 3.0 (TID 3, localhost, executor driver, parti
19/01/13 20:45:40 INFO Executor: Running task 0.0 in stage 3.0 (TID 3)
19/01/13 20:45:40 INFO Executor: Finished task 0.0 in stage 3.0 (TID 3). 703 bytes result sent to driver
19/01/13 20:45:40 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 3) in 6 ms on localhost (executor c
19/01/13 20:45:40 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
19/01/13 20:45:40 INFO DAGScheduler: ResultStage 3 (count at Assignment18.scala:29) finished in 0.016 s
19/01/13 20:45:40 INFO DAGScheduler: Job 3 finished: count at Assignment18.scala:29, took 0.021008 s

-----
SUM OF ALL NUMBERS : 55
TOTAL ELEMENTS IN THE LIST : 10
AVERAGE OF ELEMENTS IN THE LIST : 5
SUM OF EVEN NUMBERS IN THE LIST : 30
TOTAL ELEMENTS DIVISIBLE BY BOTH 5 AND 3 : 0
-----

19/01/13 20:45:40 INFO SparkUI: Stopped Spark web UI at http://ASPLAPNAV118.Aspiresys.com:4040
19/01/13 20:45:40 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
19/01/13 20:45:40 INFO MemoryStore: MemoryStore cleared
19/01/13 20:45:40 INFO BlockManager: BlockManager stopped
19/01/13 20:45:40 INFO BlockManagerMaster: BlockManagerMaster stopped
19/01/13 20:45:40 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoor
19/01/13 20:45:40 INFO SparkContext: Successfully stopped SparkContext
19/01/13 20:45:40 INFO ShutdownHookManager: Shutdown hook called
19/01/13 20:45:40 INFO ShutdownHookManager: Deleting directory C:\Users\saravanan.ponnaiah\AppData\Lo
```



TASK 2

- 1) Pen down the limitations of MapReduce.

Though MapR is suitable for batch processing, there are still certain limitations for MapR to handle,

- a. Does not support real-time processing
- b. Does not support iterative processing. That is, when we need to process a set of data repeatedly, then MapR is not a suggested option to choose.
- c. Does not support In-memory processing. The response time of MapR program is not as much fast as other Hadoop components like Spark in which data are processed in in-memory. As data are processed within in-memory, there will be very less disk I/O, thereby it increases the performance and reduce response time.
- d. MapR program is comparatively complex to develop than other processing components like Pig, Spark
- e. Does not support graph processing

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

Resilient Distributed Dataset (RDD) is the fundamental and primary data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided in to logical partitions that can be used to compute in different nodes of a cluster. An RDD is a read-only, partitioned

collection of records. As it is read-only, it is fault tolerant and will persist to recover when one or multiple node failure.

RDD can be created in 2 ways – i.) It can be created by referencing a dataset from external file system like HDFS, S3 bucket, HBase ii.) it can be created by parallelizing an existing RDD collection. Spark maintains RDD collection in memory instead of disk, it helps to achieve faster and efficient MapR operations.

Features of RDD:

- Resilient – Fault tolerant with lineage graph. In case of any node failure, the RDD can be recovered.
- Distributed in nature – Data are partitioned and resides in distributed nodes in HDFS
- In-memory – Data in RDD are maintained in in-memory. Thus, it helps to achieve faster performance.
- Immutable – RDDs are read-only. Thus it provides consistency across distributed nodes.
- Lazy evaluation – The original data processing will be performed only after action is triggered. During transformations, no data processing will be performed.
- Cacheable – Data can be cached in in-memory for iterative operations using persist() or cache() methods.

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

In Spark, there are 2 different types of RDD operations,

- Transformations – During transformation, it will operate on data and will create another RDD as a result. It is similar to mapper in MapReduce. It takes only RDD as input.
- Actions – During actions, it will trigger the original computation or processing of data. Transformations will be triggered only after action is called. It is similar to reducer in MapReduce.

Below are some of the operations in RDD,

Operation	Type	Description
MAP	Transformation	Returns a new RDD formed by passing each element from source RDD through a function. Ex: <code>Val x = sc.parallelize(Array(1,2,3))</code> <code>Val y = x.map(a => (a,1))</code> Output: (1,1),(2,1),(3,1)
FILTER	Transformation	Returns a new RDD formed by selecting the elements from source RDD that satisfy certain condition. Ex: <code>Val x = sc.parallelize(Array(1,2,3,4))</code> <code>Val y = x.filter(a => a%2 == 0)</code>

		Output: 2,4
FLATMAP	Transformation	Similar to map, but each input item can be mapped to 0 or more output items
COLLECT	Action	Return all the elements of the dataset as an array. This is mostly used after transformation is executed that returns a subset of data.
COUNT	Action	Returns the number of elements in the dataset
SaveAsTextFile	Action	Writes the elements of the dataset as a text file in the local file system, HDFS or any other Hadoop supported file systems.