MPI4Spark 0.2 User Guide

HIGH-PERFORMANCE BIG DATA TEAM
http://hibd.cse.ohio-state.edu

NETWORK-BASED COMPUTING LABORATORY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE OHIO STATE UNIVERSITY

Last revised: September 26, 2023

Contents

1 Overview of the MPISpark Project

MPI4Spark is a version of Apache Spark designed with a high-performance MPI-based Netty communication backend. It harnesses the power of modern HPC clusters supporting advanced interconnects like InfiniBand, Intel OPA, and HPE Slingshot for Big Data stacks. This project incorporates compatibility with various cluster managers, including the YARN cluster manager and Spark's standalone cluster manager.

The new YARN design in MPI4Spark v0.2 aims to enhance scalability for HPC environments. The MPI-based Netty communication backend relies on the MVAPICH2-J Java bindings library for MVAPICH. For any inquiries or feedback, please reach out to mvapich-discuss@lists.osu.edu.

2 Features

High-level features of MPI4Spark are listed below. New features and enhancements are marked as (NEW).

- Based on Apache Spark 3.3.0
- (NEW) Support for the YARN cluster manager
- Compliant with user-level Apache Spark APIs and packages
- High performance design that utilizes MPI-based communication
  - Utilizes MPI point-to-point operations
  - Relies on MPI Dynamic Process Management (DPM) features for launching executor processes for the standalone cluster manager
  - (NEW) Relies on the Multiple-Program-Multiple-Data (MPMD) launcher mode for launching executors when using the YARN cluster manager
- Built on top of the MVAPICH2-J Java bindings for MVAPICH2 family of MPI libraries
- Tested with
  - (NEW) OSU HiBD-Benchmarks, GroupBy and SortBy
  - (NEW) Intel HiBench Suite, Micro Benchmarks, Machine Learning Workloads and Micro Benchmarks on a cluster size of 32 *NodeManager* nodes
  - Mellanox InfiniBand adapters (EDR and HDR 100G and 200G)
  - HPC systems with Intel OPA and Cray Slingshot interconnects
  - Various multi-core platforms

3 Setup Instructions

3.1 Installation Pre-requisites

Please make sure that the following is done on a Linux operating system and at least Java version 8 or OpenJDK version 1.8 is being used. The MVAPICH2 library along with its Java bindings are required by MPI4Spark. Please follow the userguides below for installing and building MVAPICH2 and its Java bindings.

- MVAPICH2 2.3.7 Userguide
- MVAPICH2 Java Bindings Userguide

3.1.1 Installing MPI4Spark

1. Install Hadoop YARN:

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz

tar xvzf hadoop-3.3.4.tar.gz

export HADOOP_HOME=path/to/hadoop
```

2. Install MPI4Spark:

```
$ wget http://hibd.cse.ohio-state.edu/download/hibd/mpi4spark-0.2-x86-bin.tar.gz

$ tar xzvf mpi4spark-0.2-x86-bin.tar.gz

$ export SPARK_HOME=/path/to/mpi4spark-0.2-x86-bin
```

Copy the jar file created by the Java bindings inside of MPI4Spark. You will also have to copy the `hadoop-yarn-server-nodemanager-3.3.4.jar` jar file into Hadoop.

