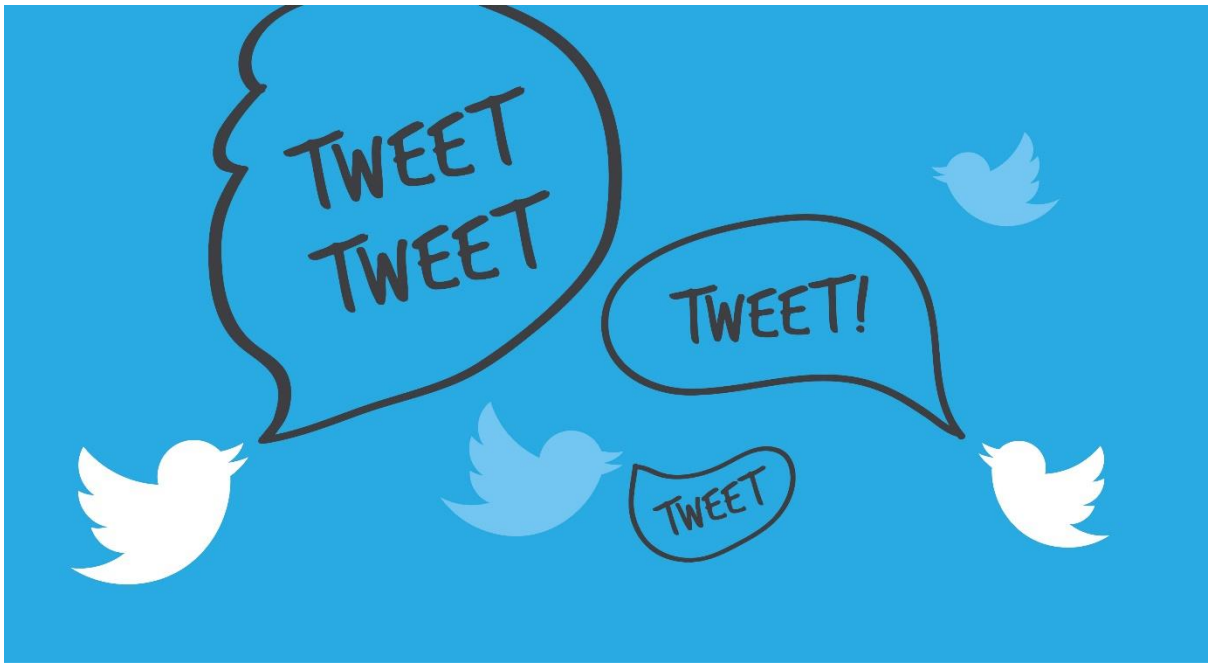
PRINCIPLES OF BIG DATA MANAGEMENT

PROJECT REPORT

PHASE-2



Submitted By:

Pavan Jakkepalli (16232736)

Chanikya Merugu(16229774)

Sangeetha Muppidi(16232846)

Swetcha Reddy Vangala(16232857)

Table of Contents

I. Project Theme

"Movies" is the theme we chose for analysing the tweets. Based on the data collected from twitter we proposed five varied analytical queries on our theme 'Movies'. Using this data, the queries are written in Scala which gives analysis on movies where more tweets were done and discussed. The data extracted from queries is displayed in the form of tables. Based on the obtained information we visualize the data in the further phases. Movies use social media for conversations and publicity. Promotion through social media plays a key role in making movie known to most of the people. This large data is holding up to be scrutinized.

II. Collecting Twitter tweets:

(i)About Twitter

Twitter is a huge source of data. At any point of time something is going on, people throughout the world starts tweeting endlessly. Twitter is kind of a place to search for huge data. It's a case of the ubiquitous administrations used by customers and organizations alike that helps to result this large amount of information in the start place.

(ii) Access tokens:

We have given access tokens from dev.twitter.com/apps.

AccessToken-"790254444306833408-VUXVM722Hj6CdIUaCycBFzKbA3WGnmO";
Access Secret- "kHDXSiv3ZTpS3h966nlnZkbd5dZzBJtl49aU5r4sLTpZ2";
Consumer Key- "71GFiDzYipT3O3rVRTQOuQMiT";
ConsumerSecret-"8YnMMwT1NxQEHLnyrlExYiGMamVfbDYR8fy219JovwpMaXvicy";

(iii) Streaming Twitter tweets:

