Lab 11 Spark

ISTD, SUTD

Jul 8, 2021

Learning outcome

By the end of this lesson, you are able to

- Submit PySpark jobs to a Spark cluster
- Paralelize data processing using PySpark

Submit a PySpark job to a Spark cluster

1. Using FlintRock, launch a Hadoop-Spark cluster with 4 nodes (1 name node and 3 worker/data nodes). For the ease of subsequence exercise, you might considering using t2.medium instance instead of t2.micro. If you want to stick to t2.micro, you should at least enable swap on the nodes. (For details, refer to lab10)

2. On the Spark cluster master node, check out the following github repo.

```
$ cd
$ mkdir git
$ cd git
$ git clone https://github.com/istd50043-2021-fall/50043-labs/
```

if it exists, update

```
$ cd git
$ git pull
```

3. Change to the `50043-labs` directory

4. Copy some data into HDFS `/input/` if it is empty, we re-use the data from lab10.

```
$ hdfs dfs -mkdir /input/
$ hdfs dfs -rm -r /output
$ hdfs dfs -put ~/git/50043-labs/lab10/data/TheCompleteSherlockHolmes.txt /input/
```

5. Edit the PySpark source code in `lab11/wordcount.py` so that it uses the correct HDFS address.

6. Start the job

If you have turned on Yarn.

```
$ spark-submit --master yarn lab11/wordcount.py
```

Otherwise

```
$ spark-submit --master spark://<internal-ip-of-spark-master>:7077 lab11/wordcount.py
```

7. Check the output

```
$ hdfs dfs -cat /output/*
```

Exercise 1

Write a PySpark application which takes a (set of) Comma-seperated-value(CSV) file(s) with 2 columns and output a CSV file with first two columns same as the input file, and the third column captures the values of split the values from the first column by using the value from the second column as delimiter.

For example, given input from a HDFS file

```
50000.0#0#0#,#
0@1000.0@,@
1$,$
1000.00^Test_string,^
```

the program should output

```
0@1000.0@,@,[u'0', u'1000.0', u'']
1$,$,[u'1', u'']
1000.00^Test_string,^,[u'1000.00', u'Test_string']
```

back to the HDFS

Exercise 2

Write PySpark application which aggregates a (set of) CSV file(s) with 4 columns based on its third column, (the destination IP).

Given input

```
05:49:56.604899, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604900, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604899, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604900, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604899, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604900, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604899, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604900, 10.0.0.2.54880, 10.0.0.3.5001, 2
```

```
05:49:56.604899, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604900, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604899, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604900, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604899, 10.0.0.2.54880, 10.0.0.3.5001, 2
05:49:56.604908, 10.0.0.3.5001, 10.0.0.2.54880, 2
05:49:56.604908, 10.0.0.3.5001, 10.0.0.2.54880, 2
05:49:56.604908, 10.0.0.3.5001, 10.0.0.2.54880, 2
05:49:56.604908, 10.0.0.3.5001, 10.0.0.2.54880, 2
05:49:56.604908, 10.0.0.3.5001, 10.0.0.2.54880, 2
05:49:56.604908, 10.0.0.3.5001, 10.0.0.2.54880, 2
05:49:56.604908, 10.0.0.3.5001, 10.0.0.2.54880, 2
```

the program should output

```
 10.0.0.3.5001,13
 10.0.0.2.54880,7
```

Exercise 3

Given the same input as the Exercise 2, write a PySpark application which outputs the following

```
05:49:56.604899,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604900,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604899,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604900,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604899,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604900,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604899,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604900,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604899,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604900,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604899,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604900,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604899,10.0.0.2.54880, 10.0.0.3.5001, 2, 13
05:49:56.604908, 10.0.0.3.5001,10.0.0.2.54880, 2, 7
05:49:56.604908, 10.0.0.3.5001,10.0.0.2.54880, 2, 7
05:49:56.604908, 10.0.0.3.5001,10.0.0.2.54880, 2, 7
05:49:56.604908, 10.0.0.3.5001,10.0.0.2.54880, 2, 7
05:49:56.604908, 10.0.0.3.5001,10.0.0.2.54880, 2, 7
05:49:56.604908, 10.0.0.3.5001,10.0.0.2.54880, 2, 7
05:49:56.604908, 10.0.0.3.5001,10.0.0.2.54880, 2, 7
```

In the event the input is very huge with too many unique destination IP values, can your program scale?

The questions were adopted from

`https://jaceklaskowski.github.io/spark-workshop/exercises/`