Experiences Implementing Apache Spark™ and Apache Hadoop™ across a Clustered Environment using Linux on z

Bob McNamara
Dulce Smith
Table of Contents

1. Introduction ........................................................................................................................................... 3
2. Describing our Clustering Environment ............................................................................................ 3
3. Filesystem Configuration ....................................................................................................................... 7
4. Hadoop® Cluster Implementation ........................................................................................................ 9
5. Installing and Configuring Hadoop YARN as the Cluster Manager ................................................... 11
6. Apache Spark™ Implementation .......................................................................................................... 12
7. Analyzing our Environment with SparkBench ..................................................................................... 14
8. Summary ............................................................................................................................................. 20
1. Introduction

With the challenges attached to big data, Apache Spark has become an instrumental highly scalable framework in the open source community. Spark takes a different approach to big data by breaking down the computing and data into smaller segments. This paper focuses on the implementation of Spark in a cluster of RHEL servers running on System z. In the subsequent pages, we’ll explain how we built our cluster environment and implemented Spark by using Apache Hadoop for data storage and cluster management. Also, we will show you how we implemented YARN as the resource manager. We’ll also discuss how Spark acts as a powerful processing engine by using its in-memory technology to process big data. Lastly, we’ll show you how to install and run SparkBench, a benchmark tool useful to assess environments.

2. Describing our Clustering Environment

In Hadoop, you could technically have a one-host cluster, which is mostly used for testing environments. For production environments, you would want to configure a cluster of more than one host. Our environment consists of a cluster of nine RHEL 7.1 hosts running as guests on one z/VM LPAR. This platform allows you to build cost effective clusters with the ability to virtualize resources, resize and add nodes dynamically, and enjoy generous network bandwidths.

To keep everything simple, we configured each guest with:

- 4 processors
- 32 GBs of memory
- 1 1Gb Ethernet adapter
- 2 8Gb SCSI adapters, each connected to a separate SAN fabric
- 1 500GB external LUNs A Storwize V7000 with RAID6

To determine how to use our z/VM guests, we considered the different components in Spark, Hadoop, and YARN.

The Apache Spark site illustrates the Spark architecture. The nodes types are: Client Driver, Cluster Manager, and Worker Node (or Slave). Our cluster is composed of seven nodes as follows: one node holds the Spark Client Driver (bgnsb00), another node holds the Spark Cluster Manager (bgnsb01) also known as the Spark Master, and the last five nodes hold the Spark Workers (bgnsb02, bgnsb03, bgnsd01, bgnsd02, and bgnsd03).

Furthermore, there are three types of cluster managers: 1) Standalone, where workers are registered with the Client Master (in our case, Spark Master), 2) YARN, where workers register the YARN Cluster Manager, and 3) Mesos, where workers register with Mesos. We tested both cluster manager configurations, Standalone and YARN. However, in this paper, we will focus the YARN cluster manager configuration.
Hadoop is the next piece of open source technology we deployed in our cluster. It uses a NameNode as the coordinator to the client. The NameNode holds metadata and communicates with DataNodes to store and retrieve data for the client program. In our testing environment, we have five DataNodes: bgnsb02, bgnsb03, bgnsd01, bgnsd02, and bgnsd03. These nodes are also our Spark workers.

The next chart depicts the primary NameNode as well as the secondary NameNode. For a production environment, the secondary NameNode should be on another z/VM guest, but we kept it on the same node as our primary NameNode for testing purposes.
YARN, sometimes called MapReduce 2.0, is part of the Hadoop 2.0 open source suite. YARN allows you to run multiple applications such as MapReduce and Spark under the same cluster environment. Spark applications can run on YARN under the YARN Cluster Manager. There are three YARN components: the ResourceManager (RM), ApplicationMaster (AP), and NodeManager (NM).