```
$ cp $MV2J_HOME/lib/mvapich2-j.jar $SPARK_HOME/jars/
$ cp $SPARK_HOME/jars/hadoop-yarn-server-nodemanager-3.3.4.jar
   $HADOOP_HOME/share/hadoop/yarn/
```

4 Cluster Manager Overview

4.1 MPI4Spark with Standalone Cluster Manager

MPI4Spark uses a Java wrapper program (SparkMPI.java) to launch the Apache Spark cluster. Figure 1 illustrates the steps required in launching Spark with two workers, a master, and a driver using MPI. In **Step A** 4 wrapper processes are launched on separate nodes with their respective MPI ranks. In **Step B** the Spark cluster is created where each process forks Spark processes. In **Step C** the executor processes are launched

using the DPM operation `MPI_Comm_spawn_multiple()`. The node view is illustrated under nodes A, B, C, and D.

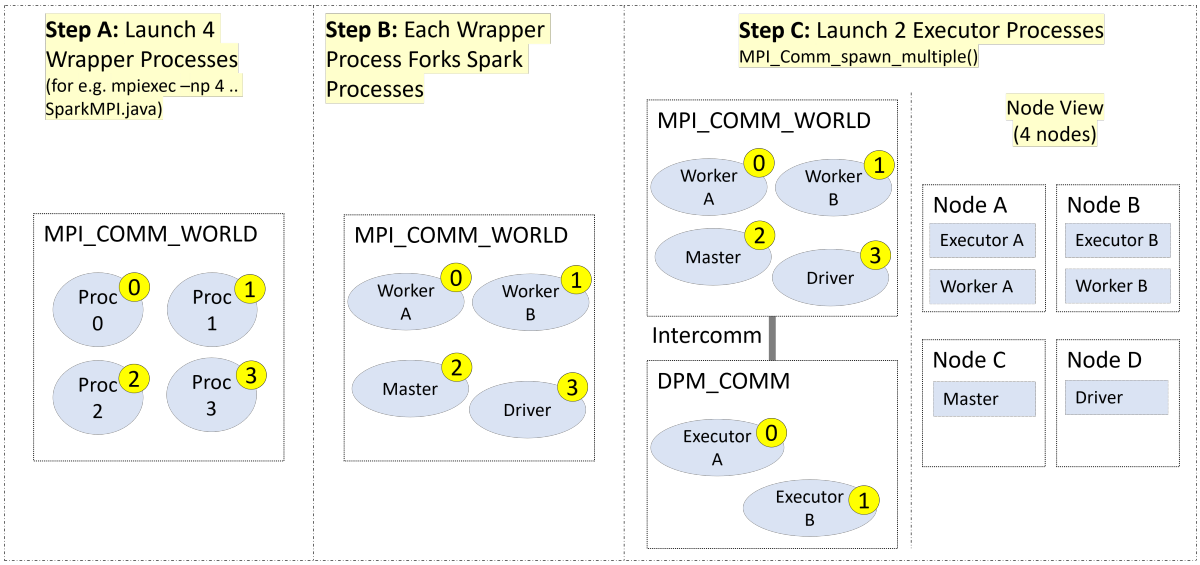

Figure 1: An example illustrating how a MPI4Spark execution is launched

4.2 MPI4Spark with YARN Cluster Manager

MPI4Spark introduces a new design when using the YARN cluster manager that launches Spark executor processes using the Multiple-Program-Multiple-Data (MPMD) launcher mode which is supported by the MVAPICH MPI launcher (i.e., mpirun_rsh). In this design, only Spark executors are launched in an MPI environment, as opposed to the entire Spark cluster, as seen when using the standalone cluster manager. Figure 2 illustrates the new YARN design and figure 3 gives an overview of the implementation.

In both figures, only executors are launched within an MPI COMM_WORLD communicator. This is different from the standalone design, where the entire cluster is launched in a COMM_WORLD communicator and later the executors are dynamically launched within a DPM_COMM.

5 Starting a Hadoop YARN cluster when using YARN Cluster Manager with Spark

When using the YARN cluster manager, we will have to configure Hadoop YARN. We will also need to add the `HADOOP_CONF_DIR` environment variable.

```
$ echo 'export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop' >> ~/.bashrc
$ source ~/.bashrc
```

Let's begin with configuring the Hadoop YARN cluster. Please make sure you have the respective `hadoop-yarn-server-nodemanager-3.3.4.jar` jar file that was made available with this version release of MPI4Spark into your Hadoop directory.

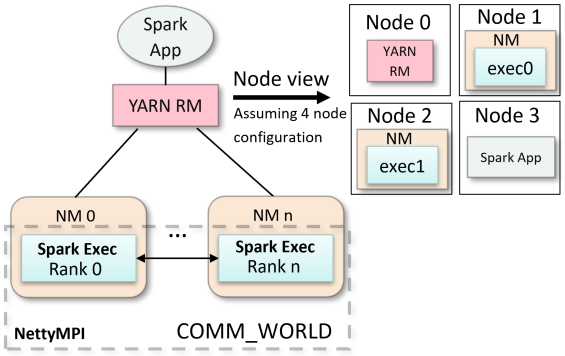

Figure 2: Design

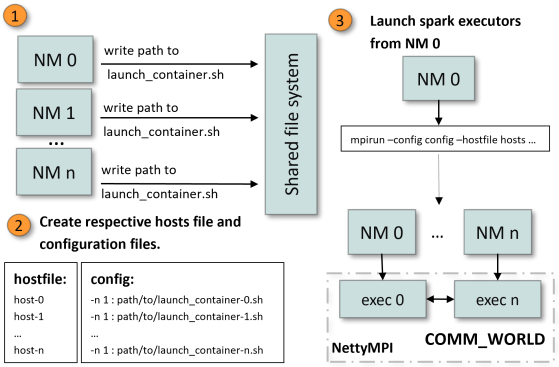

Figure 3: Implementation

Figure 4: MPI4Spark with YARN

Inside the Hadoop directory, there are several files that will need to be updated, `etc/hadoop/core-site.xml` and `etc/hadoop/yarn-site.xml`.

The contents of `etc/hadoop/core-site.xml` are the following, `MASTER_NODE` is a placeholder for the hostname the user wishes to use for setting up the `NameNode` for the Hadoop HDFS cluster.