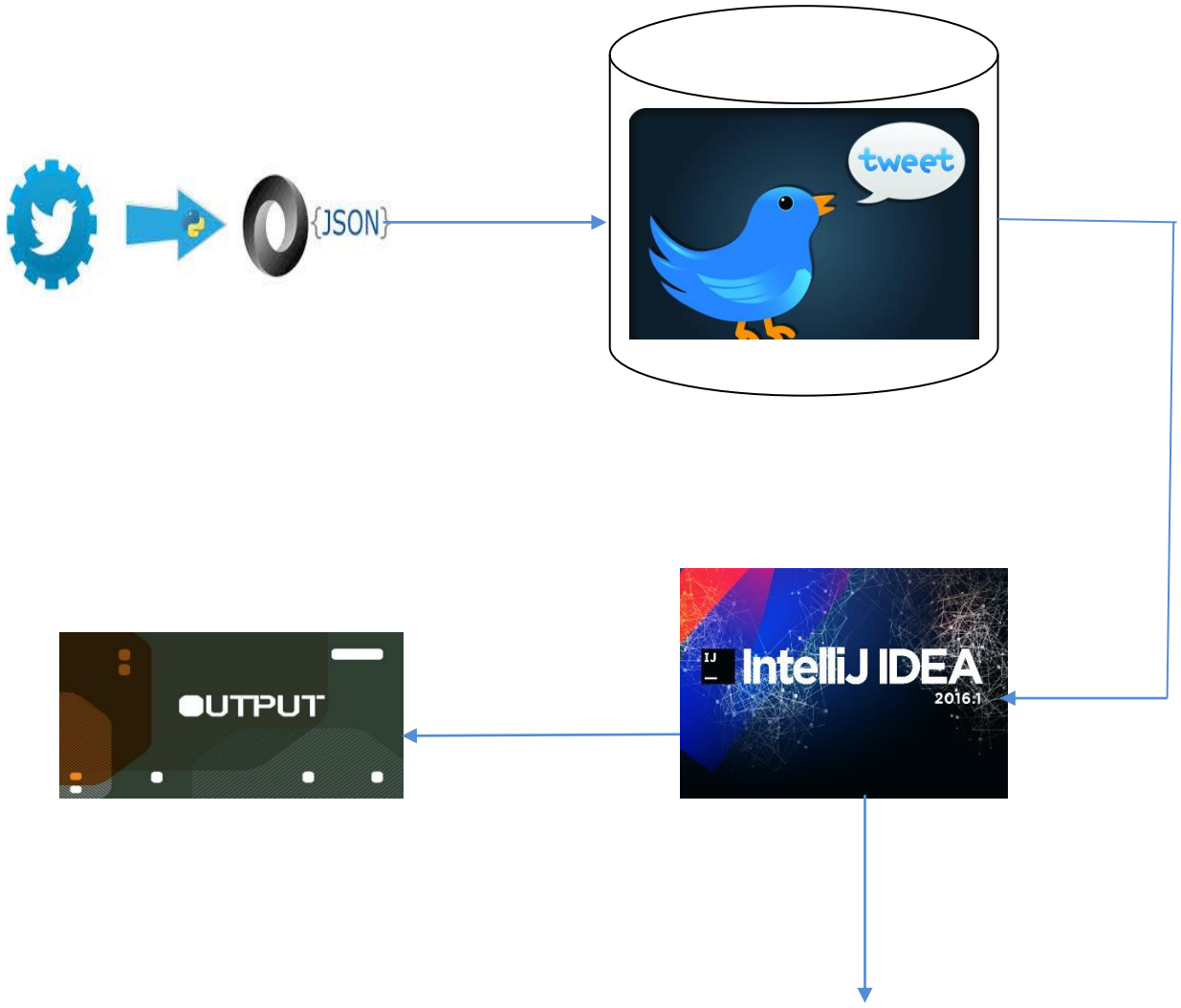
We have to download the twitter tweets by using Tweepy python and we have downloaded by giving our access tokens and "movies" as our search in python.

III.Architectural Design

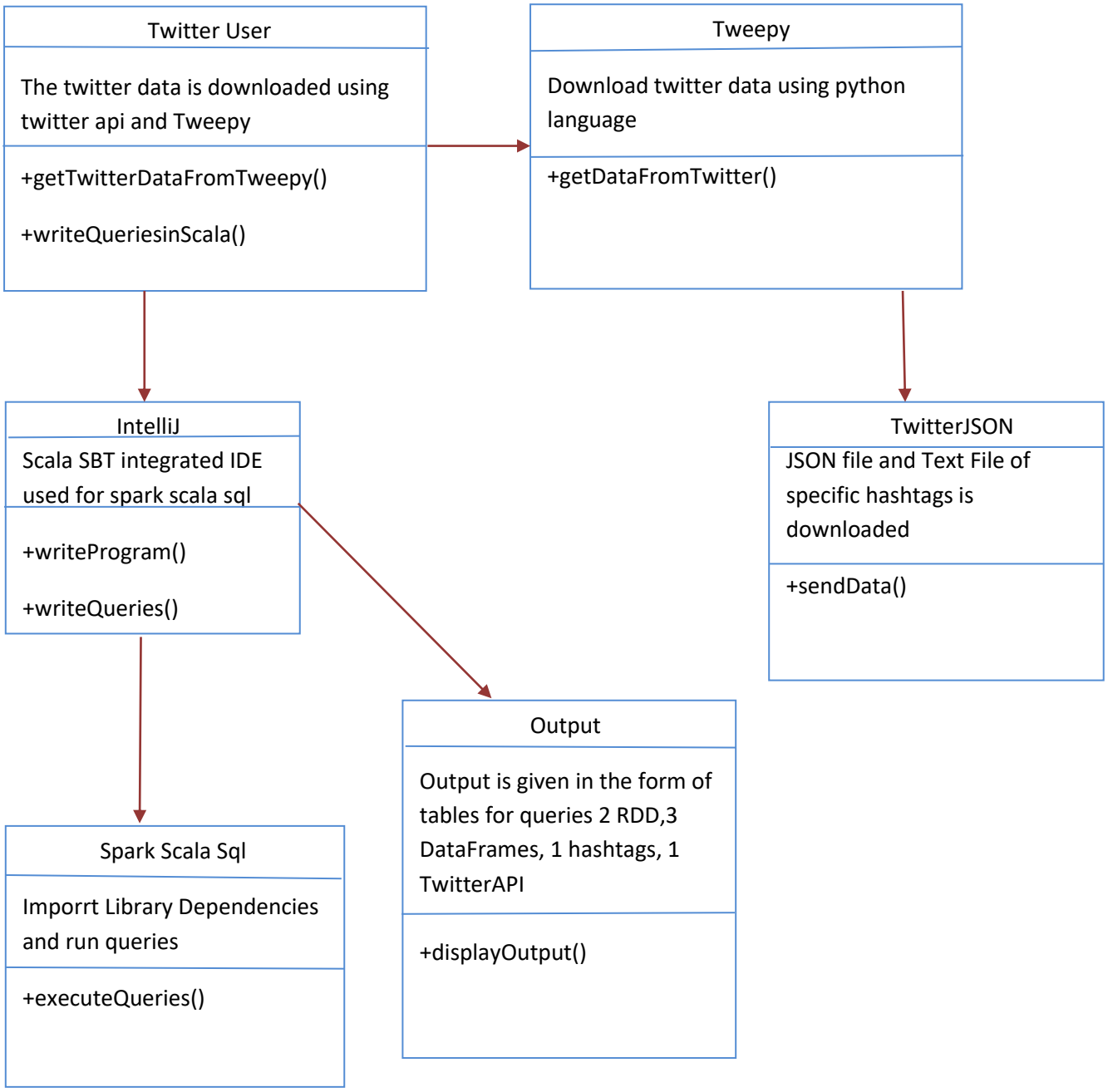
(i)The outline:

- Data collection i.e, collecting data on ‘Movies’ from twitter.
- Input data is loaded into Apache Spark
- Input is parsed into words.
- Reduce these words into a pair that contains the word and the count of it’s occurrences.
- Save the outputs.

(ii) Data Flow Diagram:



(iii)UML Diagram:



IV. Libraries, APIs and Programming languages

- Eclipse framework is used for implementing the Analytics using Apache Spark.
- Spark is locally used in our Java process space.
- Sparkconf object is designed which points to our Spark instance as in this case 'local' and in order to interact with Spark we require a Spark Context instance, for which we designed JavaSparkContext.
- Tweets are stored in Spark SQL database in tweet table and SQL query is run on this table for required data analytics.

(i)Library dependencies:

```
scalaVersion := "2.11.8"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.6.0"

libraryDependencies += "org.apache.spark" %% "spark-sql" % "1.7.0"

libraryDependencies  +=  "oauth.signpost" % "signpost-core" % "1.2"

libraryDependencies += "oauth.signpost" % "signpost-commonshttp4" % "1.2"

libraryDependencies += "org.apache.httpcomponents" % "httpclient" % "4.2"
```

(ii)Language Requirements:

- Python
- Scala
- Spark
- SQL

V. Development Machine's Specifications

Software Requirements:

- Scala 2.11.8
  Scala is a general-purpose programming language. Scala has full support for functional programming and a strong static type system. Designed to be concise, many of Scala's design decisions were inspired by criticism of Java's

shortcomings.Many versions of Scala have been released and the version which is currently in use is 2.11.8.

- **IntelliJ IDEA** is a Java integrated development environment (IDE) for developing computer software. It is developed by JetBrains (formerly known as IntelliJ), and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition. Both can be used for commercial development. Intellij IDEA Ultimate Edition has few additional advantages over Community Edition. For example, it supports additional languages such as HTML, CSS, Python, SQL that are not supported by Community Edition.

- **Python**

  Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

  Python supports multiple programming paradigms, including object oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

- **JDK 1.8**
  The **Java Development Kit** (**JDK**) is an implementation of either one of the Java Platform, Standard Edition; Java Platform, Enterprise Edition or Java Platform, Micro Edition platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, Mac OS X or Windows. The JDK includes a private JVM and a few other resources to finish the development of a Java Application. Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK)

- **Apache Spark 1.7.0**
  Apache Spark is an open source cluster computing framework. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.

  Apache Spark provides programmers with an application programming interface centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. It was developed in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

VI. Queries

(i)RDD Query 1:

The tweets that are done on foreign movies, South Indian movies and North Indian movie are filtered, and displays the count of each by using RDD operations(contains, filter, union)

```
build.sbt ×   Queries.scala ×   TwitterPull.scala ×
Queries   main(args: Array[String])

System.setProperty("hadoop.home.dir","C:\\Users\\sange\\Downloads\\hadooponwindows-master\\hadooponwindows-master")
val sparkConf = new SparkConf().setAppName("Queries").setMaster("local[*]")
val sc = new SparkContext(sparkConf)
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val lines= sc.textFile("C:\\Users\\sange\\Desktop\\Pavan\\tweets1.txt")
val tel=lines.filter(x=>x.contains("tel"))
val tam=lines.filter(z=>z.contains("tamil"))
val kan=lines.filter(z=>z.contains("kannada"))
val mal=lines.filter(z=>z.contains("malayalam"))
val hin=lines.filter(z=>z.contains("hindi"))
val pun=lines.filter(z=>z.contains("punjabi"))
val bho=lines.filter(z=>z.contains("bhojpuri"))
val ben=lines.filter(z=>z.contains("bengali"))
val kor=lines.filter(z=>z.contains("korean"))
val eng=lines.filter(y=>y.contains("eng"))
val fre=lines.filter(z=>z.contains("french"))
val jap=lines.filter(z=>z.contains("japanese"))
val chi=lines.filter(z=>z.contains("chinese"))
val south=mal.union(kan).union(tam).union(tel).count()
val foreign=eng.union(kor).union(fre).union(jap).union(chi).count()
val north=hin.union(pun).union(bho).union(ben).count()
println("South Indian Movies %s".format(south))
println("North Indian Movies %s".format(north))
println("Foreign Movies %s".format(foreign))
```