The following diagram was obtained from the Apache Hadoop website, which shows the ResourceManager allowing multiple clients under the same environment: https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html
This is the overall view of our Spark on YARN cluster mode, which we’ll explain how to setup. It depicts our cluster configuration running under a single z/VM LPAR, sharing the same subnet.
3. Filesystem Configuration

Hadoop NameNode

A Namespace on the NameNode contains the metadata for the HDFS filesystem. The NameNode uses a transaction log, called the EditLog, to record updates to the filesystem metadata. The EditLog is persisted in a local file system on the NameNode. The local filesystem also contains the FSImage, which is a point-in-time image of the HDFS file system directory tree and mapping of blocks to files.

We created two file systems with 8 GBs of space, each, to hold the NameSpace data. A separate disk for each file system is recommended. The filesystems are as follow:

/nn/namespace1
/nn/namespace2

In addition, we added references to them in the hdfs-site.xml config file. Here is what we added:

hdfs-site.xml:
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///nn/namespace1,file:///nn/namespace2</value>
</property>

Also, we created a 4GB file system called /hadoop_log on all of the Hadoop nodes to hold the log and audit files and referenced this to the Hadoop-env.sh file as follow:

# Where log files are stored.  $HADOOP_HOME/logs by default.
#export HADOOP_LOG_DIR=${HADOOP_LOG_DIR}/$USER
export HADOOP_LOG_DIR=/hadoop_log

Hadoop DataNode

The DataNode stores blocks of a HDFS file in files on its local file system. The blocks of the same HDFS file are spread over multiple DataNodes. An individual DataNode only stores the blocks. It has no knowledge of the HDFS filesystem.

We created two file systems with 128 GBs on each DataNode. The file systems are:

/dn/data1
/dn/data2

In addition, we added references to these filesystems in the hdfs-site.xml file of each DataNode. Again, a separate disk is recommended for each local filesystem.

hdfs-site.xml:
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///dn/data1,file:///dn/data2</value>
</property>

Spark Worker node
The Spark worker can use a directory to store its log files and scratch space. When we tested the Spark standalone cluster, we created a single 64 GBs local file system for each worker node called /spark1. This configuration was added to the spark-env.sh configuration file as shown below:

```
SPARK_LOCAL_DIRS=/spark1
```

When we ran in YARN cluster mode, the YARN local filesystem was used. Those are reviewed next.

**YARN NodeManager**

The NodeManager uses local directories to store local files and scratch space. Spark applications running in YARN cluster mode also uses these directories. We created two file systems with 8 GBs per NodeManager. In addition, we created two log file directories with 4 GB each to hold the NodeManager application container stderr and syslog files. Here are the file systems:

```
/nm/data1
/nm/data2
/nm/log1
/nm/log2
```

Here in the yarn-site.xml is where we reference these file systems:

```
yarn-site.xml:
<property>
  <name>yarn.nodemanager.local-dirs</name>
  <value>file:///nm/data1,file:///nm/data2</value>
  <description>The name of locals paths for data</description>
</property>
<property>
  <name>yarn.nodemanager.log-dirs</name>
  <value>file:///nm/log1,file:///nm/log2</value>
  <description>The FS where logs are written</description>
</property>
```

Then, we kept the NodeManager logs in /Hadoop_log by changing the yarn.env.sh file:

```
# default log directory & file
if [ "$YARN_LOG_DIR" = "" ]; then
  # YARN_LOG_DIR="$HADOOP_YARN_HOME/logs"
  YARN_LOG_DIR=~/hadoop_log
```

Since we used five z/VM guests with a Hadoop DataNode, Spark worker, and YARN NodeManager on each, these are the filesystems we created on each of the five guests in addition to the OS file systems:

```
/dev/mapper/sparkvg-dndata1  131980772   62644 125207244   1% /dn/data1
/dev/mapper/sparkvg-dndata2  131980772   62948 125206940   1% /dn/data2
/dev/mapper/sparkvg-hadoop_log  3997376   12376  3775288   1% /hadoop_log
/dev/mapper/sparkvg-nmlog2   8125880   18772  7687680   1% /nm/log2
/dev/mapper/sparkvg-nmddata1   8125880  572996  7133456   8% /nm/data1
/dev/mapper/sparkvg-nmlog1   8125880  18744  7687708   1% /nm/log1
/dev/mapper/sparkvg-nmdata2   8125880 203452  7503000   3% /nm/data2
```

**YARN ResourceManager**
The only file system configuration needed for the ResourceManager was the /hadoop_log filesystem used for the ResourceManager’s stdout and log files.

