Spark 1.1.0 Technical Preview - with HDP 2.1.5

Introduction

The Spark Technical preview lets you evaluate Apache Spark 1.1.0 on YARN with HDP 2.1.5. With YARN, Hadoop can now support various types of workloads; Spark on YARN becomes yet another workload running against the same set of hardware resources.

This technical preview describes how to:

- Run Spark on YARN and runs the canonical Spark examples of running SparkPI and Wordcount
- Work with a built-in UDF, collect-list, a key feature of Hive 13. This technical preview provides support for Hive 0.13.1 and instructions on how to call this UDF from Spark shell.
- Use ORC file as an HadoopRDD.

When you are ready to go beyond that these tasks, try the machine learning examples at Apache Spark.

Requirements

To evaluate Spark on the HDP 2.1 Sandbox, add an entry to on your Host machine in **/etc/hosts** to enable Sandbox or localhost to resolve to 127.0.0.1. For example:

127.0.0.1 localhost sandbox.hortonworks.com

Installing

The Spark 1.1.0 Technical Preview is provided as a single tarball.

Download the Spark Tarball

Use wget to download the Spark tarball:

```
wget http://public-repo-1.hortonworks.com/spark/centos6/1.1.0/tars/spark-1.1.0.2.1.5.0-695-bin-2.4.0.2.1.5.0-695.tgz
```

Copy the Spark Tarball to a HDP 2.1 Cluster:

Copy the downloaded Spark tarball to your HDP 2.1 Sandbox or to your Hadoop cluster.

For example, the following command copies Spark to HDP 2.1 Sandbox:

```
scp -P 2222 spark-1.1.0.2.1.5.0-695-bin-2.4.0.2.1.5.0-695.tgz root@127.0.0.1:/root
```

Note: The password for HDP 2.1 Sandbox is `hadoop`.

Untar the Tarball

To untar the Spark tarball, run:

```
tar xvfz spark-1.1.0.2.1.5.0-695-bin-2.4.0.2.1.5.0-695.tgz
```

Set the YARN environment variable

Specify the appropriate directory for your Hadoop cluster. For example, if your Hadoop and YARN config files are in /etc/hadoop/conf:

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

Set yarn.application.classpath in yarn-site.xml. In the HDP 2.1 Sandbox, yarn.application.classpath is already set, so there is no need to set yarn.application.classpath to set up Spark on the HDP 2.1 Sandbox.

If you are running Spark against your own HDP 2.1 cluster ensure that yarn-site.xml has the following value for yarn.application.classpath property:

```
<property>
   <name>yarn.application.classpath</name>
<value>/etc/hadoop/conf,/usr/lib/hadoop/*,/usr/lib/hadoop/lib/*,/usr/
lib/hadoop-hdfs/*,/usr/lib/hadoop-hdfs/lib/*,/usr/lib/hadoop-yarn/*,/
usr/lib/hadoop-yarn/lib/*</value>
</property>
```

Run Spark Pi Example

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 calculate Pi with Spark:

1. Change to your Spark directory:

```
cd spark-1.1.0.2.1.5.0-695-bin-2.4.0.2.1.5.0-695
```

2. Run the Spark Pi example:

```
./bin/spark-submit --class org.apache.spark.examples.SparkPi    -
-master yarn-cluster  --num-executors 3 --driver-memory 512m  --
executor-memory 512m   --executor-cores 1  lib/spark-examples*.jar 10
```

Note: The Pi job should complete without any failure messages and produce output similar to:

```
14/09/12 09:52:01 INFO yarn.Client: Application report from
ResourceManager:
      application identifier: application_1410479103337_0003
      appId: 3
      clientToAMToken: null
      appDiagnostics:
      appMasterHost: sandbox.hortonworks.com
      appQueue: default
      appMasterRpcPort: 0
      appStartTime: 1410540670802
      yarnAppState: FINISHED
      distributedFinalState: SUCCEEDED
      appTrackingUrl: http://sandbox.hortonworks.com:8088/proxy/
application_1410479103337_0003/A
      appUser: root
```

1. To view the results in a browser, copy the appTrackingUrl and go to:

`http://`**`sandbox.hortonworks.com:8088`**`/proxy/`
`application_1410479103337_0003/A`

Note: The two values above in bold are specific to your environment. These instructions assume that HDP 2.1 Sandbox is installed and that /etc/hosts is mapping sandbox.hortonworks.com to localhost.