```
<configuration>
   <property>
      <name>fs.defaultFS</name>
      <value>hdfs://MASTER_NODE:8020</value>
   </property>
</configuration>
```

The contents of `etc/hadoop/yarn-site.xml` are the following. You will have to add your respective values to each property listed here. The `yarn.nodemanager.env-whitelist` should be left unchanged.

```
<configuration>
   <property>
      <name>yarn.resourcemanager.hostname</name>
      <value>MASTER_NODE</value>
   </property>

   <property>
      <name>yarn.scheduler.maximum-allocation-mb</name>
      <value>MAX-ALLC-MB</value>
   </property>

   <property>
      <name>yarn.nodemanager.resource.memory-mb</name>
      <value>MEM-MB</value>
   </property>

   <property>
      <name>yarn.scheduler.maximum-allocation-vcores</name>
      <value>MAX-ALLC-VCORES</value>
   </property>
```

```
   <property>
      <name>yarn.nodemanager.resource.cpu-vcores</name>
      <value>CPU-VCORES</value>
   </property>

  <property>
     <name>yarn.nodemanager.env-whitelist</name>
     <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,YARN_HOME,
     SPARK_DRIVER_HOSTNAME,HOSTNAME,SPARK_HOME,SPARK_MPI_USE_YARN</value>
  </property>
</configuration>
```

The user will also have to add the following to the `etc/hadoop/yarn-env.sh` file in their Hadoop directory. Update the line below with the respective driver node hostname. For MPI4Spark, the driver node must be a separate node outside of the YARN cluster.

```
$ echo 'export SPARK_DRIVER_HOSTNAME=<DRIVER-HOSTNAME>' >> etc/hadoop/yarn-env.sh
$ echo 'export SPARK_MPI_USE_YARN=1' >> etc/hadoop/yarn-env.sh
```

The user must also update the `etc/hadoop/workers` file with a list of hostnames listing out the workers to have the `NodeManager` and `DataNode` processes launched on.

Also, make sure to clear mpmd-config directory at $`SPARK_HOME`. Once this has all been done, and to startup the Hadoop YARN cluster, simply run the following commands from MASTER NODE. Here $`HADOOP_HOME` refers to the Hadoop home directory.

```
$ rm $SPARK_HOME/mpmd-config/*
$ echo 'N' | $HADOOP_HOME/bin/hdfs namenode -format hadoop_cluster
$ $HADOOP_HOME/sbin/start-dfs.sh
$ $HADOOP_HOME/sbin/start-yarn.sh
```

6 Running Examples

6.1 SparkPi

In this subsection, we will run a simple Apache Spark application called SparkPi that computes the value of Pi. The setup for running the SparkPi application is different depending on whether or not the user is using the standalone or YARN cluster manager.

6.1.0.1 Using the Standalone Cluster Manager When using the standalone cluster manager, we will begin with setting up a hostfile. The host file will contain the hostnames for the different components in the Spark standalone cluster.