4. Hadoop® Cluster Implementation

Apache Hadoop is an open source framework that allows for the processing of large data sets (schema-less: structured and unstructured) across clusters. There are many components or Hadoop’s projects, as it is called in the open source community; however, its two main components are: 1, HDFS (file system) and 2, MapReduce (data processing). Hadoop splits data and distributes them across nodes in the cluster. Each node provides local processing and storage. In sum, Hadoop brings massive parallel competing to commodity servers. You can start small and scale linearly as your work requires.

Installing Hadoop typically involves unpacking the software on all members of the cluster. We designated one machine as the NameNode and another as the ResourceManager, exclusively. These are the Masters. Other nodes in the cluster, they each run as both, DataNodes and NodeManagers; these are the commonly known as Slaves.

Each of these nodes run on a single z/VM LPAR, which houses all the nodes in the cluster. We used RHEL 7.1 for the operating system using ECKD DASD. However, we store the data on SCSI DASD. Hadoop HDFS was used as the file system of choice though others can be used. We installed all software under the /usr/local/ directory on each node.

To install Hadoop, we followed the Apache website for a cluster installation (https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/ClusterSetup.html). To summarize:

1. Add all hostnames for the cluster to /etc/hosts and distribute to all guests
2. Install Java on all guests
3. Download the Hadoop 2.7.2 binary file from the Apache Hadoop download site to an NFS filesystem mounted as /mnt; this NFS filesystem is mounted on all guests. Create a directory under /mnt and call it ‘Hadoop.’ Unpack the binary image under /mnt/hadoop
4. Copy the hadoop image directory to all guests with the command “cp –r /mnt/hadoop/hadoop-2.7.2 /usr/local/hadoop”
5. Create a userid called ‘hdfs.’ This is the primary userid for administering the hdfs filesystem and starting/stopping the Hadoop cluster. Add “/usr/local/hadoop/bin” to the PATH variable in .bash_profile
6. Using the hdfs userid, setup password-less ssh from the NameNode to the Hadoop DataNodes. This allowed us to stop/start the cluster from the NameNode without a password prompt from each DataNode
7. Configure the hadoop-env.sh file to add:

    export JAVA_HOME=/usr/bin/java
export HADOOP_LOG_DIR=/hadoop_log

8. Configure the following files under /usr/local/hadoop/etc/hadoop and distribute to all Hadoop nodes:

a. Change core-site.xml to add the Hadoop NameNode:

```xml
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://bgnsd00:8020</value>
</property>
```

b. Configure hdfs-site.xml to show the local filesystems used by the Hadoop NameNode and DataNodes.

```xml
<configuration>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///nn/namespace1,file:///nn/namespace2</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///dn/data1,file:///dn/data2</value>
  </property>
  <property>
    <name>dfs.permissions</name>
    <value>false</value>
  </property>
</configuration>
```

c. Configure the slaves file to show all five slaves on our cluster

bgnsb02
bgnsb03
bgnsd01
bgnsb02
bgnsd03

To start the Hadoop cluster, we first formatted the NameNode as user hdfs with the command:

```
/usr/local/hadoop/bin/hdfs namenode -format
```

Then we ran start-dfs.sh from the NameNode. This started daemons on the NameNode, Secondary NameNode and the DataNodes (again, for testing purposes, we installed the Secondary NameNode on the same node). To start all HDFS processes, we used the following command:

```
/usr/local/hadoop/sbin/start-dfs.sh
```