Output:

```
Run: TwitterPull  Queries
16/11/13 19:23:20 INFO TaskSetManager: Starting task 22.0 in stage 2.0 (TID 76, localhost, partition 22,PROCESS_LOCAL, 2176 bytes)
16/11/13 19:23:20 INFO TaskSetManager: Finished task 18.0 in stage 2.0 (TID 72) in 423 ms on localhost (19/24)
16/11/13 19:23:20 INFO Executor: Running task 22.0 in stage 2.0 (TID 76)
16/11/13 19:23:20 INFO HadoopRDD: Input split: file:/C:/Users/sange/Desktop/Pavan/tweets1.txt:134217728+33554432
16/11/13 19:23:20 INFO Executor: Finished task 19.0 in stage 2.0 (TID 73). 2082 bytes result sent to driver
16/11/13 19:23:20 INFO TaskSetManager: Starting task 23.0 in stage 2.0 (TID 77, localhost, partition 23,PROCESS_LOCAL, 2176 bytes)
16/11/13 19:23:20 INFO TaskSetManager: Finished task 19.0 in stage 2.0 (TID 73) in 427 ms on localhost (20/24)
16/11/13 19:23:20 INFO Executor: Running task 23.0 in stage 2.0 (TID 77)
16/11/13 19:23:20 INFO HadoopRDD: Input split: file:/C:/Users/sange/Desktop/Pavan/tweets1.txt:167772160+31558347
16/11/13 19:23:20 INFO Executor: Finished task 20.0 in stage 2.0 (TID 74). 2082 bytes result sent to driver
16/11/13 19:23:20 INFO TaskSetManager: Finished task 20.0 in stage 2.0 (TID 74) in 458 ms on localhost (21/24)
16/11/13 19:23:20 INFO Executor: Finished task 21.0 in stage 2.0 (TID 75). 2082 bytes result sent to driver
16/11/13 19:23:20 INFO TaskSetManager: Finished task 21.0 in stage 2.0 (TID 75) in 403 ms on localhost (22/24)
16/11/13 19:23:20 INFO Executor: Finished task 22.0 in stage 2.0 (TID 76). 2082 bytes result sent to driver
16/11/13 19:23:20 INFO TaskSetManager: Finished task 22.0 in stage 2.0 (TID 76) in 366 ms on localhost (23/24)
16/11/13 19:23:20 INFO Executor: Finished task 23.0 in stage 2.0 (TID 77). 2082 bytes result sent to driver
16/11/13 19:23:20 INFO TaskSetManager: Finished task 23.0 in stage 2.0 (TID 77) in 254 ms on localhost (24/24)
16/11/13 19:23:20 INFO DAGScheduler: ResultStage 2 (count at Queries.scala:35) finished in 2.439 s
16/11/13 19:23:20 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
16/11/13 19:23:20 INFO DAGScheduler: Job 2 finished: count at Queries.scala:35, took 2.473412 s
South Indian Movies 1677
North Indian Movies 166
Foreign Movies 1273
16/11/13 19:23:20 INFO JSONRelation: Listing file:/C:/Users/sange/Desktop/Pavan/data2.json on driver
16/11/13 19:23:20 INFO MemoryStore: Block broadcast_4 stored as values in memory (estimated size 127.9 KB, free 258.1 KB)
16/11/13 19:23:20 INFO MemoryStore: Block broadcast_4_piece0 stored as bytes in memory (estimated size 14.0 KB, free 272.1 KB)
16/11/13 19:23:20 INFO BlockManagerInfo: Added broadcast_4_piece0 in memory on localhost:51701 (size: 14.0 KB, free: 1128.3 MB)
16/11/13 19:23:20 INFO SparkContext: Created broadcast 4 from json at Queries.scala:44
16/11/13 19:23:20 INFO FileInputFormat: Total input paths to process : 1
16/11/13 19:23:20 INFO SparkContext: Starting job: json at Queries.scala:44
16/11/13 19:23:20 INFO DAGScheduler: Registering RDD 29 (json at Queries.scala:44)
16/11/13 19:23:20 INFO DAGScheduler: Got job 3 (json at Queries.scala:44) with 2 output partitions
Compilation completed successfully in 5s 234ms (a minute ago)
69 chars, 4 lines  479:1  CRLF  UTF-8
```

(ii)RDD Query 2:

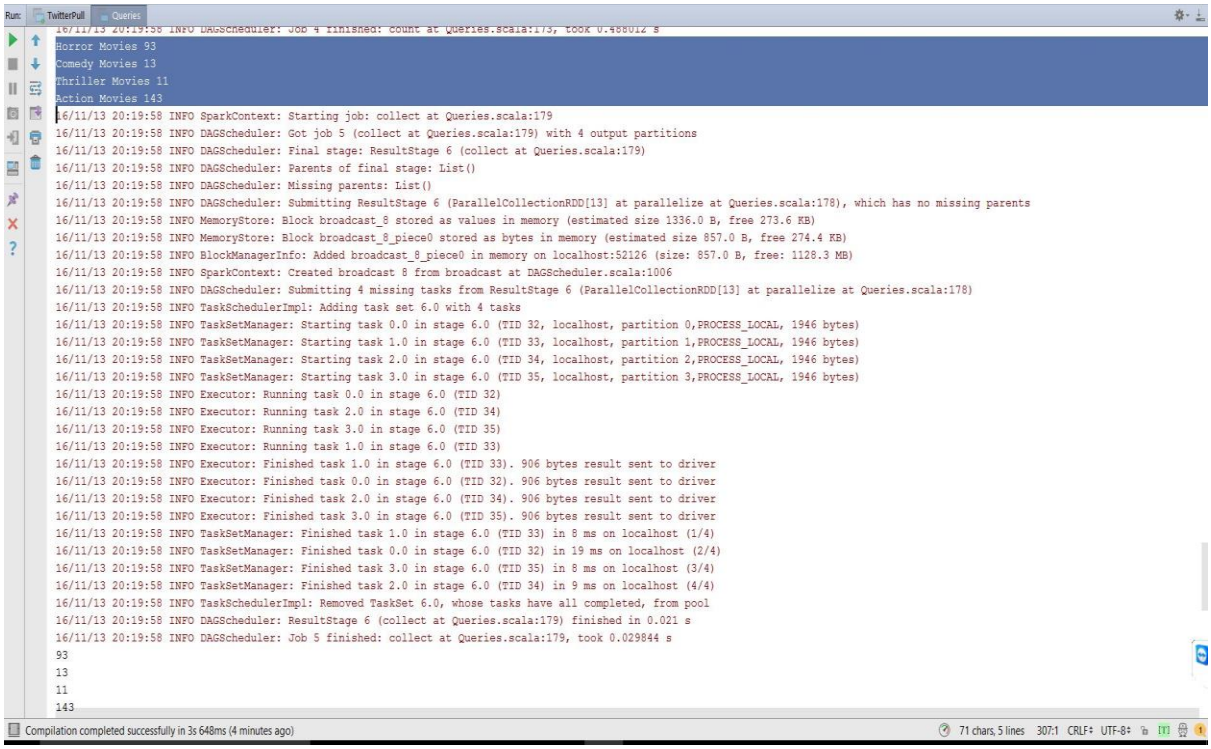
Finds the number of users searching the type of movies like horror, thriller, animated, comedy and prints the count. It also parallelizes the data and prints the same.

```
build.sbt ×  Queries.scala ×  TwitterPull.scala ×
Queries  main(args: Array[String])

object Queries{
  def main(args: Array[String]): Unit = {

    System.setProperty("hadoop.home.dir","C:\\Users\\sange\\Downloads\\hadooponwindows-master\\hadooponwindows-master")
    val sparkConf = new SparkConf().setAppName("Queries").setMaster("local[*]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    val lines= sc.textFile("C:\\Users\\sange\\Desktop\\Pavan\\tweets1.txt")
    val df=sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\data2.json")
    df.registerTempTable("dftable")
    val horror=lines.filter(line=>line.contains("#horror")).count()
    val comedy=lines.filter(line=>line.contains("#comedy")).count()
    val thriller=lines.filter(line=>line.contains("#thriller")).count()
    val action=lines.filter(line=>line.contains("#action")).count()
    println("Horror Movies %s".format(horror))
    println("Comedy Movies %s".format(comedy))
    println("Thriller Movies %s".format(thriller))
    println("Action Movies %s".format(action))
    val rdd1 = sc.parallelize(List(horror,comedy,thriller,action))
    rdd1.collect().foreach(println)
```