Using WordCount with Spark

Click the logs link in the bottom right
The browser shows the YARN container output after a redirect.
Note the following output on the page. (Other output omitted for brevity.)

```
…..
14/09/12 09:52:00 INFO yarn.ApplicationMaster: AppMaster received a
signal.
14/09/12 09:52:00 INFO yarn.ApplicationMaster: Deleting staging
directory .sparkStaging/application_1410479103337_0003
14/09/12 09:52:00 INFO yarn.ApplicationMaster$$anon$1: Invoking sc
stop from shutdown hook
14/09/12 09:52:00 INFO ui.SparkUI: Stopped Spark web UI at http://
sandbox.hortonworks.com:42078
14/09/12 09:52:00 INFO spark.SparkContext: SparkContext already
```

```
stopped

Log Type: stdout
Log Length: 23
Pi is roughly 3.144484
```

Copy input file for Spark WordCount Example

Upload the input file you want to use in WordCount to HDFS. You can use any text file as input. In the following example, log4j.properties is used as an example:

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

Run Spark WordCount

To run WordCount:

1. Run the Spark shell:

   ```
   ./bin/spark-shell
   ```

   Output similar to below displays before the Scala REPL prompt, `scala>`:

   ```
   Spark assembly has been built with Hive, including Datanucleus
   jars on classpath
   14/09/11 17:33:47 INFO spark.SecurityManager: Changing view acls
   to: root,
   14/09/11 17:33:47 INFO spark.SecurityManager: Changing modify
   acls to: root,
   14/09/11 17:33:47 INFO spark.SecurityManager: SecurityManager:
   authentication disabled; ui acls disabled; users with view
   permissions: Set(root, ); users with modify permissions:
   Set(root, )
   14/09/11 17:33:47 INFO spark.HttpServer: Starting HTTP Server
   14/09/11 17:33:47 INFO server.Server: jetty-8.y.z-SNAPSHOT
   14/09/11 17:33:47 INFO server.AbstractConnector: Started
   SocketConnector@0.0.0.0:44066
   14/09/11 17:33:47 INFO util.Utils: Successfully started
   service 'HTTP class server' on port 44066.
   Welcome to
         ____              __
        / __/__  ___ _____/ /__
       _\ \/ _ \/ _ `/ __/  '_/
      /___/ .__/\_,_/_/ /_/\_\   version 1.1.0
   ```

```
      /_/

   Spark context available as sc.

   scala>
```

1. At the Scala REPL prompt enter:

```
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")
```

Viewing the WordCount output with Scala Shell

To view the output in the scala shell:

```
scala > counts.count()
```

To print the full output of the WordCount job:

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

Viewing the WordCount output with HDFS

To read the output of WordCount using HDFS command:

1. Exit the scala shell.

   ```
   scala > exit
   ```

2. View WordCount Results:

   ```
   hadoop fs -ls /tmp/wordcount
   ```

   It should display output similar to:

   ```
   /tmp/wordcount/_SUCCESS
   /tmp/wordcount/part-00000
   /tmp/wordcount/part-00001
   ```

1. Use the HDFS cat command to see the WordCount output. For example,

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

Running Hive 0.13.1 UDF

Before running Hive examples run the following steps:

Copy hive-site to Spark conf

For example, ensure the paths used match your environment:

```
cp /usr/lib/hive/conf/hive-site.xml /root/spark-1.1.0.2.1.5.0-695-bin-2.4.0.2.1.5.0-695/conf/
```