16/04/20 09:54:10 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using built-in-java classes where applicable
Starting namenodes on [bgnsd00]
bgnsd00: starting namenode, logging to /hadoop_log/hadoop-hdfs-namenode-bgnad00.out
bgnsb02: starting datanode, logging to /hadoop_log/hadoop-hdfs-datanode-bgnab02.out
bgnsb03: starting datanode, logging to /hadoop_log/hadoop-hdfs-datanode-bgnab03.out
bgnsd01: starting datanode, logging to /hadoop_log/hadoop-hdfs-datanode-bgnad01.out
bgnsd03: starting datanode, logging to /hadoop_log/hadoop-hdfs-datanode-bgnad03.out
bgnsb02: starting datanode, logging to /hadoop_log/hadoop-hdfs-datanode-bgnad02.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /hadoop_log/hadoop-hdfs-secondarynamenode-bgnad00.out
The warning on the native-Hadoop library is due to big endian versus little endian encoding. We’ll experiment later to see if a Hadoop source compile will resolve this.

To start the Hadoop user interface, we entered the following link on a Web browser:

http://10.20.82.251:50070 (NameNode address followed by port 50070)

5. Installing and Configuring Hadoop YARN as the Cluster Manager

Yet Another Resource Negotiator (YARN) is a platform used to run big data applications and is distributed with Hadoop version 2 (also known as MapReduce2). YARN is a resource manager that provides daemons and APIs to handle and schedule resource requests made by applications such as MapReduce and Spark. It also supervises those applications. We initially configured Spark to run in standalone mode but later proceeded to configure and test YARN to manage Spark applications.

YARN is part of the Hadoop suite. Therefore, when we installed Hadoop, YARN was also installed. Our YARN ResourceManager is bgnsd04, and our YARN NodeManagers are found on each of the slaves (bgnsb02, bgnsb03, bgnsd01, bgnsd02, and bgnsd03). To accommodate our environment, we configured YARN as follows:

1. Create a userid called ‘yarn’ to administer the YARN cluster. We added the YARN ID to the hdfs group so that it could write to the local /hadoop_log on the slave nodes and ResourceManager. We made the same changes to the .bash_profile as the hdfs userid to point to the Hadoop bin directory.

2. Setup password-less ssh from the ResourceManager to the YARN NodeManager guests. This is done so that we can start/stop the YARN cluster from the ResourceManager with the YARN userid.

3. Configure the following files under /usr/local/hadoop/etc/hadoop on all YARN nodes:
   a. yarn-env.sh – set JAVA_HOME and the YARN logging directory:
      b. JAVA_HOME=/usr/bin/java
      c. YARN_LOG_DIR=/hadoop_log
   d. yarn-site.xml – set the ResourceManager hostname and NodeManager local directories:

   <property>
   <name>yarn.resourcemanager.hostname</name>
   <value>bgnsd04</value>
   <description>The hostname of the ResourceManager</description>
   </property>

   <property>
   <name>yarn.nodemanager.local-dirs</name>
   <value>file:///nm/data1,file:///nm/data2</value>
   <description>The name of locals paths for data</description>
   </property>

   <property>
   <name>yarn.nodemanager.log-dirs</name>
   <value>file:///nm/log1,file:///nm/log2</value>
   <description>The FS where logs are written</description>
   </property>

   4. Start the YARN daemons from the resource manager
6. Apache Spark™ Implementation

Apache Spark is an open source processing engine that utilizes in-memory technology to compute and analyze big data. This is a major indicator why IBM System Z is suitable for this open source technology as it supports up to 10 TBs of memory, allowing environments to achieve optimal performance. In addition, Spark is based on Scala, a language that integrates object oriented and functional programming concepts. Furthermore, Spark can run in standalone mode (client mode) or on top of YARN (cluster mode). In YARN client mode, the driver runs in the client process, and the application master is only used for requesting resources from YARN. In YARN cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application. We are running our cluster in YARN cluster mode, which we’ll show you shortly how we configured it. For now, let us show you how we installed and configured Spark (including Scala). Here is what we did:

