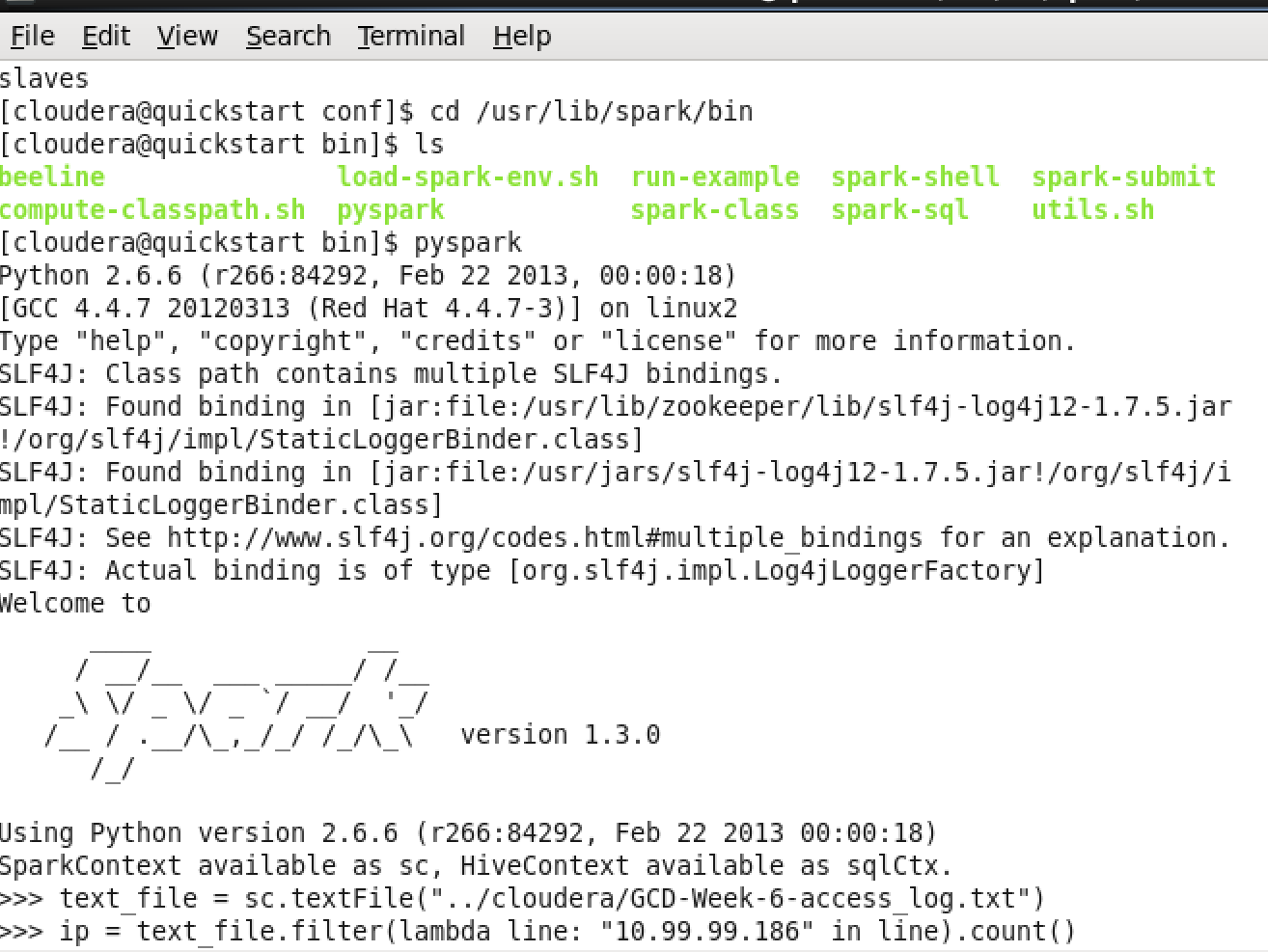
**GCD Assignment Week 6**

**Spark**

**Combine SQL, streaming, and complex analytics.**

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application. Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3. You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos. Access data in HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source. Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

**2.Open PySpark (pyspark) Result:**



### **Activity 2.1 : How many hits ? :**

text\_file = sc.textFile("../cloudera/GCD-Week-6-access\_log.txt")

assets = text\_file.filter(lambda line: "/assets/js/the-associates.js" in line).count()

count = 2456

### **Activity 2.2 : How many hits ? :**

text\_file = sc.textFile("../cloudera/GCD-Week-6-access\_log.txt")

ip = text\_file.filter(lambda line: "10.99.99.186" in line).count()

count = 6

### **Activity 2.3 : What is the most popular file on the website and how many times it occurs in the log ?**

text\_file = sc.textFile("./GCD-Week-6-access\_log.txt")

wordCounts = text\_file.flatMap(lambda line: line[52:line.find(" H")].split(' ')).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)

wordCounts.takeOrdered(20, lambda (key, value): -1 \* value)

### **Activity Gutenberg word count:**

text\_file = sc.textFile("./Gutenberg1/Gutenberg/GutenbergSmall/\*")

counts = text\_file.flatMap(lambda line: line.split(" ")) \

.map(lambda word: (word, 1)) \

.reduceByKey(lambda a, b: a + b)

counts .takeOrdered(20, lambda (key, value): -1 \* value)

#### **Activity explore spark basics:**

A Resilient Distributed Dataset (RDD), the basic abstraction in Spark. Represents an immutable, partitioned collection of elements that can be operated on in parallel. This class contains the basic operations available on all RDDs, such as map, filter, and persist. In addition, PairRDDFunctions contains operations available only on RDDs of key-value pairs, such as groupByKey and join; DoubleRDDFunctions contains operations available only on RDDs of Doubles; and SequenceFileRDDFunctions contains operations available on RDDs that can be saved as SequenceFiles. These operations are automatically available on any RDD of the right type (e.g. RDD[(Int, Int)] through implicit conversions when you import spark.SparkContext.\_.

Internally, each RDD is characterized by five main properties:

* A list of splits (partitions)
* A function for computing each split
* A list of dependencies on other RDDs
* Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
* Optionally, a list of preferred locations to compute each split on (e.g. block locations for an HDFS file)

#### **Rdd’s :**

**text\_file = sc.textFile("./GCD-Week-6-access\_log.txt")**

#### **Transformations :**

One of the RDD operations, transformation, creates a new dataset from existing one. For example, **map** is transformation that passes each dataset element through a function and returns a new RDD representing the results.

All transformations in Spark are *lazy*, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently – for example, we can realize that a dataset created through map will be used in a reduce and return only the result of the reduce to the driver, rather than the larger mapped dataset.

**result = text\_file.flatMap(lambda line: line[52:line.find(" H")].split(' ')).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)**

#### **Actions :**

Actions return a value to the driver program after running a computation on the dataset. For example, **reduce** is an action that aggregates all the elements of RDD using some function and returns the final result to the driver program (although there is also a parallel **reduceByKey** that returns a distributed dataset).

By default, each transformed RDD may be recomputed each time you run an action on it.

**result.collectAsMap()**