**Spark Project**

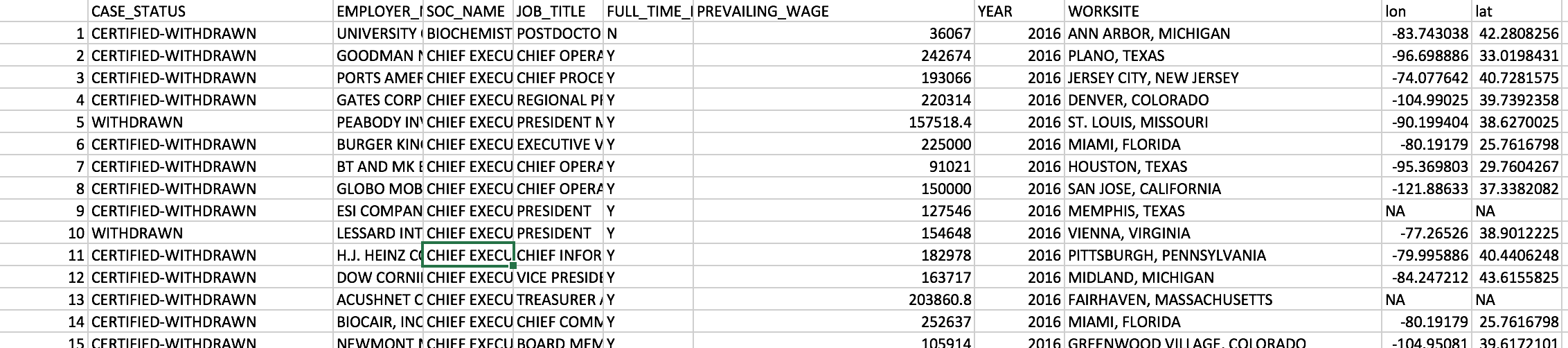
**Introduction**

This is a simple job to do analysis on H1B data for the years of 2011 to 2016, extracting the number of certified H1B petitions across all areas in the United States. The job is written in Scala for the spark cluster computing platform.

**Data Source**

This H1B data set was downloaded <https://www.kaggle.com/datasets>.

Below is a snippet of the data. Also, find the link to the entire source, listed below.



Link for data source:

https://drive.google.com/open?id=0B61JmW3QIOuQSmJBbFpkWk5iMU0

**Data Source Description**

This data source is a repository of H1B petitions filed over the past 5 years from 2011-2016 along with the status.The data is in csv format. It consists of 11 columns with headers and column values.

**Schema**

Case\_Status

Employer\_Name

SOC\_Name

Job\_Title

Full\_Time\_Position

Prevailing\_Wage

Year

Worksite

Longitude

Latitude

**Data Load**

The csv file was loaded into the local file system of the Hadoop cluster.

**Spark Algorithm**

**Step1:** Start spark in the local mode also loading the databricks package to help load the csv data file.

**Step 2:** Read the csv file as a dataframe and assign the result to df.

**Step 3:** Convert the dataframe into an RDD with the columns,CASE\_STATUS,YEAR and WORKSITE and assign the result to rddnew

**Step 4:** Filter out the petitions with CASE\_STATUS “CERTIFIED” and assign the result to rddnew1.

**Step 5:** Append the value “1” to each of the rows,making CASE\_STATUS and YEAR the key and “1” the value and assign the result to rddnew2.

**Step 6:** ReduceByKey mapOut using the sum reducer, and assign the result to the value “reduceOut”.

**Step 7:** Swap the key and value for each reduceOut tuple by using the map function and the .swap() method, and assign the result to the value “swap”.

**Step 8:** Sort swap by key, in descending order, and assign the result to the value “sort”.

**Step 9:** Print the top 100 tuples of the result.

**Output Description**

The output consists of the number of certified H1B petitions as the key and the state and year as the corresponding values.

**Code**

**>spark-shell --master local --packages com.databricks:spark-csv\_2.10:1.5.**

**>val df = sqlContext.read.format("com.databricks.spark.csv").option("header",** "true").load("file:/home/raje2904/H1B/h1b\_kaggle.csv")

df: org.apache.spark.sql.DataFrame = [: string, CASE\_STATUS: string, EMPLOYER\_NA ME: string, SOC\_NAME: string, JOB\_TITLE: string, FULL\_TIME\_POSITION: string, PRE VAILING\_WAGE: string, YEAR: string, WORKSITE: string, lon: string, lat: string]

