Hortonworks Technical Preview for Apache Spark

Released: 04/30/2014
Welcome to Hortonworks Inc, technical preview for Apache Spark. The Technical Preview provides early access to upcoming features in the Hortonworks product, letting you test and review during the development process. These features are considered under development.

Although your feedback is greatly appreciated, these features are not intended for use in your production systems and not considered Supported by Hortonworks.

Have fun and please send feedback to us on the Community forums:

http://hortonworks.com/community/forums/
Apache Spark Introduction
The Spark Technical Preview lets you evaluate Apache Spark 0.9.1 on YARN with HDP 2.1. With YARN, Hadoop can now support multiple different types of workloads and Spark on YARN becomes another workload running against the same set of hardware resources.

This guide provides the configuration changes and environment variable needed to run Spark. It also provides canonical examples of running SparkPi and WordCount with Spark shell. When you are ready to go beyond that level of testing, see the machine learning examples on the Apache Spark website, https://spark.apache.org.

System Requirements
Spark Technical Preview has the following minimum system requirements:

- Operating Systems
- Software Requirements
- Sandbox Requirements

Operating systems
- 64-bit Red Hat Enterprise Linux (RHEL) 5 and 6
- 64-bit CentOS 5 and 6
- Oracle Linux 5 and 6

Software Requirements
- yum
- rpm
- curl
- wget
- unzip
- tar

Sandbox Requirements
To evaluate Spark on the HDP 2.1 Sandbox, you must add an entry to /etc/hosts to let the Sandbox or localhost to resolve to 127.0.0.1.

Here is an example entry:

127.0.0.1 localhost sandbox.hortonworks.com
Installation
Install the Apache Spark Technical Preview on a cluster with HDP 2.1 or HDP 2.1 for Sandbox installed.

1. Download the spark-core rpm.
   
   For RHEL/Centos5/Oracle Linux 5:
   
   http://public-repo-1.hortonworks.com/spark/centos5/rpms/spark-core-0.9.1.2.1.1.0-22.el5.noarch.rpm

   For RHEL/Centos5/Oracle Linux 6:
   
   http://public-repo-1.hortonworks.com/spark/centos6/rpms/spark-core-0.9.1.2.1.1.0-22.el6.noarch.rpm

2. Optional. If you want to run Spark with Python, download the spark-python rpm.
   
   For RHEL/Centos5:
   
   http://public-repo-1.hortonworks.com/spark/centos6/rpms/spark-python-0.9.1.2.1.1.0-22.el6.noarch.rpm

   For RHEL/Centos6/Oracle Linux 6:
   
   http://public-repo-1.hortonworks.com/spark/centos6/rpms/spark-python-0.9.1.2.1.1.0-22.el6.noarch.rpm

3. Install the Spark RPM.
   
   For RHEL/Centos5/Oracle Linux 5:
   
   sudo rpm -Uvh http://public-repo-1.hortonworks.com/spark/centos5/rpms/spark-core-0.9.1.2.1.1.0-22.el5.noarch.rpm

   For RHEL/Centos6/Oracle Linux 6:
   
   sudo rpm -Uvh http://public-repo-1.hortonworks.com/spark/centos6/rpms/spark-core-0.9.1.2.1.1.0-22.el6.noarch.rpm
   
   The spark-core rpm installs in /usr/lib/spark.

4. Optional. Install the spark-python rpm.
   
   For RHEL/Centos5/Oracle Linux 5:
   
   sudo rpm -Uvh http://public-repo-1.hortonworks.com/spark/centos5/rpms/spark-python-0.9.1.2.1.1.0-22.el5.noarch.rpm
For RHEL/Centos6/Oracle Linux 6:

```
sudo rpm -Uvh http://public-repo-1.hortonworks.com/spark/centos6/rpms/spark-python-0.9.1.2.1.1.0-22.el6.noarch.rpm
```

The spark-python rpm installs in /usr/lib/spark.

**Configuration**

To configure Spark for HDP 2.1:

1. Set the YARN_CONF_DIR environment variable to the Hadoop configuration directory. For example, if you installed by default using HDP 2.1 Sandbox:

   ```
   export YARN_CONF_DIR=/etc/hadoop/conf
   ```

2. Optional. If you did not install the Spark Technical Preview with HDP 2.1 Sandbox, you may need to set the yarn.application.classpath in yarn-site.xml.