Comment out ATS Hooks

Ensure the following properties in the Spark copy of hive-site.xml are removed (or commented out)

```
<name>hive.exec.pre.hooks</name>
<value>org.apache.hadoop.hive.ql.hooks.ATSHook</value>
<name>hive.exec.failure.hooks</name>
<value>org.apache.hadoop.hive.ql.hooks.ATSHook</value>
<name>hive.exec.post.hooks</name>
<value>org.apache.hadoop.hive.ql.hooks.ATSHook</value>
```

Hive 0.13.1 provides a new built-in UDF collect_list(col) which returns a list of objects with duplicates.

Launch Spark Shell on YARN cluster

```
./bin/spark-shell --num-executors 2 --executor-memory 512m --master yarn-client
```

Create Hive Context

```
scala> val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)

You should see output similar to the following:
…
hiveContext: org.apache.spark.sql.hive.HiveContext =
org.apache.spark.sql.hive.HiveContext@7d9b2e8d
```

Create Hive Table

```
scala> hiveContext.hql("CREATE TABLE IF NOT EXISTS TestTable (key INT, value STRING)")
You should see output similar to the following:
…
res1: org.apache.spark.sql.SchemaRDD =
```

```
SchemaRDD[5] at RDD at SchemaRDD.scala:103
== Query Plan ==
<Native command: executed by Hive>
```

Load example KV value data into Table

```
scala> hiveContext.hql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/
kv1.txt' INTO TABLE TestTable")
```

You should see output similar to the following:

```
14/09/12 10:05:20 INFO log.PerfLogger: </PERFLOG method=Driver.run
start=1410541518525 end=1410541520023 duration=1498
from=org.apache.hadoop.hive.ql.Driver>
res2: org.apache.spark.sql.SchemaRDD =
SchemaRDD[8] at RDD at SchemaRDD.scala:103
== Query Plan ==
<Native command: executed by Hive>
```

Invoke Hive collect_list UDF

```
scala> hiveContext.hql("from TestTable SELECT key, collect_list(value)
group by key order by key").collect.foreach(println)
```

You should see output similar to the following:

```
...
[489,ArrayBuffer(val_489, val_489, val_489, val_489)]
[490,ArrayBuffer(val_490)]
[491,ArrayBuffer(val_491)]
[492,ArrayBuffer(val_492, val_492)]
[493,ArrayBuffer(val_493)]
[494,ArrayBuffer(val_494)]
[495,ArrayBuffer(val_495)]
[496,ArrayBuffer(val_496)]
[497,ArrayBuffer(val_497)]
[498,ArrayBuffer(val_498, val_498, val_498)]
```

Using ORC file as HadoopRDD

Create a new Hive Table with ORC format

```
scala>hiveContext.sql("create table orc_table(key INT, value STRING)
stored as orc")
```

Load Data into the ORC table

```
scala>hiveContext.hql("INSERT INTO table orc_table select * from
testtable")
```

Verify that Data is loaded into the ORC table

```
scala>hiveContext.hql("FROM orc_table SELECT
*").collect().foreach(println)
```

Read ORC Table from HDFS as HadoopRDD

```
scala> val inputRead = sc.hadoopFile("hdfs://
sandbox.hortonworks.com:8020/apps/hive/warehouse/orc_table",
classOf[org.apache.hadoop.hive.ql.io.orc.OrcInputFormat],classOf[org.a
pache.hadoop.io.NullWritable],classOf[org.apache.hadoop.hive.ql.io.orc
.OrcStruct])
```

Verify we can manipulate the ORC record through RDD

```
scala> val k = inputRead.map(pair => pair._2.toString)
scala> val c = k.collect
```