1. Installed Java 1.8 using YUM on the cluster manager and Spark workers
2. Downloaded Spark from the Apache Spark download site. Unpacked it to the NFS mounted directory and copied to the /usr/local/spark directory
3. Downloaded Scala, unpacked it, and copied it to the /usr/local/scala directory on all Spark nodes
4. Added paths to .bash_profile for /usr/local/spark/bin and /usr/local/scala/bin as follow:
   ```bash
   PATH=$PATH:$HOME/bin:/usr/local/scala/bin:/usr/local/spark/bin
   export PATH
   ```
5. Added password-less ssh from the Spark cluster manager node (bgnsb01) to the Spark worker nodes
6. Under the /usr/local/spark/conf directory, we:
   a. renamed the following files to remove the `.template` extension
      - log4j.properties.template
      - spark_defaults.conf.template
      - slaves.template
      - spark-env.sh.template
   b. edited the spark-defaults.conf file to define the Spark master node
      ```
      spark.master   spark://bgnsb01:7077
      ```
7. Under the /usr/local/spark/conf directory, we:
a. included all five workers in the ‘slaves’ file
b. used defaults on the ‘log4j.properties’ file
c. set local directories for scratch space and logging on the ‘spark-env.sh’ file

SPARK_LOCAL_DIRS=/spark1
SPARK_LOG_DIR=/hadoop_log

8. To start the Spark shell, at the command prompt, we executed:

[root@bgnsb00 ~]# spark-shell
log4j:WARN No appenders could be found for logger
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's repl log4j profile: org/apache/spark/log4j-defaults-repl.properties
To adjust logging level use sc.setLogLevel("INFO")
Welcome to

  / __/
 / _ `
| |\_ \ _ _ \_ _ \\ _ _ \\
| | | | | | | | | | \

version 1.6.0

Using Scala version 2.10.5 (OpenJDK 64-Bit Zero VM, Java 1.8.0_65)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.

16/01/14 14:35:31 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
16/01/14 14:35:31 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
16/01/14 14:35:42 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0
16/01/14 14:35:42 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
OpenJDK 64-Bit Zero VM warning: You have loaded library /tmp/libnetty-transport-native-epoll2868761762691264077.so which might have disabled stack guard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
16/01/14 14:35:47 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
16/01/14 14:35:48 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
16/01/14 14:35:58 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0
16/01/14 14:35:58 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
SQL context available as sqlContext.

Scala>

Since we did not configure HIVE or any SQL access, we were able to ignore the BoneCP and default database warnings.
9. Then, we tested starting the cluster manager and Spark workers from the standalone cluster manager node:

   [root@bgnsb01 spark]# ./sbin/start-all.sh
   bgnsb01: starting org.apache.spark.deploy.worker.Worker, logging to /hadoop_log/spark-root-org.apache.spark.deploy.worker.Worker-1-bgnsb01.out
   bgnsb03: starting org.apache.spark.deploy.worker.Worker, logging to /hadoop_log/spark-root-org.apache.spark.deploy.worker.Worker-1-bgnsb03.out
   bgnsb02: starting org.apache.spark.deploy.worker.Worker, logging to /hadoop_log/spark-root-org.apache.spark.deploy.worker.Worker-1-bgnsb02.out
   bgnsd03: starting org.apache.spark.deploy.worker.Worker, logging to /hadoop_log/spark-root-org.apache.spark.deploy.worker.Worker-1-bgnsd03.out
   bgnsd02: starting org.apache.spark.deploy.worker.Worker, logging to /hadoop_log/spark-root-org.apache.spark.deploy.worker.Worker-1-bgnsd02.out
   bgnsb03: starting org.apache.spark.deploy.worker.Worker, logging to /hadoop_log/spark-root-org.apache.spark.deploy.worker.Worker-1-bgnsb03.out
   bgnsd03: starting org.apache.spark.deploy.worker.Worker, logging to /hadoop_log/spark-root-org.apache.spark.deploy.worker.Worker-1-bgnsd03.out
   bgnsb02: starting org.apache.spark.deploy.worker.Worker, logging to /hadoop_log/spark-root-org.apache.spark.deploy.worker.Worker-1-bgnsb02.out

