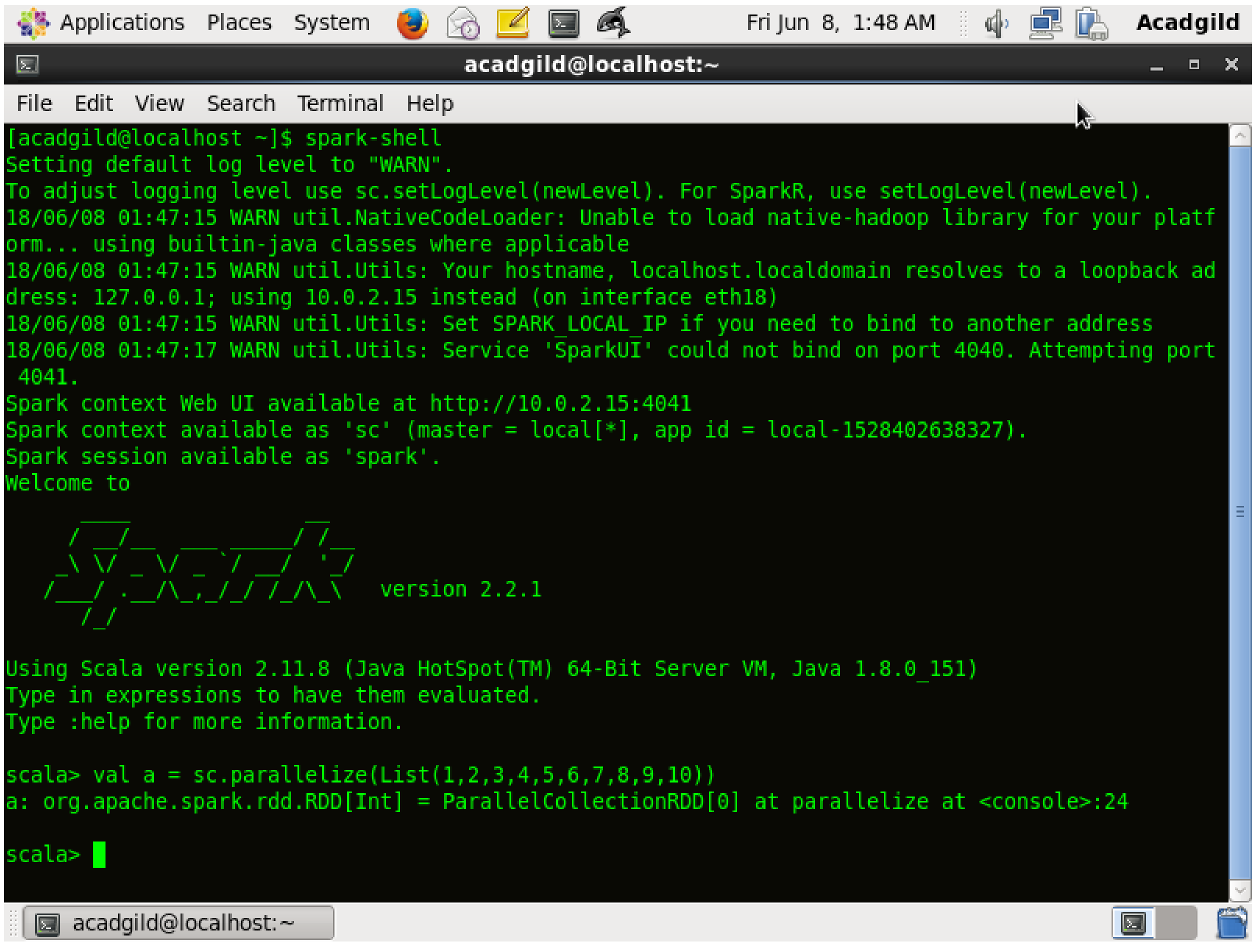
Enter spark scala shell by entering the command spark-shell

Use list keyword to form a list containing of following numbers and place it in parallelize to form spark RDD



```
[acadgild@localhost ~]$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
18/06/08 01:47:15 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platf
orm... using builtin-java classes where applicable
18/06/08 01:47:15 WARN util.Utils: Your hostname, localhost.localdomain resolves to a loopback ad
dress: 127.0.0.1; using 10.0.2.15 instead (on interface eth18)
18/06/08 01:47:15 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
18/06/08 01:47:17 WARN util.Utils: Service 'SparkUI' could not bind on port 4040. Attempting port
 4041.
Spark context Web UI available at http://10.0.2.15:4041
Spark context available as 'sc' (master = local[*], app id = local-1528402638327).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.2.1
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val a = sc.parallelize(List(1,2,3,4,5,6,7,8,9,10))
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24

scala> 
```