Output:



```
Run: TwitterPull  Queries
16/11/13 20:19:58 INFO DAGScheduler: Job 4 finished: count at Queries.scala:173, took 0.488012 s
Horror Movies 93
Comedy Movies 13
Thriller Movies 11
Action Movies 143
16/11/13 20:19:58 INFO SparkContext: Starting job: collect at Queries.scala:179
16/11/13 20:19:58 INFO DAGScheduler: Got job 5 (collect at Queries.scala:179) with 4 output partitions
16/11/13 20:19:58 INFO DAGScheduler: Final stage: ResultStage 6 (collect at Queries.scala:179)
16/11/13 20:19:58 INFO DAGScheduler: Parents of final stage: List()
16/11/13 20:19:58 INFO DAGScheduler: Missing parents: List()
16/11/13 20:19:58 INFO DAGScheduler: Submitting ResultStage 6 (ParallelCollectionRDD[13] at parallelize at Queries.scala:178), which has no missing parents
16/11/13 20:19:58 INFO MemoryStore: Block broadcast_8 stored as values in memory (estimated size 1336.0 B, free 273.6 KB)
16/11/13 20:19:58 INFO MemoryStore: Block broadcast_8_piece0 stored as bytes in memory (estimated size 857.0 B, free 274.4 KB)
16/11/13 20:19:58 INFO BlockManagerInfo: Added broadcast_8_piece0 in memory on localhost:52126 (size: 857.0 B, free: 1128.3 MB)
16/11/13 20:19:58 INFO SparkContext: Created broadcast 8 from broadcast at DAGScheduler.scala:1006
16/11/13 20:19:58 INFO DAGScheduler: Submitting 4 missing tasks from ResultStage 6 (ParallelCollectionRDD[13] at parallelize at Queries.scala:178)
16/11/13 20:19:58 INFO TaskSchedulerImpl: Adding task set 6.0 with 4 tasks
16/11/13 20:19:58 INFO TaskSetManager: Starting task 0.0 in stage 6.0 (TID 32, localhost, partition 0,PROCESS_LOCAL, 1946 bytes)
16/11/13 20:19:58 INFO TaskSetManager: Starting task 1.0 in stage 6.0 (TID 33, localhost, partition 1,PROCESS_LOCAL, 1946 bytes)
16/11/13 20:19:58 INFO TaskSetManager: Starting task 2.0 in stage 6.0 (TID 34, localhost, partition 2,PROCESS_LOCAL, 1946 bytes)
16/11/13 20:19:58 INFO TaskSetManager: Starting task 3.0 in stage 6.0 (TID 35, localhost, partition 3,PROCESS_LOCAL, 1946 bytes)
16/11/13 20:19:58 INFO Executor: Running task 0.0 in stage 6.0 (TID 32)
16/11/13 20:19:58 INFO Executor: Running task 2.0 in stage 6.0 (TID 34)
16/11/13 20:19:58 INFO Executor: Running task 3.0 in stage 6.0 (TID 35)
16/11/13 20:19:58 INFO Executor: Running task 1.0 in stage 6.0 (TID 33)
16/11/13 20:19:58 INFO Executor: Finished task 1.0 in stage 6.0 (TID 33). 906 bytes result sent to driver
16/11/13 20:19:58 INFO Executor: Finished task 0.0 in stage 6.0 (TID 32). 906 bytes result sent to driver
16/11/13 20:19:58 INFO Executor: Finished task 2.0 in stage 6.0 (TID 34). 906 bytes result sent to driver
16/11/13 20:19:58 INFO Executor: Finished task 3.0 in stage 6.0 (TID 35). 906 bytes result sent to driver
16/11/13 20:19:58 INFO TaskSetManager: Finished task 1.0 in stage 6.0 (TID 33) in 8 ms on localhost (1/4)
16/11/13 20:19:58 INFO TaskSetManager: Finished task 0.0 in stage 6.0 (TID 32) in 19 ms on localhost (2/4)
16/11/13 20:19:58 INFO TaskSetManager: Finished task 3.0 in stage 6.0 (TID 35) in 8 ms on localhost (3/4)
16/11/13 20:19:58 INFO TaskSetManager: Finished task 2.0 in stage 6.0 (TID 34) in 9 ms on localhost (4/4)
16/11/13 20:19:58 INFO TaskSchedulerImpl: Removed TaskSet 6.0, whose tasks have all completed, from pool
16/11/13 20:19:58 INFO DAGScheduler: ResultStage 6 (collect at Queries.scala:179) finished in 0.021 s
16/11/13 20:19:58 INFO DAGScheduler: Job 5 finished: collect at Queries.scala:179, took 0.029844 s
93
13
11
143
Compilation completed successfully in 3s 648ms (4 minutes ago)
71 chars, 5 lines  307:1  CRLF  UTF-8
```