   ```
   <property>
   <name>yarn.application.classpath</name>
   </property>
   ```

**Note:** Sandbox comes with this classpath already set.

**Running SparkPi on YARN**

To test compute-intensive tasks in Spark, the Pi example calculates \( \pi \) by throwing darts at a circle. The example points in the unit square ((0, 0) to (1, 1)) and sees how many fall in the unit circle. The fraction should be \( \pi / 4 \), which is used to estimate \( \pi \).

To execute SparkPi:

1. Change into your Spark home directory:

   ```
   cd /usr/lib/spark
   ```

2. Run the SparkPi example from a jar on the local file system.

   ```
   SPARK_JAR=lib/spark-assembly_2.10-0.9.1.2.1.1.0-22-hadoop2.4.0.2.1.1.0-385.jar \\
   .bin/spark-class org.apache.spark.deploy.yarn.Client --jar examples/lib/spark-
   examples_2.10-0.9.1.2.1.1.0-22.jar --class org.apache.spark.examples.SparkPi --
   args yarn-standalone --num-workers 2 --master-memory 512m --worker-memory 512m --
   worker-cores 1
   ```

   **Note:** Run this command as all as one line.

   Record the application ID in the following output. For example:
3. View the SparkPi results using a browser. For example if you are using the HDP 2.1 Sandbox, in a browser, navigate to:

http://sandbox.hortonworks.com:8088/cluster/app/application_1397599505608_0005

**Note:** The two values above in bold are specific to your environment. The example assumes that HDP 2.1 Sandbox is installed and on the host /etc/hosts maps to sandbox.hortonworks.com to localhost. The second bold value is the application ID from the previous step.

4. Click the **Logs** link in the bottom-right corner of the browser. The browser shows the YARN container output after a redirect:

... 
Log Type: stdout 
Log Length: 22 
Pi is roughly 3.13292 
...

**Note:** Other output has been omitted for brevity.

**Running Spark with WordCount**

This example uploads the Spark assembly and Spark examples jar to HDFS and use it from there, but you can also use the local files.
1. Before you begin, you must upload the input file to use in Word count to HDFS. You can use any input file. The following example uses log4j.properties as the input file:

    hadoop fs -copyFromLocal /etc/hadoop/conf/log4j.properties /tmp/data

2. Set the following environment variables:

    export SPARK_YARN_MODE=true
    export SPARK_JAR=/usr/lib/spark/lib/spark-assembly_2.10-0.9.1.2.1.10-22-hadoop2.4.0.2.1.10-385.jar
    export SPARK_YARN_APP_JAR=/usr/lib/spark/examples/lib/spark-examples_2.10-0.9.1.2.1.10-22.jar
    export SPARK_WORKER_MEMORY=512m
    export SPARK_MASTER_MEMORY=512m
    export MASTER=yarn-client

3. Run the Spark shell:

    /usr/lib/spark/bin/spark-shell

    **Note:** You may need to hit Enter if Spark-shell appears to hang getting the Scala prompt:

    scala>
    val file = sc.textFile("hdfs://sandbox.hortonworks.com:8020/tmp/data")
    val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
    counts.saveAsTextFile("hdfs://sandbox.hortonworks.com:8020/tmp/wordcount")

    **Note:** Run this command as all as one line.

4. View the WordCount output.

    a. Using Scala to print the number of unique words:

        scala > counts.count()

    b. Using Scala to print each word:

        scala > counts.toArray().foreach(println)

    c. Using the WordCount output.

        i. Exit the Scala shell and run:

        hadoop fs -ls /tmp/wordcount

        It should display output similar to the following:

        /tmp/wordcount/_SUCCESS
        /tmp/wordcount/part-00000
        /tmp/wordcount/part-00001
ii. Use the HDFS cat command to see the WordCount output. For example:

    hadoop fs -cat /tmp/wordcount/part-00000

Running Machine Learning Using Spark

Make sure all of your NodeManager nodes have the gfortran library. If not, install it in all of your NodeManager nodes:

    sudo yum install gcc-gfortran

**Note:** The gfortran package is usually available in the updates repos for CentOS. To install gcc-gfortran from the updates repo:

    sudo yum install gcc-gfortran --enablerepo=updates

If it cannot detect these libraries automatically, MLlib throws a linking error. For example, if you run Collaborative Filtering without the **gfortran runtime library** installed, you get this linking error:

```
java.lang.UnsatisfiedLinkError:
  org.jblas.NativeBlas.dposv(CII[DII[DII)
    at org.jblas.NativeBlas.dposv(Native Method)
    at org.jblas.SimpleBlas.posv(SimpleBlas.java:369)
    at org.jblas.Solve.solvePositive(Solve.java:68)
```