**>val rddnew=df.map(row => (row(1),row(4),row(6),row(7),row(8)))**

rddnew: org.apache.spark.rdd.RDD[(Any, Any, Any, Any, Any)] = MapPartitionsRDD[9] at map at <console>:27

**>val rddnew1=rddnew.filter(\_.\_1 == "CERTIFIED")**

rddnew1: org.apache.spark.rdd.RDD[(Any, Any, Any, Any, Any)] = MapPartitionsRDD[10] at filter at <console>:29

**>val rddnew2=rddnew1.map(row$1 =>((row$1.\_5,row$1.\_4),1))**

rddnew2: org.apache.spark.rdd.RDD[((Any, Any), Int)] = MapPartitionsRDD[11] at map at <console>:31

**>val reduceOut = rddnew2.reduceByKey((a,b) => (a+b))**

reduceOut: org.apache.spark.rdd.RDD[((Any, Any), Int)] = ShuffledRDD[12] at reduceByKey at <console>:33

**>val swap = reduceOut.map(reduceOut => reduceOut.swap)**

swap: org.apache.spark.rdd.RDD[(Int, (Any, Any))] = MapPartitionsRDD[13] at map at <console>:35

**>val sort = swap.sortByKey(false)**

sort: org.apache.spark.rdd.RDD[(Int, (Any, Any))] = ShuffledRDD[16] at sortByKey at <console>:37

**>sort.take(100).foreach(println)**

**Output**

(34639,(NEW YORK, NEW YORK,2016))

(31266,(NEW YORK, NEW YORK,2015))

(27634,(NEW YORK, NEW YORK,2014))

(23737,(NEW YORK, NEW YORK,2012))

(23537,(NEW YORK, NEW YORK,2013))

(23172,(NEW YORK, NEW YORK,2011))

(15242,(HOUSTON, TEXAS,2015))

(13836,(SAN FRANCISCO, CALIFORNIA,2016))

(13655,(HOUSTON, TEXAS,2016))

(13360,(HOUSTON, TEXAS,2014))

(12594,(SAN FRANCISCO, CALIFORNIA,2015))

(11678,(ATLANTA, GEORGIA,2016))

(11136,(HOUSTON, TEXAS,2013))

(11064,(CHICAGO, ILLINOIS,2016))

(10500,(ATLANTA, GEORGIA,2015))

(9963,(HOUSTON, TEXAS,2012))

(9798,(SAN FRANCISCO, CALIFORNIA,2014))

(9642,(SAN JOSE, CALIFORNIA,2016))

(9589,(SAN JOSE, CALIFORNIA,2015))

(9239,(CHICAGO, ILLINOIS,2015))

(8223,(SAN JOSE, CALIFORNIA,2014))

(8213,(ATLANTA, GEORGIA,2014))

(8184,(HOUSTON, TEXAS,2011))

(7767,(CHICAGO, ILLINOIS,2014))

(7502,(SUNNYVALE, CALIFORNIA,2015))

(7286,(IRVING, TEXAS,2016))

(7281,(SAN FRANCISCO, CALIFORNIA,2013))

(7227,(SUNNYVALE, CALIFORNIA,2016))

(6954,(CHARLOTTE, NORTH CAROLINA,2016))

(6722,(SAN JOSE, CALIFORNIA,2013))

(6509,(CHARLOTTE, NORTH CAROLINA,2015))

(6501,(DALLAS, TEXAS,2016))

(6377,(ATLANTA, GEORGIA,2013))

(6320,(CHICAGO, ILLINOIS,2013))

(6141,(REDMOND, WASHINGTON,2016))

(6120,(IRVING, TEXAS,2015))

(6116,(SAN FRANCISCO, CALIFORNIA,2012))

(6039,(DALLAS, TEXAS,2015))

(5906,(BOSTON, MASSACHUSETTS,2016))

(5778,(SEATTLE, WASHINGTON,2016))

(5712,(SUNNYVALE, CALIFORNIA,2014))

(5671,(CHICAGO, ILLINOIS,2012))

(5607,(AUSTIN, TEXAS,2016))

(5589,(MOUNTAIN VIEW, CALIFORNIA,2016))

(5565,(ATLANTA, GEORGIA,2012))

(5533,(JERSEY CITY, NEW JERSEY,2015))

(5438,(REDMOND, WASHINGTON,2015))

(5416,(SAN JOSE, CALIFORNIA,2012))

(5304,(DALLAS, TEXAS,2014))

(5235,(SEATTLE, WASHINGTON,2015))

(5207,(AUSTIN, TEXAS,2015))

Ws2ed3rf4!

