Assignment 18

Task1

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

Command

```
val x = sc.parallelize(1 to 10)
```

a. **Find the sum of all numbers**

Command

```
val y = x.sum
```

b. **Find the total elements in the list**

Command

```
val z =x.count
```

c. **Calculate the average of the numbers in the list**

Command

```
val a =y/z
```

d. **Find the sum of all the even numbers in the list**

Command

```
val b = x.filter(s=>s%2==0).sum
```

e. **Find the total number of elements in the list divisible by both 5 and 3**

Command

```
val c=x.filter(s=>(s%3==0 & s%5==0)).count
```

Task2

a. Pen down the limitations of MapReduce

- Slow processing Speed

For large datasets, data is distributed and processed over the cluster in MapReduce which increases the time and reduces processing speed.

- Batch Processing

MapReduce jobs are designed for batch processing only. Real time processing is not possible when using MapReduce. Thus, instant results cannot be extracted.

- 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).

- Increased Latency

In MapReduce, the tasks are split into key-value pair during Map functionality and then Reduce functionality takes that as an input and processes further. This takes a lot of time for processing, thus, increasing the latency.

- More Lines of Code

For performing any task, one has to write more lines of code in MapReduce, when compared to that in Spark or Pig along with MapReduce.

- No Caching

MapReduce cannot cache the intermediate data in memory for a further requirement which diminishes the performance of Hadoop.

- Uncertainity of job completion

Due to batch processing in MapReduce, there is no certainity when will the job complete.

b. What is RDD? Explain few features of RDD?

Apache Spark RDDs (Resilient Distributed Datasets) are a basic abstraction of spark. They are an automatic immutable distributed collection of objects. These are logically partitioned that we can also apply parallel operations on them. Spark RDDs give power to users to control them. Above all, users may also persist an RDD in memory. Also, can reuse RDD efficiently across the parallel operation.

- RDD is an automatic immutable distributed collection of objects.
- RDD is Spark's way of understanding data, which is stored in different places, like HDFS, local file system.
- Each RDD is split into multiple partitions (), which may be computed on different nodes on the cluster.
- RDD does not store data. It just gives the reference to the data location.
- RDD understands data in terms of partition.

Features of RDD

- Immutable and Read-only, thus ensures consistency of data.
- Partitioned, thereby, ensuring parallel processing.
- Fault-tolerance: RDD recovers itself whenever data is lost.
- Coarse-grained operations: This ensures that we can perform an operation on the entire cluster at the same time.
- We can use as many number of RDDs as required for our data processing.

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

RDD Operations

There are 2 different RDD operations:

i. Transformations

They are the set of instructions given to Spark, to perform the changes we want to happen to the data. Transformations return RDD. Few of the transformations are as follows:

Transformation	Description
map(func)	Returns a new distributed dataset formed by passing each element of the source through a function func.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Sequence rather than a single item)
Union(other dataset)	Returns a new dataset that contains the union of the elements in the source dataset and the argument.

distinct([numPartitions]))	Returns a new dataset that contains the distinct elements of the source dataset.
coalesce(numPartitions)	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.
Filter(func)	Returns a new dataset formed by selecting those elements of the source on which funcreturns true.
Cartesian (other dataset)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).

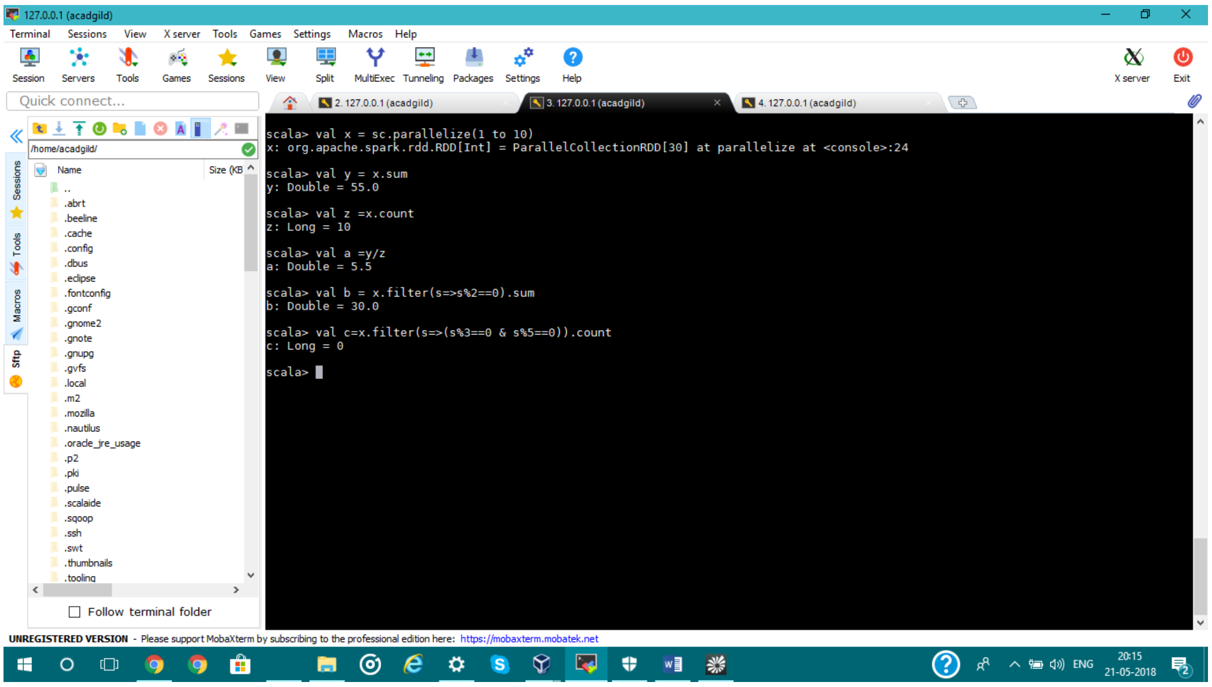
ii. <u>Actions</u>

Actions are operations, triggers the process by returning the result back to program. They return some other data type. Some of the actions are as follows:

Actions	**Description**
Reduce(func)	Aggregate the elements of the dataset using a function func (which takes two arguments and returns one)
Collect()	Return all the elements of the dataset as an array at the driver program.
Count()	Returns the number of elements in the dataset
First()	Returns the first element in the dataset
Take(n)	Returns an array with first n elements of the dataset
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
takeOrdered(n,[ordering])	Return the first n elements of the RDD using either their natural order or a custom comparator

<u>Screenshot</u>

Task1



```
scala> val x = sc.parallelize(1 to 10)
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[30] at parallelize at <console>:24

scala> val y = x.sum
y: Double = 55.0

scala> val z =x.count
z: Long = 10

scala> val a =y/z
a: Double = 5.5

scala> val b = x.filter(s=>s%2==0).sum
b: Double = 30.0

scala> val c=x.filter(s=>(s%3==0 & s%5==0)).count
c: Long = 0

scala> 
```

127.0.0.1 (acadgild)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
2. 127.0.0.1 (acadgild)
3. 127.0.0.1 (acadgild)
4. 127.0.0.1 (acadgild)
/home/acadgild/
Name Size (KB
Follow terminal folder
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
ENG 20:15 21-05-2018