ILLINOIS DATA SCIENCE INITIATIVE

SETTING UP SPARK FOR PYTHON

Version: 0.0.1

Author(s): Professor Brunner, Joshua Change, Quinn Jarrell, Benjamin Congdon, Sameet Sapra

April 11, 2017

Setting up Spark for Python

Professor Brunner[1,2,3], **Joshua Change**[1,2,3], **Quinn Jarrell**[1,2,3], **Benjamin Congdon**[1,2,3], **and Sameet Sapra**[1,2,3]

[1] *National Center For Supercomputing Applications (NCSA)*
[2] *Laboratory for Computation, Data, and Machine Learning*
[3] *Univeristy of Illinois*

Compiled April 11, 2017

An introduction to setting up Apache Spark with Python

https://github.com/lcdm-uiuc

ASSUMPTIONS

This technical report makes the following assumptions:

- The implementer has setup Apache Ambari on a cluster.

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 in 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; we will install and configure the Java APIs and PySpark, Python's API for Spark.

WHY PYSPARK?

PySpark is an API that interfaces with RDD's (Resilient Distributed Dataset) 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.

Additionally, Python allows for community data science packages to be used in the context of cloud computing. This ensures that the language does not become obsolete and receives continuous support from the community to have updated solutions to tackle the latest issues in data science.

PYSPARK PREREQUISITES

Installation will occur under a RedHat variant of Linux. 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.

INSTALLATION: HADOOP WITH YARN

If an older version of Java is already installed (1.5 and above), then skip to the Hadoop installation section.
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. The following instructions were adapted from the official Apache Ambari Installation guide[1].

Navigate to http://main.install.hostname:8080 on a browser, and login. The default credentials should be "admin" for the username, and "admin" for the password, unless it was changed in the Ambari configuration. Then you can create a cluster at the Welcome page. To build up the cluster, the installation wizard needs to know about all of the hosts. Enter all of the host names for the cluster into the **Target Hosts** textbox.

After clicking next, you can decide which of the services need to be installed on the cluster. HDFS will be installed as a minimum, but check the following boxes: YARN + MapReduce2, Tez, Hive, Pig, and Spark.

For assigning masters, slaves, and clients, leave everything on the default settings and continue to press next. Press Deploy on the final page, where it will install, start, and test all of the services previously checked.

This may take up to an hour. Once everything appears to be done, ensure that all installed services are running properly by navigating to the dashboard, where you should see green checkmarks for all of the installed services. Everything is running!

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
```

Note that this will change the version of Python on the PATH to default to 2.7.
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/python2.7
> export SPARK_HIVE=true
```

Then, reload the environment:

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

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

```
> spark-shell
```

Note that if SPARK_HOME is not set in your PATH, then you may have to change directories to the directory specified by SPARK_HOME to start the shell.

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

```
> python
```

You should see the Python REPL come up with the version that was just installed (2.7). Finally, we can start writing code in Python.

WRITING OUR FIRST LINES OF PYSPARK

This example is taken from the Apache Spark website[2]. 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

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

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

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

Note that my_file.py needs to be an executable so that Spark can recognize it.

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. Using the smallest sized executor as possible would lose the benefits of running tasks simultaneously in the same Java Virtual Machine. It is important to pick executors such that you leave overhead memory for the operating system and heap memory for Hadoop. It's best to keep up to 5 cores per executor for the maximum HDFS output.

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.

SOURCES

[1]: https://ambari.apache.org/1.2.1/installing-hadoop-using-ambari/content/
[2]: http://spark.apache.org/examples.html