(5192,(BOSTON, MASSACHUSETTS,2015))

(5188,(CHICAGO, ILLINOIS,2011))

(5126,(PHOENIX, ARIZONA,2016))

(5113,(PHILADELPHIA, PENNSYLVANIA,2016))

(5104,(MOUNTAIN VIEW, CALIFORNIA,2015))

(5056,(PLANO, TEXAS,2016))

(5011,(SANTA CLARA, CALIFORNIA,2015))

(4883,(JERSEY CITY, NEW JERSEY,2016))

(4883,(SANTA CLARA, CALIFORNIA,2016))

(4713,(SAN JOSE, CALIFORNIA,2011))

(4711,(SAN FRANCISCO, CALIFORNIA,2011))

(4709,(LOS ANGELES, CALIFORNIA,2016))

(4612,(CHARLOTTE, NORTH CAROLINA,2014))

(4555,(PHOENIX, ARIZONA,2015))

(4544,(ATLANTA, GEORGIA,2011))

(4518,(CHARLOTTE, NORTH CAROLINA,2013))

(4436,(ALPHARETTA, GEORGIA,2016))

(4408,(BOSTON, MASSACHUSETTS,2014))

(4389,(MOUNTAIN VIEW, CALIFORNIA,2014))

(4389,(JERSEY CITY, NEW JERSEY,2014))

(4384,(SANTA CLARA, CALIFORNIA,2014))

(4383,(REDMOND, WASHINGTON,2014))

(4345,(IRVING, TEXAS,2014))

(4342,(SAN DIEGO, CALIFORNIA,2013))

(4311,(PHILADELPHIA, PENNSYLVANIA,2015))

(4272,(LOS ANGELES, CALIFORNIA,2015))

(4260,(SEATTLE, WASHINGTON,2014))

(4255,(SAN DIEGO, CALIFORNIA,2015))

(4181,(SAN DIEGO, CALIFORNIA,2012))

(4162,(BELLEVUE, WASHINGTON,2016))

(4149,(SUNNYVALE, CALIFORNIA,2013))

(4021,(ALPHARETTA, GEORGIA,2015))

(3998,(SAN DIEGO, CALIFORNIA,2014))

(3956,(SAN DIEGO, CALIFORNIA,2016))

(3955,(AUSTIN, TEXAS,2014))

(3944,(REDMOND, WASHINGTON,2013))

(3943,(PLANO, TEXAS,2015))

(3913,(BELLEVUE, WASHINGTON,2015))

(3850,(BOSTON, MASSACHUSETTS,2012))

(3829,(BOSTON, MASSACHUSETTS,2013))

(3808,(REDMOND, WASHINGTON,2012))

(3791,(SANTA CLARA, CALIFORNIA,2013))

(3749,(PHILADELPHIA, PENNSYLVANIA,2014))

(3722,(SEATTLE, WASHINGTON,2013))

(3705,(REDMOND, WASHINGTON,2011))

(3651,(BOSTON, MASSACHUSETTS,2011))

(3646,(DALLAS, TEXAS,2013))

(3586,(LOS ANGELES, CALIFORNIA,2012))

(3585,(LOS ANGELES, CALIFORNIA,2014))

**Output Verification**

I took a few lines of the data and verified it against the code.

**Perfromance/Scale Characterisitcs**

The data source is around 400mb in size.I computed the time roughly for my code against this size of data and it took nearly 12 seconds.

I added another 600mb of data and the time taken for my code to run against this 1gb of data was nearly around the same.

**What would I have done differently?**

Since this is a csv file ,I would have tried to map the column header as key and the column value as the value.This would create an Array of Maps from which I would be able to extract the certified petitions.Later convert this rdd to a data frame and find the count of the certified petitions across the states.

**Algorithm for a different approach**

**Step 1:** Start the spark shell in local mode.

**Step 2:** Load the csv file and assign the result to a.

**Step 3:** Split the file based on “,” and assign this to b.

**Step 4:** Extract the column headers from b and assign this to c.

**Step 5:** Extract the column values from b and assign this to d.

**Step 6:** Using the map function ,map the column header to its column value,with the header as key and value as value.

**Step 7:** Filter the petitions with CASE\_STATUS “CERTIFIED”

**Step 8:** Convert this rdd to a data frame.

**Step 9:** Group by the WORKSITE.

**Step 10:** Perfrom a count operation which will give the number of certified petitions in each area.

**Conclusion**

Spark helped to simplify the challenging and compute-intensive task of processing high volumes of real-time or archived data, both structured(in this case) and unstructured.