```
$ cd mpi4spark-0.2-x86-bin/

$ cat hostfile
hostname-1
hostname-2
```

```
...
hostname-n
```

MPI4Spark launches the components of its cluster in the following way:

- The worker and executor processes are launched on hostnames 1 through n-2.
- The master process is launched on hostname n-1.
- The driver process is launched on hostname n.

Considering a Spark cluster consisting of a master, two workers, and a driver, let's look at the following hostfile for an example,

```
 $ cat hostfile
 hostname-1
 hostname-2
 hostname-3
 hostname-4
```

MPI4Spark will launch the workers on hostname-1 and hostname-2 along with their respective executor processes. The master process will be launched on hostname-3 and the driver process on hostname-4. MPI4Spark relies on the client deployment mode for running applications.

Moving on, to run SparkPi, we will add a new file (app.sh) at the root of the MPI4Spark directory, and it will contain the following,

```
./bin/spark-submit --master spark://$1:7077 --class org.apache.spark.examples.SparkPi
    examples/jars/spark-examples_2.12-3.3.0-SNAPSHOT.jar
```

The app.sh file submits the application we want to run on the Spark standalone cluster. This file will be executed inside of the SparkMPI.java wrapper program.

We will now configure MPI4Spark. We will edit the template files inside of the conf/ directory.

```
$ cd conf
$ ls
fairscheduler.xml.template log4j.properties.template metrics.properties.template
    spark-defaults.conf.template spark-env.sh.template

$ cp spark-env.sh.template spark-env.sh
$ cp spark-default.conf.template spark-defaults.conf
```

Add the following inside of your spark-env.sh file. There is a variable SPARK_USE_MPI is set to 1 by default and is used to enable MPI communication. We don't have to use it specify it here since it's set to 1 by default.

```
export SPARK_HOME=path/to/sparkmpi
export SPARK_NO_DAEMONIZE=1
export JAVA_LIBRARY_PATH=$JAVA_LIBRARY_PATH:$MV2J_HOME
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MV2J_HOME/lib
export SPARK_LIBRARY_PATH=$MV2J_HOME/lib
export JAVA_BINARY=path/to/java/binary
export WORK_DIR=$SPARK_HOME/exec-wdir
```

And in your spark-defaults.conf file add the following.

```
spark.executor.extraJavaOptions -Djava.library.path=/path/to/mvapich2-j/lib
```

You can run the application now using the following command.

```
$ ./sbin/start-mpi4spark.sh
```

A number of files will be created in your directory. You will find app.log, master.log, exec.log and workers-x.log, where x refers to the number of workers. The app.log file will contain the output of running the application, and for SparkPi you should see this line printed in the file.

```
Pi is roughly 3.142532
```

For our two workers example, we will have two files, worker-0.log and worker-1.log, along with master.log, app.log, and exec.log.

6.1.0.2 Using the YARN Cluster Manager

Make sure you have the YARN cluster up and running by following the steps in 5.

Also, add the following to your $SPARK_HOME/conf/spark-env.sh file. Make sure you update `SPARK_DRIVER_HOSTNAME` with the desired respective driver node. The driver node must be a separate node outside of the YARN and HDFS cluster.

```
export SPARK_HOME=path/to/sparkmpi
export SPARK_NO_DAEMONIZE=1
export JAVA_LIBRARY_PATH=$JAVA_LIBRARY_PATH:$MV2J_HOME
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MV2J_HOME/lib
export SPARK_LIBRARY_PATH=$MV2J_HOME/lib
export SPARK_MPI_USE_YARN=1
export SPARK_DRIVER_HOSTNAME=<DRIVER-HOSTNAME>
```

and the following to your $SPARK_HOME/conf/spark-defaults.conf file.

```
spark.executor.extraJavaOptions -Djava.library.path=/path/to/mvapich2-j/lib
spark.yarn.am.extraJavaOptions -Djava.library.path=/path/to/mvapich2-j/lib
```

Now ssh to your driver node and run spark-submit. Here we are intentionally setting SPARK_USE_MPI to 0, since the driver process does not use MPI in the YARN design for MPI4Spark. Only the executors are launched within an MPI environment. Since MPI is enabled by default, we don't have to explicitly set it up for the executor processes. Make sure to clear mpmd-config directory at `$SPARK_HOME` and update the placeholder values in the setup-and-run-yarn-mpmd.sh in $SPARK_HOME/sbin/ before running the command below.

```
SPARK_MPI_USE_YARN=1 SPARK_USE_MPI=0 ./bin/spark-submit --master yarn
    --num-executors=<NUM-EXEC> --deploy-mode client --class
    org.apache.spark.examples.SparkPi
    examples/jars/spark-examples_2.12-3.3.0-SNAPSHOT.jar
```

You should see a lot of output. The following will be printed amongst it, and indicates that the application has finished running.

