**Assignment – 43**

**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

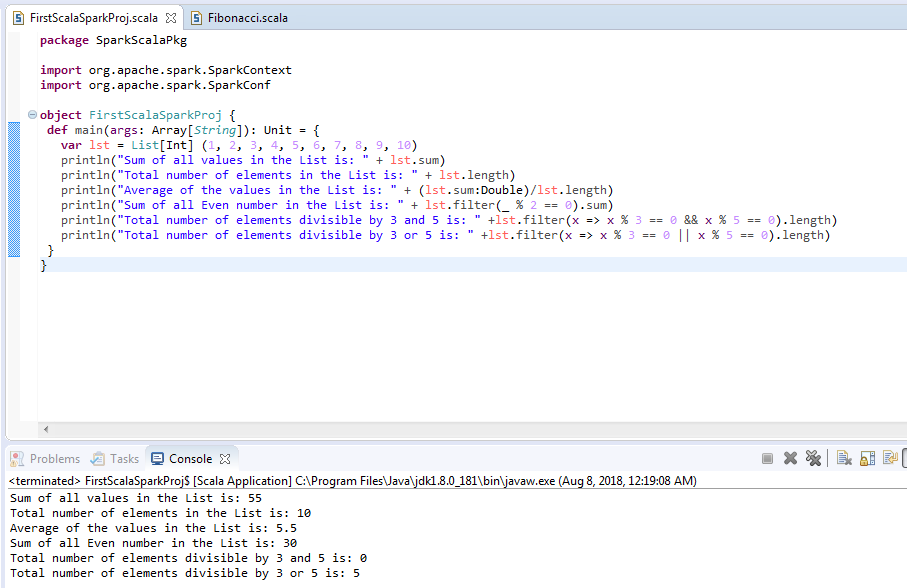
- find the total elements in the list

- 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

**Below file available in “FirstScalaSparkProj.scala” in GitHub**



**Task 2**

1. Pen down the limitation of MapReduce.
2. 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.
3. In Hadoop, with a parallel and distributed algorithm, 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.
4. Apache Hadoop is designed for batch processing, that means it take a huge amount of data in input, process it and produce the result. Hadoop is not suitable for Real-time data processing.
5. 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).
6. 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.
7. In Hadoop, MapReduce developers need to hand code for each and every operation which makes it very difficult to work.
8. 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.
9. 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.
10. What is RDD? Explain few features of RDD?
11. An RDD is simply a distributed collection of elements. In Spark all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result. Under the hood, Spark automatically distributes the data contained in RDDs across your cluster and parallelizes the operations you perform on them.
12. An RDD in Spark is simply an immutable distributed collection of objects. Each RDD is split into multiple partitions, which may be computed on different nodes of the cluster.
13. Decomposing the name RDD:
14. Resilient, i.e. fault-tolerant with the help of RDD lineage graph(DAG) and so able to recompute missing or damaged partitions due to node failures.
15. Distributed, since Data resides on multiple nodes.
16. Dataset represents records of the data you work with. The user can load the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.
17. Features of RDD
18. In-memory computation- The data inside RDD are stored in memory for as long as you want to store. Keeping the data in-memory improves the performance by an order of magnitudes.
19. Lazy Evaluation- The data inside RDDs are not evaluated on the go. The changes or the computation is performed only after an action is triggered. Thus, it limits how much work it has to do.
20. Fault Tolerance- Upon the failure of worker node, using lineage of operations we can re-compute the lost partition of RDD from the original one. Thus, we can easily recover the lost data.
21. Immutability- RDDS are immutable in nature meaning once we create an RDD we cannot manipulate it. And if we perform any transformation, it creates new RDD. We achieve consistency through immutability.
22. Persistence- We can store the frequently used RDD in in-memory and we can also retrieve them directly from memory without going to disk, this speedup the execution. We can perform Multiple operations on the same data, this happens by storing the data explicitly in memory by calling persist() or cache() function.
23. Partitioning- RDD partition the records logically and distributes the data across various nodes in the cluster. The logical divisions are only for processing and internally it has no division. Thus, it provides parallelism.
24. Parallel- Rdd, process the data parallelly over the cluster.
25. List down few Spark RDD operations and explain each of them.
    1. **Transformations**

Transformations are operations on RDDs that return a new RDD. Transformed RDDs are computed lazily, only when you use them in an action.

Examples:

**val** inputRDD **=** sc.textFile("log.txt") **//CREATE AN RDD**

**val** errorsRDD **=** inputRDD.filter(line **=>** line.contains("error")) **//TRANSFORM RDD**

* 1. **Actions**

Actions are the second type of RDD operation. They are the operations that return a final value to the driver program or write data to an external storage system. Actions force the evaluation of the transformations required for the RDD they are called on, since they are required to actually produce output.

**Examples:**

println("Input had " + inputRDD.count() + " lines") //Action to get count

println("Here are 10 examples:")

inputRDD.take(10).foreach(println) //Action to loop and Print top 10 lines

**Task 3**

1. Write a program to read a text file and print the number of rows of data in the document.
2. Write a program to read a text file and print the number of words in the document.
3. We have a document where the word separator is -, instead of space. Write a spark
4. code, to obtain the count of the total number of words present in the document.

**Below file available in “FirstScalaSparkProj.scala” in GitHub**