7. Analyzing our Environment with SparkBench

SparkBench is a Spark-specific benchmarking suite. It’s composed of a set of workloads belonging to four different application types: 1, machine learning 2, graph processing 3, steaming and 4, SQL queries. These applications help analyze and cover the following resources: CPU, memory, and shuffle and I/O intensive workloads. It includes a data generator that allows users to generate input data.

This suit is supposed to help users understand the trade-off between different system designs, configuration optimization, and cluster provisioning for Spark deployments.

Here are the installation steps we followed:

1. Copy the scala-2.11.7.tgz file into /usr/local/scala
2. Untar the file: tar -xvf scala-2.11.7.tgz
3. Install the git, unzip, and java-devel via YUM
4. Install Maven under /usr/local/maven (created directory)
5. Create the /usr/local/sparkbench directory and cd to it
7. Change directories to /usr/local/sparkbench/wikixmlj
8. Execute: /usr/local/maven/apache-maven-3.3.9/bin/mvn package install
9. Unzip the spark-bench-master.zip (benchmark suite) file into the /usr/local/ directory
10. Change directories to /usr/local/spark-bench-master/bin and execute: ./build-all.sh
11. At the end of the output, we looked for the following results to confirm execution success:

    [INFO] ------------------------------------------------------------------------
    [INFO] Reactor Summary:
    [INFO] SparkBench ......................................... SUCCESS
    [INFO] Common ........................................... SUCCESS
INFO] Tera
sortApp ................................... SUCCESS
INFO] PCAApp .................................. SUCCESS
INFO] LogisticRegressionApp .................... SUCCESS
INFO] KMeansApp ................................ SUCCESS
INFO] LinearRegressionApp ....................... SUCCESS
INFO] SVMApp .................................. SUCCESS
INFO] DecisionTreeApp .......................... SUCCESS
INFO] MFApp ..................................... SUCCESS
INFO] PageRankApp ................................ SUCCESS
INFO] TriangleCountApp .......................... SUCCESS
INFO] StronglyConnectedComponentApp ............ SUCCESS
INFO] ShortestPathsApp .......................... SUCCESS
INFO] SVDPlusPlusApp ............................ SUCCESS
INFO] LabelPropagationApp ....................... SUCCESS
INFO] ConnectedComponentApp ..................... SUCCESS
INFO] PregelOperationApp ......................... SUCCESS
INFO] SQLApp .................................... SUCCESS
INFO] StreamingApp ................................ SUCCESS
INFO] ---------------------------------------------------------------------------
INFO] BUILD SUCCESS
INFO]--------------------------------------------------------------------------------
INFO] Finished at: 2016-03-29T00:40:07-04:00
INFO] Final Memory: 76M/324M
INFO]--------------------------------------------------------------------------------
Build all done!
[root@bgnsd05 bin]#

12. Rename the enc.sh to env.sh file under the /usr/local/spark-bench-master/conf directory

13. Edit the env.sh file to update the following values:
   a. SPARK_HOME The Spark installation location
      HADOOP_HOME The HADOOP installation location
      SPARK_MASTER Spark master
      HDFS_MASTER HDFS master
   
   b. Our file looks like this:

   c. [root@bgnsb00 conf]# cat env.sh
   
   # global settings
   
   master=bgnsb01
   #A list of machines where the spark cluster is running
   MC_LIST=/YOUR/SLAVES
   
