Illinois Data Science Initiative

Technical Reports

Setting up Spark for Python

Author:
Sameet Sapra & Joshua Chang & Professor Brunner

February 25, 2017

Setting up Spark for Python

PROFESSOR BRUNNER

Compiled February 25, 2017

An introduction to setting up Apache Spark with Python

https://github.com/lcdm-uiuc

1. WHAT IS APACHE SPARK?

Apache Spark is a fast cluster computing system. Spark can be configured with multiple cluster managers like Yarn, Mesos, Amazon EC2, or standalone mode. This technical report will install Spark on top of Yarn because Hadoop MapReduce is a feature that ships with Yarn.

Spark supports many high level tools like GraphX and MLlib, for graphs processing and machine learning. It also has many APIs in Java, Scala, and Python, and we will install and configure the Java APIs and PySpark, Python's API for Spark.

2. WHY PYSPARK?

PySpark is an API that interfaces with RDD's in Python. It is built on top of Spark's Java API and exposes the Spark programming model to Python. PySpark makes use of a library called Py4J, which enables Python programs to dynamically access Java objects in a Java Virtual Machine. This allows data to be processed in Python and cached in the JVM.

3. PYSPARK PREREQUISITES

Assume that the environment is CentOS. While many Spark tutorials will cover installation of Java, Scala, Python, or any other API languages, we can verify that Java and Yarn are installed by checking the versions. If they are installed, skip to the Spark configuration steps, otherwise follow the installation instructions given below.

4. INSTALLATION: HADOOP WITH YARN

You can install Java on the system with:

```
> sudo yum install java-1.7.0-openjdk-devel
```

Next we will install Hadoop, since Yarn ships with it. To install Hadoop, we will create a Hadoop user account, install Hadoop, and set environment variables:

```
> useradd -d /opt/hadoop hadoop
> passwd hadoop
> curl -O http://apache.javapipe.com/hadoop/
    common/hadoop-2.7.2/hadoop-2.7.2.tar.gz
> tar xfz hadoop-2.7.2.tar.gz
> cp -rf hadoop-2.7.2/* /opt/hadoop/
> chown -R hadoop:hadoop /opt/hadoop/
> su - hadoop
```

Open up the *.bashrc* file and set the following environment variables:

```
> vi .bashrc

export HDP=/usr/hdp/current/hadoop-mapreduce-
    client
export STREAMING=/usr/hdp/current/hadoop-
    mapreduce-client/hadoop-streaming.jar
export EXAMPLES=/usr/hdp/current/hadoop-
    mapreduce-client/hadoop-mapreduce-examples
    .jar
export MAP=yarn
```

Exit out and reload the bashrc with:

```
> source ~/.bash_profile
```

To test that it is running correctly, try creating a random directory in the HDFS file system:

```
> hdfs dfs -mkdir /my_storage
```

Exit out of the hadoop user role. Now we will move onto installing Spark.

5. INSTALLATION: SPARK

Install Spark on the system by downloading the rpm and moving it to the correct location in your path environment variable. Here we will install spark-1.6:

```
> wget http://apache.mirrors.ionfish.org/spark
    /spark-1.6.0/spark-1.6.0-bin-hadoop2.6.tgz
> tar -zxvf spark-1.6.0-bin-hadoop2.6.tgz
> mv spark-1.6.0-bin-hadoop2.6 spark
```

Let's configure Spark to only show errors on startup, rather than all info. This will make the Spark shell less cluttered when it is opened on startup.

```
> cd /usr/hdp/2.3.6.0-3796/spark/conf
> vi log4j.properties
```

Find a line that describe the rootCategory of log:

```
log4j.rootCategory=INFO, console
```

Replace INFO with ERROR:

```
log4j.rootCategory=ERROR, console
```

If all that worked, you should see the Spark shell start up when you type

```
> spark-shell
```

6. SETTING UP PYSPARK

Now let's ensure that Python 2.7 is installed and configured.

```
> yum install -y centos-release-SCL
> yum install -y python27
> yum -y install python-pip
```

Finally, let's set up the environment variables for Spark and Python.

```
> vi $HOME/.bashrc
> export SPARK_HOME=/usr/hdp/2.3.6.0-3796/spark
> export PYTHONPATH=$SPARK_HOME/python
> export SPARK_HIVE=true
```

Then, reload the environment:

```
> source $HOME/.bashrc
```

Again, let's make sure Python is setup correctly. Type

```
> python
```

You should see the Python REPL come up. Finally, we can start writing code in Python.

7. WRITING OUR FIRST LINES OF PYSPARK

This example is taken from the Apache Spark website. It runs a parallelized operation to compute the value of π and is a good example to see the benefits of Spark.

```
def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
             .filter(inside).count()
print "Pi is roughly %f" % (4.0 * count / NUM_SAMPLES)
```

Once you've written your Python code, you can compile and deploy it with:

```
> spark-submit --master yarn-cluster MY_PYTHON_FILE.py
```

The *'–num-executors 10'* flag (arbitrary number) may be added to specify how many executors, the objects responsible for executing tasks, are to be used. Using as many executors as data nodes is recommended to minimize the runtime.

8. CONCLUSION

This technical report serves as a guide to set up an environment to run Spark on HDFS and write some simple Python code to take advantage of Spark using PySpark. With this setup, you can now take advantage of Spark's Resilient Distributed Datasets and have the data stored in memory and more quickly accessible.