Start the Spark-shell, but first ensure that the following environment variables are set:

```
SPARK_YARN_MODE=true SPARK_JAR=/usr/lib/spark/lib/spark-assembly_2.10-0.9.1.2.1.1.0-22-
hadoop2.4.0.2.1.1.0-385.jar
SPARK_YARN_APP_JAR=/usr/lib/spark/examples/lib/spark-examples_2.10-0.9.1.2.1.1.0-22.jar
SPARK_WORKER_MEMORY=512m
SPARK_MASTER_MEMORY=512m
MASTER=yarn-client
/usr/lib/spark/bin/spark-shell
```

**Note:** The machine learning examples at [https://spark.apache.org/docs/latest/mllib-guide.html](https://spark.apache.org/docs/latest/mllib-guide.html) assume the example data is in HDFS.
You can create the appropriate directory tree and upload the file in HDFS. For example, to upload the file used in BinaryClassification example at http://spark.apache.org/docs/latest/mllib-guide.html:

```
hadoop fs -mkdir mllib
hadoop fs -mkdir mllib/data
hadoop fs -copyFromLocal /usr/lib/spark/mllib/data/sample_svm_data.txt
data/mllib/data/sample_svm_data.txt
```

**Troubleshooting**

*A null-pointer exception occurs when yarn.application.classpath is not set:*

```
14/04/21 11:43:22 INFO yarn.Client: Uploading file:/usr/lib/spark/lib/spark-assembly_2.10-0.9.1.2.1.1.0-22-hadoop2.4.0.2.1.1.0-385.jar to hdfs://192.168.25.130:8020/user/hdfs/.sparkStaging/application_1397899229161_0003/spark-assembly_2.10-0.9.1.2.1.1.0-22-hadoop2.4.0.2.1.1.0-385.jar
14/04/21 11:43:24 INFO yarn.Client: Setting up the launch environment Exception in thread "main" java.lang.NullPointerException
at scala.collection.mutable.ArrayOps$ofRef$.length$extension(ArrayOps.scala:114)
at scala.collection.mutable.ArrayOps$ofRef.length(ArrayOps.scala:114)
at scala.collection.IndexedSeqOptimized$class.foreach(IndexedSeqOptimized.scala:32)
at scala.collection.mutable.ArrayOps$ofRef.foreach(ArrayOps.scala:108)
at org.apache.spark.deploy.yarn.Client$.populateHadoopClasspath(Client.scala:500)
at org.apache.spark.deploy.yarn.Client$.populateClasspath(Client.scala:522)
at org.apache.spark.deploy.yarn.Client.setupLaunchEnv(Client.scala:334)
at org.apache.spark.deploy.yarn.Client.runApp(Client.scala:95)
at org.apache.spark.deploy.yarn.Client.run(Client.scala:116)
at org.apache.spark.deploy.yarn.Client$.main(Client.scala:495)
at org.apache.spark.deploy.yarn.Client.main(Client.scala)
```

**Solution:** Fix the yarn-site.xml file by adding the yarn.application.classpath variable:

```
<property>
  <name>yarn.application.classpath</name>
</property>
```
Error about HDFS non-existent InputPath when running Machine Learning examples

After a successful Spark installation, you can run Spark in any directory as long as you give the correct path, an absolute path for the local file system, a correct full path for HDFS. Such as:

```
at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:285)
at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:228)
at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:304)
at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scala:140)
at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:207)
at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:205)
at scala.Option.getOrElse(Option.scala:120)
at org.apache.spark.rdd.RDD.partitions(RDD.scala:205)
at org.apache.spark.rdd.MappedRDD.getPartitions(MappedRDD.scala:28)
at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:207)
```

(Omitted for brevity.)

Solution: Verify that the input data is uploaded to HDFS. See the previous section, Running Machine Learning Using Spark.

Known Issues

At the time of this release, there are no known issues for Apache Spark but you can visit the forum for the latest discussions on issues:

http://hortonworks.com/community/forums/

Further Reading

Apache Spark documentation is available here:

https://spark.apache.org/docs/latest/
About Hortonworks
Hortonworks is a leading commercial vendor of Apache Hadoop, the preeminent open source platform for storing, managing and analyzing big data. Hortonworks Data Platform provides an open and stable foundation for enterprises and a growing ecosystem to build and deploy big data solutions. Hortonworks is the trusted source for information on Hadoop, and together with the Apache community, Hortonworks is making Hadoop easier to install, manage and use. Hortonworks provides technical support, training & certification programs for enterprises, systems integrators & technology vendors.