   [ -z "$HADOOP_HOME" ] && export HADOOP_HOME=/usr/local/hadoop/
   # base dir for DataSet
   DATA_HDFS=",hdfs://10.20.82.251:9000"
   DATA_HDFS=",hdfs://SparkBench",/home/`whoami`/SparkBench
   DATA_HDFS=",hdfs://10.20.82.251:8020/SparkBench"
   DATA_HDFS=",file://tmp"
   
   #Local dataset optional
   DATASET_DIR=/home/`whoami`/SparkBench/dataset
   
   SPARK_VERSION=1.5.1 #1.4.0
   [ -z "$SPARK_HOME" ] && export SPARK_HOME=/usr/local/spark
   
   #SPARK_MASTER=local
   #SPARK_MASTER=local[K]
   #SPARK_MASTER=local[*]
   #SPARK_MASTER=spark://HOST:PORT
   #SPARK_MASTER=mesos://HOST:PORT
   #SPARK_MASTER=yarn-client
   #SPARK_MASTER=yarn-cluster
Configuration to Run Spark in YARN Cluster Mode

When Spark was first installed, it was running in Standalone mode. Here are the steps we took to run it in YARN Cluster mode:

1. The Hadoop configuration files need to be accessible to the YARN client so that they can be distributed to the YARN ResourceManager. Update the env.sh file on the Spark client (bgnsb00), under the /usr/local/spark-bench-master/conf directory, to export the directories where the YARN and Hadoop configuration files are located within the Spark Client node. The following was added to the env.sh file:

   ```
   Export HADOOP_CONF_DIR=/usr/local/spark-bench-master/hadoop_conf
   Export YARN_CONF_DIR=/usr/local/spark-bench-master/hadoop_conf
   ```

2. Because we didn’t install Spark on the Hadoop HDFS node (e.g. in a production scenario, the application would not, normally, run right on the same node as the data node), we had to copy the Hadoop and YARN configuration files to the /usr/local/spark-bench-master/hadoop_conf directory located in the Spark Client node (bgnsb00). By doing this, the Spark application had the needed Hadoop/YARN configuration files and was able to proceed with its usual tasks. These are the files we copied:

   ```
core-site.xml
hadoop-env.sh
```
3. Update the config.sh file under /usr/local/spark-bench-master/bin/ on the Spark Client node to send all hdfs commands to the Hadoop NameNode (in blue):

```
#!/bin/bash
myhdfs="ssh bgnsd00 /usr/local/hadoop/bin/hdfs"
this=$( cd -P "$( dirname "$BASH_SOURCE")" )/ "pwd -P }

if [ -f "$this"../conf/env.sh ] ; then
  "$this"../conf/env.sh
else
  echo "env.sh is missing"
  exit 1
fi

export BENCH_VERSION="2.0"
[ -z "$BENCH_HOME" ] && export BENCH_HOME="$this"../"
[ -z "$BENCH_CONF" ] && export BENCH_CONF="$this"../conf/
[ -f "$BENCH_HOME"/bin/funcs.sh" ] && . "$BENCH_HOME"/bin/funcs.sh"

export BENCH_NUM=${BENCH_HOME}/num;
if [ ! -d $(BENCH_NUM) ] ; then
  mkdir -p $(BENCH_NUM);
  mkdir -p $(BENCH_NUM)/old;
fi

# local report
export BENCH_REPORT=${BENCH_NUM}/bench-report.dat

if [ -z "$MllibJar" ] ; then
  export MllibJar=~/.m2/repository/org/apache/spark/spark-mllib_2.10/$SPARK_VERSION/spark-mllib_2.10-$SPARK_VERSION.jar
fi

#RM="hdfs dfs -rm -f"
RM="$myhdfs dfs -rm -f"
#DU="hdfs dfs -du"
DU="$myhdfs dfs -du"
HADOOP_CONF_DIR="$HADOOP_CONF_DIR:-$HADOOP_HOME/conf"
#MKDIR="hdfs dfs -mkdir -p"
MKDIR="$myhdfs dfs -mkdir -p"
#CPFROM="hdfs dfs -cp "
CPFROM="$myhdfs dfs -cp "
```