```
Pi is roughly 3.142532
```

6.2 OSU HiBD-Benchmarks (OHB)

OSU HiBD-Benchmarks (OHB) is a Big Data benchmark package that offers benchmarks for the Apache Spark software. These benchmarks are included in the MPI4Spark package. We will focus on two benchmarks: 1) GroupByTest, and 2) SortByTest.

6.2.1 GroupByTest

6.2.1.1 Using the Standalone Cluster Manager This benchmark groups RDD data and performs a count on the result. It forces a shuffling of data to take place across the network, and is a good litmus test for determining the performance of the shuffle phase.

We will begin with setting up the hostfile. This file contains the names of the compute nodes that will execute the given benchmark.

```
$ cd mpi4spark-0.1-x86-bin/

$ vim hostfile
.. write name of the compute nodes ..

$ cat hostfile
hostname-1
hostname-2
hostname-3
hostname-4
```

Create an app.sh file at the root of the MPI4Spark directory,

```
./bin/spark-submit --master spark://$1:7077 --class
    org.apache.spark.examples.GroupByTest
    examples/jars/spark-examples_2.12-3.3.0-SNAPSHOT.jar
```

The app.sh file submits the application we want to run to the Spark cluster. This file will be executed inside of the SparkMPI.java wrapper program.

The benchmark also allows users to specify the number of mappers, number of key-value pairs, the key size, and the number of reducers. Below is an example of how you can do that:

```
./bin/spark-submit --master spark://$1:7077 --class
    org.apache.spark.examples.GroupByTest
    examples/jars/spark-examples_2.12-3.3.0-SNAPSHOT.jar [numMappers] [numKVPairs]
    [KeySize] [numReducers]
```

Moving on, we will now configure MPI4Spark. We will edit the template files inside of the conf/ directory.

```
$ cd conf
$ ls
```

```
fairscheduler.xml.template log4j.properties.template metrics.properties.template
    spark-defaults.conf.template spark-env.sh.template

$ cp spark-env.sh.template spark-env.sh
$ cp spark-default.conf.template spark-defaults.conf
```

Add the following inside of your spark-env.sh file.

```
export SPARK_HOME=path/to/sparkmpi
export SPARK_NO_DAEMONIZE=1
export JAVA_LIBRARY_PATH=$JAVA_LIBRARY_PATH:$MV2J_HOME
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MV2J_HOME/lib
export SPARK_LIBRARY_PATH=$MV2J_HOME/lib
export JAVA_BINARY=path/to/java/binary
export WORK_DIR=$SPARK_HOME/exec-wdir
```

And in your spark-defaults.conf file add the following.

```
spark.executor.extraJavaOptions -Djava.library.path=/path/to/mvapich2-j/lib
```

You can run the application now using the following command.

```
$ ./sbin/start-mpi4spark.sh
```

A number of files will be created in your directory. You will find app.log, master.log, exec.log and workers-x.log, where x refers to the number of workers. In our example, we will have two workers, so two files will be generated. worker-0.log and worker-1.log.

6.2.1.2 Using the YARN Cluster Manager Make sure you have the YARN cluster up and running by following the steps in 5. Make sure you update SPARK_DRIVER_HOSTNAME with the desired respective driver node. The driver node must be a separate node outside of the YARN and HDFS cluster.

Also, add the following to your $SPARK_HOME/conf/spark-env.sh file.

```
   export SPARK_HOME=path/to/sparkmpi
   export SPARK_NO_DAEMONIZE=1
   export JAVA_LIBRARY_PATH=$JAVA_LIBRARY_PATH:$MV2J_HOME
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MV2J_HOME/lib
   export SPARK_LIBRARY_PATH=$MV2J_HOME/lib
   export SPARK_MPI_USE_YARN=1
   export SPARK_DRIVER_HOSTNAME=<DRIVE-NODE>
```

and the following to your $SPARK_HOME/conf/spark-defaults.conf file.

```
spark.executor.extraJavaOptions -Djava.library.path=/path/to/mvapich2-j/lib
spark.yarn.am.extraJavaOptions -Djava.library.path=/path/to/mvapich2-j/lib
```

Now ssh to your driver node and run spark-submit. Here we are intentionally setting SPARK_USE_MPI to 0, since the driver process does not use MPI in the YARN design for MPI4Spark. Only the executors are launched within an MPI environment. Since MPI is enabled by default, we don't have to explicitly set it up for the executor processes. Make sure to clear mpmd-config directory at $SPARK_HOME and update the placeholder values in the setup-and-run-yarn-mpmd.sh in $SPARK_HOME/sbin/ before running the command below.