(iii)Data Frame Query 3:

Displays the type of movie (horror, comedy, animated, thriller) that is having the highest number of tweets by using data frame queries.

```scala
object Queries{
  def main(args: Array[String]): Unit = {

    System.setProperty("hadoop.home.dir","C:\\Users\\sange\\Downloads\\hadooponwindows-master\\hadooponwindows-master")
    val sparkConf = new SparkConf().setAppName("Queries").setMaster("local[*]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    val lines= sc.textFile("C:\\Users\\sange\\Desktop\\Pavan\\tweets1.txt")



    val df=sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\data2.json")
    df.registerTempTable("dftable")
    val horror=sqlContext.sql("select user.name,user.followers_count from dftable where text LIKE '%horror%'")
    val horrorcount=horror.count()
    println(horrorcount)
    val comedy=sqlContext.sql("select user.name,user.followers_count as count from dftable where text LIKE '%comedy%'")
    val comedycount=comedy.count()
    val thriller=sqlContext.sql("select user.name,user.followers_count as count from dftable where text LIKE '%thriller%'")
    val thrillercount=thriller.count()
    val animated=sqlContext.sql("select user.name,user.followers_count as count from dftable where text LIKE '%animated%'")
    val animatedcount=animated.count()
    if(horrorcount>comedycount){
      if (horrorcount>thrillercount){
        if(horrorcount>animatedcount){
          println("Horror Movies are searched maximum")
        }else{
          println("Animated Movies are searched maximum")
        }
      }else if(thrillercount>animatedcount){
        println("Thriller Movies are searched maximum")
      }
    }else if(comedycount>thrillercount){
      if(comedycount>animatedcount){
        println("Comedy Movies are searched maximum")
      }
    }
```

Output:

```
16/11/13 19:31:57 INFO DAGScheduler: looking for newly runnable stages
16/11/13 19:31:57 INFO DAGScheduler: running: Set()
16/11/13 19:31:57 INFO DAGScheduler: waiting: Set(ResultStage 9)
16/11/13 19:31:57 INFO DAGScheduler: failed: Set()
16/11/13 19:31:57 INFO DAGScheduler: Submitting ResultStage 9 (MapPartitionsRDD[52] at count at Queries.scala:33), which has no missing parents
16/11/13 19:31:57 INFO MemoryStore: Block broadcast_19 stored as values in memory (estimated size 12.5 KB, free 449.1 KB)
16/11/13 19:31:57 INFO MemoryStore: Block broadcast_19_piece0 stored as bytes in memory (estimated size 6.2 KB, free 455.3 KB)
16/11/13 19:31:57 INFO BlockManagerInfo: Added broadcast_19_piece0 in memory on localhost:51774 (size: 6.2 KB, free: 1128.3 MB)
16/11/13 19:31:57 INFO SparkContext: Created broadcast 19 from broadcast at DAGScheduler.scala:1006
16/11/13 19:31:57 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 9 (MapPartitionsRDD[52] at count at Queries.scala:33)
16/11/13 19:31:57 INFO TaskSchedulerImpl: Adding task set 9.0 with 1 tasks
16/11/13 19:31:57 INFO TaskSetManager: Starting task 0.0 in stage 9.0 (TID 35, localhost, partition 0,NODE_LOCAL, 1918 bytes)
16/11/13 19:31:57 INFO Executor: Running task 0.0 in stage 9.0 (TID 35)
16/11/13 19:31:57 INFO ShuffleBlockFetcherIterator: Getting 6 non-empty blocks out of 6 blocks
16/11/13 19:31:57 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
16/11/13 19:31:57 INFO Executor: Finished task 0.0 in stage 9.0 (TID 35). 1959 bytes result sent to driver
16/11/13 19:31:57 INFO TaskSetManager: Finished task 0.0 in stage 9.0 (TID 35) in 15 ms on localhost (1/1)
16/11/13 19:31:57 INFO TaskSchedulerImpl: Removed TaskSet 9.0, whose tasks have all completed, from pool
16/11/13 19:31:57 INFO DAGScheduler: ResultStage 9 (count at Queries.scala:33) finished in 0.015 s
16/11/13 19:31:57 INFO DAGScheduler: Job 4 finished: count at Queries.scala:33, took 1.185779 s
Horror Movies are searched maximum
16/11/13 19:31:57 INFO JSONRelation: Listing file:/C:/Users/sange/Desktop/Pavan/hashtags.txt on driver
16/11/13 19:31:57 INFO MemoryStore: Block broadcast_20 stored as values in memory (estimated size 127.9 KB, free 583.2 KB)
16/11/13 19:31:57 INFO MemoryStore: Block broadcast_20_piece0 stored as bytes in memory (estimated size 14.0 KB, free 597.2 KB)
16/11/13 19:31:57 INFO BlockManagerInfo: Added broadcast_20_piece0 in memory on localhost:51774 (size: 14.0 KB, free: 1128.3 MB)
16/11/13 19:31:57 INFO SparkContext: Created broadcast 20 from json at Queries.scala:101
16/11/13 19:31:57 INFO FileInputFormat: Total input paths to process : 1
16/11/13 19:31:57 INFO SparkContext: Starting job: json at Queries.scala:101
16/11/13 19:31:57 INFO DAGScheduler: Got job 5 (json at Queries.scala:101) with 2 output partitions
16/11/13 19:31:57 INFO DAGScheduler: Final stage: ResultStage 10 (json at Queries.scala:101)
16/11/13 19:31:57 INFO DAGScheduler: Parents of final stage: List()
```

