# Running Spark Example Applications with Spark HPC Package on a Multicore Big-Memory Machine

September 2016

Hewlett Packard Labs

The Spark-HPC package contains a native shuffle engine written in C++ and an off-heap memory store, which are built upon Retail Memory Broker that manages memory allocation from shared-memory region.

The current implementation of the native shuffle engine supports the key types of `int`, `long`, and `byte[]`. All example applications provided in Spark-HPC package can run with the native shuffle engine.

In the Spark HPC repository, under "tests/multinodes", there are several applications that include PageRank, TeraSort and a graph analytics application called Belief Propagation. The other applications include GroupBy, Join, PartitionBy, ReduceBy and SortBy are micro-benchmark applications to test out the RDD shuffle-related operators.

Each application contains a README.md to explain how to run the application in detailed steps. Note that the micro-benchmark applications require "sbt" installed.

Off-heap memory store is demonstrated in the Belief Propagation application. Both the native shuffle engine and off-heap memory store are dynamically loaded at runtime.

The example applications described in this document have been tested on HPE DL980 (8 sockets, 80 cores, 2 TB memory), HPE DL580 (4 sockets, 60 cores, 3 TB memory) HPE Superdome X (8 sockets, 240 core, 12 TB memory) and on Redhat 7.

Please refer to the companion document *Compile and Run Spark Applications with Spark HPC Package on a Multicore Big-Memory Machine* to prepare the Spark HPC package. `SPARK_HOME` refers to the directory where Spark 2.0 is downloaded from the Apache Spark web site and get fully compiled.

# 1 To Turn on/off Native Shuffle Engine

In <SPARK_HOME>/conf/spark-defaults.conf, we have a configuration parameter: `spark.shuffle.manager` which can have the value to be set as `org.apache.spark.shuffle.shm.ShmShuffleManager`, to allow the Spark application to choose the native shuffle engine. For the same application, by having the same configuration to be set as "sort" will revert the shuffle engine back to the default sort-based shuffle engine.

This configuration parameter can be changed for each application instance.

Correspondingly, we will have the two following configuration parameters to be set:

```
spark.shuffle.shm.serializer.buffer.max.mb 1024
spark.executor.shm.globalheap.name /dev/shm/nvm/global0
```

The `spark.shuffle.shm.serializer.buffer.max.mb` specifies the native memory buffer used for shuffle, and `spark.executor.shm.globalheap.name` specifies the global heap (or called shared memory region) that intermediate shuffle data will be stored.

# 2 To Make Use of Off-Heap Memory Store

For the application such as Belief Propagation that makes use of off-heap memory store to improve memory efficiency and speed up graph processing, in <SPARK_HOME>/conf/spark-defaults.conf, we

have a configuration parameter: `spark.offheapstore.globalheap.name` to be set as `/dev/shm/nvm/global0`. Typically in our experiment, we set the global heap to be the same as the one used for shuffle engine.

# 3 Configuration Parameters to Control Jetty/HTTP Server Thread Numbers

The Jetty package is implemented in such a way that the Jetty server thread pool size is calculated based on the number of the cores that the process can observe. For a large multicore machine, the number of the server threads can be excessive. However, via our virtual node deployment model, each worker process or executor process is deployed with their CPU and processor bound to the corresponding socket and thus, the number of the server threads should be based on the number of the cores the worker/executor process has been bound to.

The corresponding class in Spark 2.0 has been updated so that the number of the server threads that get launched will be based on the specification of the following parameters.

- `spark.rpc.netty.dispatcher.numThreads` – This parameter already exists in Spark 2.0.
- `spark.jetty.httpserver.numThreads` – This parameter is introduced to force the Jetty's HTTP server threads to be set, instead of based on the default setting policy that is based on the total number of the cores on the machine.

The setting of the above two parameters appears in <SPARK_HOME>/conf/spark-defaults.conf, as well as in conf/spark-defaults.conf of the master virtual node and each of the worker nodes. We can set each of these two parameters to be with the value that is the same as the number of the CPU cores that the worker node (and thus the executor process on the worker node) will be bound to. We can set the driver application and master node to have the same thread numbers as the worker node.

# 4 Example Applications

In each example directory, there is a README.md that details the steps required to run the example.

All of the example applications rely on Spark's default file partitioning scheme to partition the tasks to process the corresponding input files. One exception is TeraSort, in which we will need to have user-defined file partitioning scheme to override the default one. To do so, $SPARK_HOME/conf/spark-env.sh will need to have the following line:

```
HADOOP_CONF_DIR=<SPARK_HOME>/conf
```

And provide the file "hdfs-site.xml" under $SPARK_HOME/conf, with the following XML content:

```
<configuration>
  <property>
    <name>dfs.block.size</name>
    <value>1073741824</value>
    <description></description>
```

```
  </property>
  <property>
    <name>mapred.min.split.size</name>
    <value>1073741824</value>
    <description></description>
  </property>
  <property>
    <name>mapred.max.split.size</name>
    <value>1073741824</value>
    <description></description>
  </property>
</configuration>
```

Note that since we only need such Hadoop configuration parameters for TeraSort, to other example applications, we should avoid having this configuration file "hdfs-site.xml" under $SPARK_HOME/conf.

With respect to memory setting, for all of the example applications, we can set:

- `worker_memory = 96g`
- `driver_memory = 24g`
- `executor_memory = 96g`