```
SPARK_MPI_USE_YARN=1 SPARK_USE_MPI=0 ./bin/spark-submit --master yarn
    --num-executors=<NUM-EXEC> --deploy-mode client --class
    org.apache.spark.examples.GroupByTest
    examples/jars/spark-examples_2.12-3.3.0-SNAPSHOT.jar [numMappers] [numKVPairs]
    [KeySize] [numReducers]
```

6.2.2 SortByTest

This benchmark sorts RDD data and performs a count on the result. It forces a shuffling of data to take place across the network, and is a good litmus test for determining the performance of the shuffle phase.

The steps here are identical to GroupByTest, though the only difference is inside of the app.sh file. You will have to edit the app.sh file to run the SortBy benchmark. SortByTest also allows the user to specify the number of mappers, etc, using the same format as shown above in the GroupByTest example.

```
./bin/spark-submit --master spark://$1:7077 --class
   org.apache.spark.examples.SortByTest
   examples/jars/spark-examples_2.12-3.3.0-SNAPSHOT.jar [numMappers] [numKVPairs]
   [KeySize] [numReducers]
```

```
SPARK_MPI_USE_YARN=1 SPARK_USE_MPI=0 ./bin/spark-submit --master yarn
    --num-executors=<NUM-EXEC> --deploy-mode client --class
    org.apache.spark.examples.SortByTest
    examples/jars/spark-examples_2.12-3.3.0-SNAPSHOT.jar [numMappers] [numKVPairs]
    [KeySize] [numReducers]
```

6.3 Intel HiBench Benchmarks

HiBench is a Big Data benchmark suite developed by Intel that provides micro-benchmarks, ML workloads, SQL and Graph workloads for a variety of Big Data frameworks including Apache Spark.

In this section we will run the micro-benchmark Sort using MPI4Spark. Hadoop will be used for data generation. If using the Standalone cluster manager, you will want to use a regular Hadoop YARN version here, meaning we won't be using the `hadoop-yarn-server-nodemanager-3.3.4.jar` jar file we added in 5.

6.3.1 Installing and Building Intel HiBench

We will need maven for building the Intel HiBench package.

```
$ git clone https://github.com/Intel-bigdata/HiBench.git
$ export HIBENCH_HOME=path/to/HiBench
$ cd $HIBENCH_HOME
$ wget https://archive.apache.org/dist/maven/maven-3/3.8.1/binaries/apache-maven-3.8.1-
bin.tar.gz
$ tar xzvf apache-maven-3.8.1-bin.tar.gz
$ echo 'export MVN_HOME=path/to/mvn' >> ~/.bashrc
$ echo 'export PATH=$MVN_HOME/bin:$PATH' >> ~/.bashrc
$ source ~/.bashrc
$ mvn -Phadoopbench -Psparkbench -Dspark=3.0 -Dscala=2.12 clean package
```

6.3.2 Configuring HiBench

Change directory to $HIBENCH_HOME and edit conf/spark.conf and update the following variables with the desired values.

```
hibench.spark.home path/to/sparkmpi
hibench.spark.master spark://master_node:7077 (for standalone)
hibench.spark.master yarn (for yarn)
spark.executor.extraJavaOptions -Djava.library.path=/path/to/mv2j/lib
spark.yarn.am.extraJavaOptions -Djava.library.path=/path/to/mv2j/lib
```

Edit conf/hadoop.conf and set the path to the root HDFS path in order to store the generated data

```
 hibench.hdfs.master hdfs://master_node:8020/path/to/HiBenchData
```

Edit conf/hibench.conf, place your desired values for the map and shuffle parallelism parameters.

```
 hibench.scale.profile tiny
# Mapper number in hadoop, partition number in Spark
hibench.default.map.parallelism <desired-configuration>

# Reducer number in hadoop, shuffle partition number in Spark
hibench.default.shuffle.parallelism <desired-configuration>
```

6.3.3 Preparing Data

Make sure the Hadoop YARN cluster is setup here by following the steps in 5.

To prepare the data simply run the following command,

```
 $ cd $HIBENCH_HOME
 $ ./bin/workloads/micro/sort/prepare/prepare.sh
```

6.3.4 Running the Benchmark with Standalone

Copy the sparkbench.conf configuration file to your MPI4Spark directory, and update the spark path in there to reflect MPI4Spark's path. Make sure that you add the following to your conf/spark-env.sh file.

