

DSSP

Apache Spark Lab

Apostolos N. Papadopoulos

Preliminaries (I)

Login to host

master-bigdata.polytechnique.fr

Windows users: **use PuTTY**

Linux – Mac: **use ssh**

ssh `username@master-bigdata.polytechnique.fr`

(same username you used in the previous lecture)

Preliminaries (II)

The cluster is composed of 32 machines.

One machine is the master. The other 31 machines are slaves.

Software:

- Hadoop 2.7.3
- Yarn
- Spark 2.1.0

Preliminaries (III)

Go to your home folder at master:

```
cd ~
```

Copy the file **sparklab.tar.gz** from **/opt** to your home folder at master:

```
cp /opt/sparklab.tar.gz .
```

Preliminaries (IV)

Extract the compressed file:

```
tar xvf sparklab.tar.gz
```

You should see the folder **sparklab**. Enter the folder:

```
cd sparklab
```

List the files in this folder:

```
ls -las
```

Executing Spark Jobs (I)

The basic tool to execute spark jobs is the **spark-submit** tool

Try this:

```
spark-submit --master local[2] sort2.py 100
```

local[2] means that we are executing the job in one machine only (the master) and we are using 2 cores. Very useful for debugging!

Executing Spark Jobs (II)

To execute the job in a distributed manner using the cluster we should say:

```
spark-submit --master yarn sort2.py 1000
```

Executing Spark Jobs (III)

Control the number of workers (executors). Recall, each executor is a separate JVM.

```
spark-submit --master yarn --num-executors 4 sort2.py 1000
```


Other Examples

Find the frequency of every word in a text:

```
spark-submit --master yarn --conf  
    spark.local.dir=/home/your_username/tmp wordcount.py  
    /dssp/data/leonardo/leonardo.txt
```

Compute the PageRank:

```
spark-submit --master yarn pagerank.py /dssp/data/graph.txt 10
```

Program Anatomy

```
from __future__ import print_function
import sys
from operator import add
from pyspark import SparkContext

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: wordcount <file>", file=sys.stderr)
        exit(-1)

    sc = SparkContext(appName="PythonWordCount")
    lines = sc.textFile(sys.argv[1], 1)
    counts = lines.flatMap(lambda x: x.split(' ')) \
                   .map(lambda x: (x, 1)) \
                   .reduceByKey(add)

    output = counts.collect()
    for (word, count) in output:
        print("%s: %i" % (word, count))

    sc.stop()
```

WordCount

Program Anatomy

```
from __future__ import print_function
import sys
import numpy as np
from pyspark import SparkContext
from pyspark.mllib.clustering import KMeans
```

```
def parseVector(line):
    return np.array([float(x) for x in line.split(' ')])

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: kmeans <file> <k>", file=sys.stderr)
        exit(-1)

    sc = SparkContext(appName="KMeans")
    lines = sc.textFile(sys.argv[1])
    data = lines.map(parseVector)
    k = int(sys.argv[2])
    model = KMeans.train(data, k)
    print("Final centers: " + str(model.clusterCenters))
    print("Total Cost: " + str(model.computeCost(data)))
    sc.stop()
```

k-Means