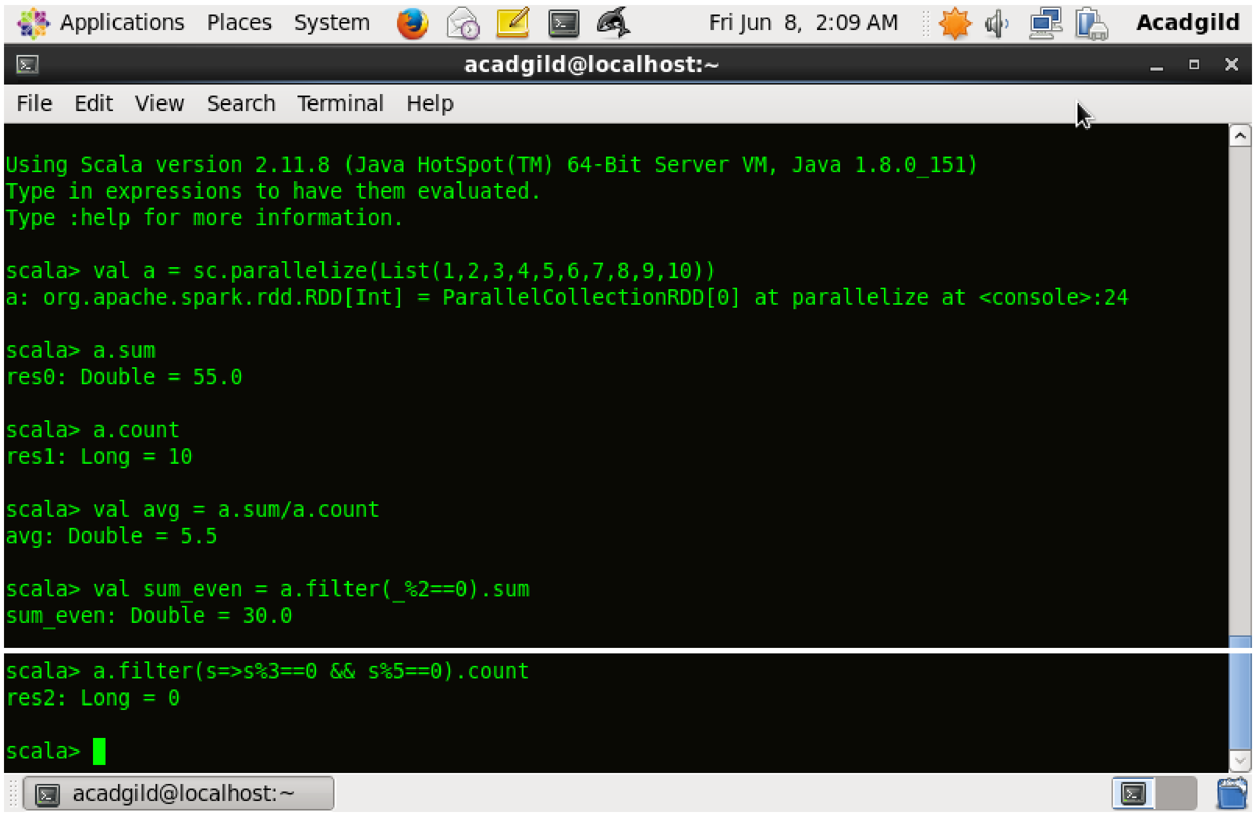
a.sum returns the sum of all the elemnts in the RDD list

a.count returns the length of the list

avg is found by using the above results

To find the sum of only even numbers sum is applied to filter function with the condition

Similarly to find the number of elements divisible by 5 and 3 again filter is used followed by count



```
Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val a = sc.parallelize(List(1,2,3,4,5,6,7,8,9,10))
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24

scala> a.sum
res0: Double = 55.0

scala> a.count
res1: Long = 10

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

scala> val sum_even = a.filter(_%2==0).sum
sum_even: Double = 30.0

scala> a.filter(s=>s%3==0 && s%5==0).count
res2: Long = 0

scala> 
```

Limitations of mapreduce

Issue with small files

Slow processing speed

Latency

No Real Time Data Processing

Length line of code

No caching

MapReduce is widely adopted for processing and generating large datasets with a parallel, distributed algorithm on a cluster. It allows users to write parallel computations, using a set of high-level operators, without having to worry about work distribution and fault tolerance.

What is RDD ? Explain new features of RDD?

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical 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.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs − **parallelizing** an existing collection in your driver program, or **referencing a dataset** in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also **persist** an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is **R**esilient **D**istributed **D**atasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is **R**esilient **D**istributed **D**atasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

Few spark RDD operations

reduceByKey takes a pair of key and value pairs and combines all the values for each unique key.

filter returns an RDD which meets the filter condition.

flatMap will take an iterable data as input and returns the RDD as the contents of the iterator.

count is used to return the number of elements in the RDD.

countByValue is used to count the number of occurrences of the elements in the RDD.

take will display the number of records we explicitly specify.