You should see output similar to the following:

```
...
scheduler.DAGScheduler: Stage 7 (collect at <console>:16) finished in
0.518 s
14/09/16 11:54:58 INFO spark.SparkContext: Job finished: collect at
<console>:16, took 0.532203184 s
c1: Array[String] = Array({238, val_238}, {86, val_86}, {311,
val_311}, {27, val_27}, {165, val_165}, {409, val_409}, {255,
val_255}, {278, val_278}, {98, val_98}, {484, val_484}, {265,
val_265}, {193, val_193}, {401, val_401}, {150, val_150}, {273,
val_273}, {224, val_224}, {369, val_369}, {66, val_66}, {128,
val_128}, {213, val_213}, {146, val_146}, {406, val_406}, {429,
val_429}, {374, val_374}, {152, val_152}, {469, val_469}, {145,
val_145}, {495, val_495}, {37, val_37}, {327, val_327}, {281,
val_281}, {277, val_277}, {209, val_209}, {15, val_15}, {82, val_82},
{403, val_403}, {166, val_166}, {417, val_417}, {430, val_430}, {252,
val_252}, {292, val_292}, {219, val_219}, {287, val_287}, {153,
val_153}, {193, val_193}, {338, val_338}, {446, val_446}, {459,
val_459}, {394, val_394}, {...
```

Running the Machine Learning Spark Application

Make sure all of your nodemanager nodes have gfortran library. If not, you need to install it in all of your nodemanager nodes.

```
sudo yum install gcc-gfortran
```

Note: It is usually available in the update repo for CentOS. For example:

```
sudo yum install gcc-gfortran --enablerepo=update
```

MLlib throws a linking error if it cannot detect these libraries automatically. For example, if you try to do Collaborative Filtering without gfortran runtime library installed, you will see the following linking error:

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

Visit http://spark.apache.org/docs/latest/mllib-guide.html for Spark ML examples.

Troubleshooting

Issue:
Spark submit fails.

Note the error about failure to set the env:

```
Exception in thread "main" java.lang.Exception: When running with
master 'yarn-cluster' either HADOOP_CONF_DIR or YARN_CONF_DIR must be
set in the environment.
      at
org.apache.spark.deploy.SparkSubmitArguments.checkRequiredArguments(Spa
rkSubmitArguments.scala:182)
      …
```

Solution:
Set the environment variable YARN_CONF_DIR as following.

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

Issue:
Spark submitted job fails to run and appears to hang.

In the YARN container log you will notice the following error:

```
14/07/15 11:36:09 WARN YarnClusterScheduler: Initial job has not
accepted any resources; check your cluster UI to ensure that workers
are registered and have sufficient memory
14/07/15 11:36:24 WARN YarnClusterScheduler: Initial job has not
accepted any resources; check your cluster UI to ensure that workers
are registered and have sufficient memory
14/07/15 11:36:39 WARN YarnClusterScheduler: Initial job has not
accepted any resources; check your cluster UI to ensure that workers
are registered and have sufficient memory
```

Solution:
The Hadoop cluster must have sufficient memory for the request. For example, submitting the following job with 1GB memory allocated for executor and Spark driver fails with the above error in the HDP 2.1 Sandbox. Reduce the memory asked for the executor and the Spark driver to 512m and re-start the cluster.

```
./bin/spark-submit --class org.apache.spark.examples.SparkPi    -
-master yarn-cluster  --num-executors 3 --driver-memory 512m  --
executor-memory 512m   --executor-cores 1  lib/spark-examples*.jar 10
```

Issue:
Error message about HDFS non-existent InputPath when running Machine Learning examples.

```
org.apache.hadoop.mapred.InvalidInputException: Input path does not exist: hdfs://
sandbox.hortonworks.com:8020/user/root/mllib/data/sample_svm_data.txt
        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:
Ensure that the input data is uploaded to HDFS.

Known Issues

Spark Thrift Server does not work with this tech preview.

There are no other known issues for Apache Spark. Visit the forum for the latest discussions on issues:

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

Further Reading

Apache Spark documentation is available here:

https://spark.apache.org/docs/latest/