Lab on Spark

Apache Spark is used for large-scale data processing and allows the users to easily execute algorithms on clusters of machines. The system partitions the data across the cluster and use the computational power and main memory of the machines to perform parallel computation.

For more details about the concepts behind Apache Spark, you can read the original research article: `http://www-bcf.usc.edu/~minlanyu/teach/csci599-fall12/papers/nsdi_spark.pdf`.

Using Databricks

Databricks is a company founded by the creators of Spark that aims to help clients with cloud-based big data processing using Spark. You can create an account for free at `https://databricks.com/`, which allows you to use one machine in their cluster. The very same code can be executed in large clusters of machines (e.g. 1000 machines). Go to their website: `https://databricks.com/` and create a community edition account. You can use Spark with Python, Java, or Scala. We recommend you to use Python for this lab.

After creating a Databricks account, you need to create a cluster using the tab *Clusters* $\rightarrow$ *Create Cluster*. After that, you will see the cluster in the Active Cluster. For the following steps create a notebook, tab *databricks* $\rightarrow$ *New* $\rightarrow$ *Notebook*. You will be able to use a notebook similar to Jupyter/iPython.

MapReduce functions in Spark

Datasets. You are provided with one dataset *steve.txt* with the content of the Wikipedia page of Steve Jobs. You are also provided with two more files containing the simple English version of Wikipedia (`http://simple.wikipedia.org`) represented as a directed graph. Such a dataset consists of a set of Wikipedia pages (*idslabels.txt*) and a set of links (*edgelist.txt*) where every line is of the kind:

$A\ B1\ B2 \ldots Bn$

where A is the *id* of a web page, $B1, \ldots, Bn$ are the pages A links to.

In the file *idslabels.txt* every line contains an identifier and the page associated with the identifier. Beginner users should only focus on the first dataset: "steve.txt".

Load all the files, *idslabels.txt*, *edgelist.txt*, and *steve.txt* to the HDFS, by using tab *Data* $\rightarrow$ *Tables* $\rightarrow$ *Create Table*, and remember the paths where the files are stored. The format of the paths will be similar to : */FileStore/tables/path/idslabels.txt*, */FileStore/tables/path/edgelist.txt*, */FileStore/tables/path/edgelist.txt*

lambda functions A lambda function is a compact way to define a function in Python. It is useful when the argument of a function is a function itself. For example, consider the following:

```
def double(a):
        return 2*a

map(double,[1,2,3,4])
```

The function map receives two arguments, a function and a list. It executes the function on every element in the list and outputs the results. In the previous code, the function double is executed for every element in the list. As a result, one would get $[2, 4, 6, 8]$. The same code can be written in a more compact way using lambda functions.

```
map(lambda a: 2*a, [1,2,3,4])
```

which outputs the same results. Notice that *lambda a: 2*a* is equivalent to *def double(a): return 2a* (except that the function created with lambda is anonymous). We recommend you to practice with lambda functions. E.g. write a code in Python which adds 1 to every element in a list using lambda functions.

Writing an application. The first step when writing a Spark application is to create an object of the type *SparkContext*. This represents the connection to a Spark cluster and can be used to create several types of other objects, such as resilient distributed datasets (RDD). In Databricks, there exists already a SparkContext object *sc* and it shouldn't be redefined. An *RDD* is a dataset partitioned across the nodes of the cluster, which can be operated on in parallel. In order to load into memory an RDD, we use the instruction

```
lines = sc.textFile("/FileStore/tables/path/steve.txt")
```

There are several operations that can be performed on an RDD, such as map, flatMap, reduce, reduceByKey, filter, take and so on. Please take a moment to read the official documentation available at `https://spark.apache.org/docs/latest/programming-guide.html#rdd-operations` for each of those operations. Be sure to check the Python explanation.

The following code prints the first paragraph of the file "steve.txt". Run such a code and study the results.

```
text_file = sc.textFile("/FileStore/tables/path/steve.txt")
print(lines.take(5))
```

We now use the function flatMap to split the text into a list of words.

```
words = text_file.flatMap(lambda line: line.split(" "))
for a in words.take(5):
  print a
```

The map function is specified by using the function map on an RDD object. This is similar to the map function we saw earlier in the text. For example the following returns a list of key value pairs, where the word is the key and 2 is the value.

```
pairs = words.map(lambda s: (s, 2))
```

Observe, we used the lambda function, however, this is not necessary. The reduce function can be specified by reduceByKey, which receives an input a function which is associative and commutative (e.g. the sum).

```
counts = pairs.reduceByKey(lambda a, b: a + b)
for (count, word) in counts.take(5):
  print "%s: %i" % (count, word)
```

The previous code outputs 5 words in the text together with a count number. What are we exactly counting?

Exercises (Beginner)

You should consider only the file "steve.txt".

Exercise 1: After understanding the previous text, write a code in Python which computes the number of occurrences for each word in the file "steve.txt" on an RDD object. You just need to do cut and paste of the previous code and make one simple change. Alternatively, you can change the file skeleton which is provided to you.

Exercise 2: Write a code in Python that shows the top 5 words with largest number of occurrences. For this exercise, you can use the function sortByKey on an RDD object. This sorts the key value pairs by key (a boolean argument specifies whether it is ascending or descending order).

Exercise (Intermediate/Advanced)

Exercise 3: Write a code in Python that shows the top 5 words with largest number of occurrences among the words containing at least 5 characters. For this exercise, you can use the function filter on an RDD object.

Exercise 4: Write a code in Python that lists the top 10 pages in Wikipedia with largest in-degree. For this exercise use the files *idslabels.txt* and *edgelist.txt*. We recommend you to use the function *join()* of an RDD object to retrieve the names of the pages. The function *collect()* might also be useful for this exercise. A documentation of these functions and other that might be useful is available at: at `https://spark.apache.org/docs/latest/programming-guide.html#rdd-operations`. For advanced users only.

Exercise 5: Write a code in Python that computes the PageRank vector of the Web graph representing the Wikipedia pages. In such a graph G, the nodes represent Web pages, while directed edges represent hyperlinks. We assume there are n pages. Do the following steps:

1. Compute the matrix M_G (M for short) associated to the Web graph G.
2. Run the PageRank algorithm until the L_1 norm of $r^{t-1} - r^t$ is at most 0.1, where r^t is the PageRank vector at step t.

To this end, you can use CoordinateMatrix and BlockMatrix, see `https://spark.apache.org/docs/2.0.2/mllib-data-types.html` that supports multiplication between distributed sparse matrices. You can also check this tutorial `https://stackoverflow.com/questions/46901755/multiply-2-sparse-matrices` `https://spark.apache.org/docs/2.0.2/mllib-data-types.html`.

What is the most important page in the graph obtained from Wikipedia?

For advanced users only.