```
export SPARKBENCH_PROPERTIES_FILES=$SPARK_HOME/sparkbench.conf
```

```
$ cp $HIBENCH_HOME/report/sort/prepare/conf/sparkbench/sparkbench.conf $SPARK_HOME
```

We will add a new file (app.sh) at the root of the MPI4Spark directory, and it will contain the following,

```
$ cd $SPARK_HOME
$ cat app.sh
$SPARK_HOME/bin/spark-submit --class com.intel.hibench.sparkbench.micro.ScalaSort
    --master spark://$1:7077
    $HIBENCH_HOME/sparkbench/assembly/target/sparkbench-assembly-8.0-SNAPSHOT-dist.jar
    hdfs://$1:8020/path/to/HiBenchData/HiBench/Sort/Input
    hdfs://$1:8020/path/to/HiBenchData/HiBench/Sort/Output
```

The app.sh file submits the application we want to run to the Spark cluster. This file will be executed inside of the SparkMPI.java wrapper program.

Make sure that you update the paths for the input and output HDFS files such that they reflect your paths. Now, add a hostfile,

```
$ vim hostfile
.. write name of the compute nodes ..

$ cat hostfile
machine1
machine2
machine3
machine4
```

Add the following inside of your spark-env.sh file.

```
export SPARK_HOME=path/to/sparkmpi
export SPARK_NO_DAEMONIZE=1
export JAVA_LIBRARY_PATH=$JAVA_LIBRARY_PATH:$MV2J_HOME
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MV2J_HOME/lib
export SPARK_LIBRARY_PATH=$MV2J_HOME/lib
export JAVA_BINARY=path/to/java/binary
export WORK_DIR=$SPARK_HOME/exec-wdir
```

Finally, run the following command,

```
$ ./sbin/start-mpi4spark.sh
```

A number of files will be created in your directory. You will find app.log, master.log, exec.log and workers-x.log, where x refers to the number of workers. In our example, we will have two workers, so two files will be generated. worker-0.log and worker-1.log.

It might take some time for the command to finish. If that's the case, open a new window and tail the app.log file.

```
$ tail -f app.log
```

6.3.5 Running the Benchmark with YARN

Make sure you add `SPARK_USE_MPI=0` to the beginning of the line that runs the Spark job for the Sort benchmark in `$HIBENCH_HOME/bin/workloads/micro/sort/spark/run.sh`. For example,

```
SPARK_MPI_USE_YARN=1 SPARK_USE_MPI=0 run_spark_job
    com.intel.hibench.sparkbench.micro.ScalaSort $INPUT_HDFS $OUTPUT_HDFS
```

Also, add the following to your $SPARK_HOME/conf/spark-env.sh file. Make sure you update `SPARK_DRIVER_HOSTNAME` with the desired respective driver node. The driver node must be a separate node outside of the YARN and HDFS cluster.

```
   export SPARK_HOME=path/to/sparkmpi
   export SPARK_NO_DAEMONIZE=1
   export JAVA_LIBRARY_PATH=$JAVA_LIBRARY_PATH:$MV2J_HOME
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MV2J_HOME/lib
   export SPARK_LIBRARY_PATH=$MV2J_HOME/lib
   export SPARK_MPI_USE_YARN=1
   export SPARK_DRIVER_HOSTNAME=<DRIVER-NODE>
```

and the following to your $SPARK_HOME/conf/spark-defaults.conf file.

```
spark.executor.extraJavaOptions -Djava.library.path=/path/to/mvapich2-j/lib
spark.yarn.am.extraJavaOptions -Djava.library.path=/path/to/mvapich2-j/lib
```

You can simply just run the script. Make sure you clear out the mpmd-config directory in $SPARK_HOME before running the following,

```
$ bash $HIBENCH_HOME/bin/workloads/micro/sort/spark/run.sh
```

6.4 Stopping MPI4Spark with Standalone Cluster Manager

After each run you will have to manually stop MPI4Spark processes. This is not needed when using the YARN cluster manager. To do so, simply run the following script,

```
$ ./bin/stop-mpi4spark.sh
```

‘

7 Troubleshooting with MPI4Spark

If you are experiencing any problems with MPI4Spark, please feel free to contact us by sending an email to (mvapich-discuss@lists.osu.edu).