Now, to run a workload, we went to the bin directory of the workload (e.g. the bin directory for the PageRank workload is /usr/local/ spark-bench-master/ PageRank/bin) and ran the gen_data.sh to generate data for this specific workload.

```
[root@bgnsb00 bin]# ./gen_data.sh
========== preparing PageRank data ==========
```

---

16/05/16 00:48:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
16/05/16 00:48:18 INFO fs.TrashPolicyDefault: Namenode trash configuration:
Deletion interval = 0 minutes, Emptier interval = 0 minutes.
...
You can see this execution on the YARN ResourceManager GUI page:

Once the data has been generated, we executed the run.sh file to run the workload:

```bash
[root@bgnsb00 bin]# ./run.sh
========== running PageRank benchmark ========== 
```

16/05/16 00:59:21 INFO RMProxy: Connecting to ResourceManager at bgnsd04/10.20.82.247:8032
16/05/16 00:59:21 INFO Client: Requesting a new application from cluster with 5 NodeManagers
...
16/05/16 01:01:27 INFO Client: Application report for application_1463003704197_0013 (state: RUNNING)
16/05/16 01:01:28 INFO Client: Application report for application_1463003704197_0013 (state: FINISHED)
16/05/16 01:01:28 INFO Client:
  client token: N/A
diagnostics: N/A
  ApplicationMaster host: 10.20.82.249
  ApplicationMaster RPC port: 0
  queue: default
  start time: 1463374778048
  final status: SUCCEEDED
  tracking URL: http://bgnsd04.fp3k.pokprv.stglabs.ibm.com:8088/proxy/application_1463003704197_0013/
  user: root
16/05/16 01:01:28 INFO ShutdownHookManager: Shutdown hook called
16/05/16 01:01:28 INFO ShutdownHookManager: Deleting directory /tmp/spark-16f8649e-8188-456c-8171-565fd292d2432
[root@bgnsb00 bin]#

The YARN GUI displayed the results:

![Hadoop YARN UI screenshot](image)

Lastly, the env.sh file under the conf directory for the specific workload contains parameters that can be changed to alter the workload data generation (e.g. the conf directory for the PageRank workload is `/usr/local/spark-bench-master/PageRank/conf`). For example, this is what the env.sh file for the PageRank workload looks like:

```
[root@bgnsb00 conf]# cat env.sh
# for preparation
numV=2000
NUM_OF_PARTITIONS=10
mu=4.0
```
sigma=1.3

# for running
MAX_ITERATION=12
TOLERANCE=0.001
RESET_PROB=0.15
SPARK_STORAGE_MEMORYFRACTION=0.48
[root@bgnsb00 conf]#

This is the env.sh file for the KMeans workload:

# for prepare
NUM_OF_POINTS=1000
NUM_OF_CLUSTERS=10
DIMENSIONS=20
SCALING=0.6
NUM_OF_SAMPLES=80000000
#SAMPLES_PER_INPUTFILE=4000000
#SAMPLES_PER_INPUT=6000000
NUM_OF_PARTITIONS=10

MAX_ITERATION=10
NUM_RUN=1
SPARK_STORAGE_MEMORYFRACTION=0.28

Those are the parameters that each workload has to tweak and obtain different results.

8. Summary
This paper documents how to do a basic install and configure of Spark and Hadoop in a clustered environment using z/VM guests on IBM System z servers. We used several Linux on z guests running Red Hat enterprise Linux 7.1 in the IBM Linux on z Test and Integration Center.

Tests were done using the Spark Standalone cluster manager and the Hadoop YARN product. An open source workload tool, SparkBench was used to test various analytic workloads at different amounts of data.

In the future, we plan to show how Ansible can be used to automate the addition or removal of nodes to the existing Spark/Hadoop cluster and the performance implications of running many lightly configured guests versus fewer guests with heavier memory and processor footprints. So, stay tuned…