(iv)Data Frame Query 4:

Counts the number of users who created their twitter accounts and prints the output as count according to the quarters. Where first quarter: January to March, second quarter: April to June, third quarter: July to September, fourth quarter: October to December.

```scala
object Queries{
  def main(args: Array[String]): Unit = {

    System.setProperty("hadoop.home.dir","C:\\Users\\sange\\Downloads\\hadooponwindows-master\\hadooponwindows-master")
    val sparkConf = new SparkConf().setAppName("Queries").setMaster("local[*]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    val lines= sc.textFile("C:\\Users\\sange\\Desktop\\Pavan\\tweets1.txt")
    val df=sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\data2.json")
    df.registerTempTable("dftable")
    val jan=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Jan%' group by user")
    val feb=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Feb%' group by user")
    val mar=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Mar%' group by user")
    val apr=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Apr%' group by user")
    val may=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%May%' group by user")
    val jun=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Jun%' group by user")
    val jul=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Jul%' group by user")
    val aug=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Aug%' group by user")
    val sep=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Sep%' group by user")
    val oct=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Oct%' group by user")
    val nov=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Nov%' group by user")
    val dec=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Dec%' group by user")
    val jancount=jan.count()
    val febcount=feb.count()
    val marcount=mar.count()
    val aprcount=apr.count()
```

```scala
    val aug=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Aug%' group by user")
    val sep=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Sep%' group by user")
    val oct=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Oct%' group by user")
    val nov=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Nov%' group by user")
    val dec=sqlContext.sql("select user.name, SUBSTRING(user.created_at,5,3) AS month,SUBSTRING(user.created_at,12,8) AS time," +
      "Count(*) as count from dftable where SUBSTRING(user.created_at,5,3) like '%Dec%' group by user")
    val jancount=jan.count()
    val febcount=feb.count()
    val marcount=mar.count()
    val aprcount=apr.count()
    val maycount=may.count()
    val juncount=jun.count()
    val julcount=jul.count()
    val augcount=aug.count()
    val sepcount=sep.count()
    val octcount=oct.count()
    val novcount=nov.count()
    val deccount=dec.count()


    val firstquarter=jancount+febcount+marcount
    val secondquarter=aprcount+maycount+juncount
    val thirdquarter=julcount+augcount+sepcount
    val fourthquarter=octcount+novcount+deccount
    println("First Quarter Of Users %s" .format(firstquarter))
    println("Second Quarter Of Users %s" .format(secondquarter))
    println("Third Quarter Of Users %s" .format(thirdquarter))
    println("Fourth Quarter Of Users %s" .format(fourthquarter))
```

Output:

(v)Hashtags Query 5:

Considers the both hashtag file and text file, prints the hashtags that are in common.

```scala
//import org.apache.spark.sql._
/**...*/


object Queries{
  def main(args: Array[String]): Unit = {

    System.setProperty("hadoop.home.dir","C:\\Users\\sange\\Downloads\\hadooponwindows-master\\hadooponwindows-master")
    val sparkConf = new SparkConf().setAppName("Queries").setMaster("local[*]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    val lines= sc.textFile("C:\\Users\\sange\\Desktop\\Pavan\\tweets1.txt")
    val df=sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\data2.json")
//    df.registerTempTable("dftable")
    df.createOrReplaceTempView("dftable")
    val hashtags= sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\hashtags.txt")

    val hashtagdf=hashtags.toDF().withColumnRenamed("_corrupt_record","hashfiles")

    hashtagdf.createOrReplaceTempView("dftab")

    val hashtagsquery=sqlContext.sql("select t.text as text,d.hashfiles as hashtags from dftable t JOIN dftab d on t.text like" +
      " CONCAT('%',d.hashfiles,'%')")

    hashtagsquery.show()
    println("Printing the Common hashtags data of Hastags Data and Our Twitter Data")


```

OutPut:

```
+--------------------+-------------+
|                text|     hashtags|
+--------------------+-------------+
|RT @radicccaaal: ...|         Bros|
|RT @radicccaaal: ...|Pierce Brosnan|
|RT @TheOnion: Mik...|   Mike Pence|
|Who says #Starwar...|    #RogueOne|
|top 3 movies #ASK...| #ASKCHRISTIAN|
|RT @TheOnion: Mik...|   Mike Pence|
|The Devil's Backb...|    Guillermo|
|007 WKND #onair T...|       Thomas|
|What are your fav...| #ASKCHRISTIAN|
|@ChristianLeave  ...| #ASKCHRISTIAN|
|RT @MarcAThomass:...|       Thomas|
|Best Pierce Brosn...|         Bros|
|Best Pierce Brosn...|Pierce Brosnan|
|You know in movie...|      Vikings|
|You know in movie...|      Vikings|
|You know in movie...|      Vikings|
|RT @Ethan_Booker:...|       Booker|
|#DVD #Movies Safe...|       #Julia|
|RT @dancintina: T...|   Guillermo|
|RT @dancintina: T...|   Guillermo|
+--------------------+-------------+
only showing top 20 rows

Printing the Common hashtags data of Hastags Data and Our Twitter Data
```

Compilation completed successfully with 1 warning in 11s 976ms (9 minutes ago)

(vi)Twitter API Query 6:

Screen names of the user are found by taking the maximum followers count in descending order and giving the screen name to the url of api.twitter.com and calling the API. API retrieves the account details and give the data of statuses of the particular screen name(user) that is given.

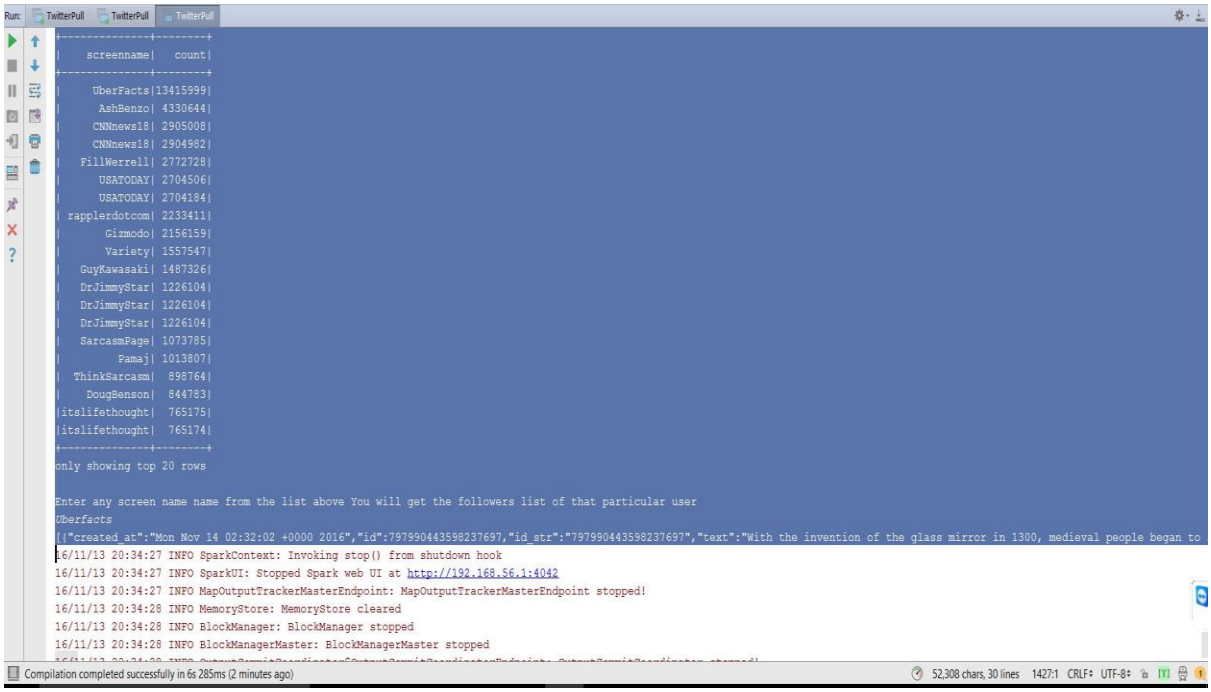
```
build.sbt ×   Queries.scala ×   TwitterPull.scala ×

object TwitterPull{


  def main(args: Array[String]) {
    val AccessToken = "790254444306833408-VUXVM722Hj6CdIUaCycBFzKbA3WGnmO";
    val AccessSecret = "kHDXSiv3ZTpS3h966nlnZkbd5dZzBJtl49aU5r4sLTpZ2";
    val ConsumerKey = "71GFiDzYipT3O3rVRTQOuQMiT";
    val ConsumerSecret = "8YnMMwT1NxQEHLnyrlExYiGMamVfbDYR8fy219JovwpMaXvicy";

    val consumer = new CommonsHttpOAuthConsumer(ConsumerKey, ConsumerSecret);
    consumer.setTokenWithSecret(AccessToken, AccessSecret);
    System.setProperty("hadoop.home.dir","C:\\Users\\sange\\Downloads\\hadooponwindows-master\\hadooponwindows-master")
    val sparkConf = new SparkConf().setAppName("TwitterPull").setMaster("local[*]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    val df=sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\data2.json")
    df.registerTempTable("dftable")
    println("The Users with their followers count in Descending Order")
    val active=sqlContext.sql("select user.screen_name as screenname, max(user.followers_count) as count from dftable where user.follower
      active.show()
    println("Enter any screen name name from the list above You will get the followers list of that particular user")
    val screenname=scala.io.StdIn.readLine()
    val request = new HttpGet("https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name="+screenname);
    consumer.sign(request);
    val client = new DefaultHttpClient();
    val response = client.execute(request);
    println(IOUtils.toString(response.getEntity().getContent()));
  }
}
```

Output:



VII. References

https://www.supergloo.com/fieldnotes/apache-spark-examples-of-transformations/

http://public-repo-1.hortonworks.com/hdp-win-alpha/winutils.exe

https://trongkhoanguyenblog.wordpress.com/2014/11/27/understand-rdd-operations-transformations-and-actions/

http://stackoverflow.com/questions/34047332/unresolved-dependency-issue-when-compiling-spark-project-with-sbt

https://mvnrepository.com/artifact/org.apache.spark/spark-core_2.10

https://github.com/apache/spark/blob/master/sql/core/src/main/scala/org/apache/spark/sql/SQLContext.scala

https://www.infoq.com/articles/apache-spark-sql

https://github.com/apache/spark/blob/master/sql/core/src/main/scala/org/apache/spark/sql/functions.scala

https://github.com/apache/spark/tree/master/sql/core/src/test/scala/org/apache/spark/sql

http://spark.apache.org/docs/latest/sql-programming-guide.html

http://www.agildata.com/apache-spark-rdd-vs-dataframe-vs-dataset/

https://mvnrepository.com/artifact/org.apache.spark

https://dzone.com/articles/access-twitter-rest-api-v11