Compile and Run Spark Applications with Spark HPC Package on a Multicore Big-Memory Machine

September 2016

Hewlett Packard Labs

Table of Contents

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 installation steps described in this document has been tested on Redhat 7.

1 Spark Core 2.0 Compilation

To compile the Spark core 2.0, download Spark 2.0 source code from the Apache Spark web site to the directory, which is called `SPARK_HOME` in this document. Then issue the following command at `SPARK_HOME`,

```
mvn -Phadoop-2.4 -Dhadoop.version=2.4.0 -DskipTests clean package
```

If to compile with YARN,

```
mvn -Pyarn -Phadoop-2.4 -Dhadoop.version=2.4.0 -DskipTests clean
package
```

Please make sure to choose maven 3.3.9 and Java JDK with version (at least 1.7.0_51).

In order to install Spark-HPC in Section 5, we need the local maven installation for spark-core (we have made modification at source code to fix Jetty HTTP server thread number issues with pull-request pending), the command with additional installation step is the following:

```
mvn -Phadoop-2.4 -Dhadoop.version=2.4.0 -DskipTests clean package
install
```

The installation will create under the local .m2 repository the spark-core JAR file, and the `pom.xml` that lists all of the dependent JAR files for the spark-core. Later when the application includes the dependency to the spark-core, Maven will automatically pull the dependent JAR files from the internet that the spark-core JAR file is dependent on.

2 Install Prerequisite Packages

The scripts to install the prerequisite packages for Spark-HPC reside under the directory `deploy/dependent-modules`.

2.1 Edit spark-inventory.yml

Add the correct machines under the `[my_nodes]` heading, one per line.

2.2 Edit spark-vars.yml

The following variables contain the list of applications and libraries respectively, to be installed on the server via YUM. If you want to install any additional YUM packages, you can add them to these variables:

- `yum_applications_to_install`
- `yum_libraries_to_install`

2.3 Install Ansible

The Spark-HPC installation scripts require the Ansible package to be installed on your machine. Use the following command to install it:

```
sudo yum install ansible
```

If the 'ansible' package is not available in the YUM repositories configured for your machine, you may need to add the EPEL repository. On a CentOS machine, for example, this is done as follows:

```
export http_proxy='http://YOUR_PROXY_SERVER:PORT'

curl  http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-6.noarch.rpm > epel-release-7-6.noarch.rpm

sudo rpm -iUvh epel-release-7-6.noarch.rpm
```

The step on setting http proxy is needed only when your machine is sitting behind the firewall. At this point, you should be able to install the 'ansible' package to your machine.

If your machine accesses the Internet via a proxy server, you will need also to configure YUM to use the proxy. Edit the file `/etc/yum.conf` and add a line

```
proxy=http://YOUR_PROXY_SERVER:PORT
```

2.4 Set up passwordless SSH connection to localhost

If you don't have a HOME/.ssh/id_rsa public/private key pair files for your account, you need to create it via the command:

```
ssh-keygen
```

Select `HOME/.ssh/id_rsa` as file to be generated, and empty password.

If you already have a `$HOME/.ssh/id_rsa` file, you may need to remove the passphrase from it:

```
ssh-keygen -p -P ''
```

Once the `$HOME/.ssh/id_rsa` file exists and the passphrase is removed, you need to add its associated public key to the list your authorized keys on the machine where you will install Spark-HPC:

```
cat ~/.ssh/id_rsa.pub | tee -a ~/.ssh/authorized_keys
```

Verify that connecting with your own local user account to the local machine doesn't ask for password or SSH key passphrase:

```
ssh localhost
```

Alternately, you can use the `ssh-agent` command to achieve the same objective. See the SSH documentation for details.

2.5 Install Prerequisites

Under `deploy/dependent-modules`, you can use the Ansible-powered script to install the pre-requisites:

```
./install-pre-reqs.sh
```

2.6 Install Boost

The Spark HPC package requires Boost library with version of 1.5.6 or higher.

If your machine already has an existing Boost library installed, but it is with version lower than 1.5.6, you will have to uninstall the old package first by issuing the following commands in order to use the newer version.

```
sudo yum install yum-utils

package-cleanup --leaves --all | grep boost

sudo yum erase boost-*
```

To install Boost library, under `deploy/dependent-modules`, run the script on the Spark-HPC machines where Boost is to be installed:

```
sudo ./install-boost.sh
```

Note: You may need to run the '`wget`' command by hand in case the connection to SourceForge has server-side problems.

```
wget
https://sourceforge.net/projects/boost/files/boost/1.56.0/boost_1_56_0
.tar.gz

mv boost_1_56_0.tar.gz /tmp

sudo ./install-boost.sh
```

2.7 Export Environment Variable: LD_LIBRARY_PATH

After installing all these prerequisite libraries, make sure that **LD_LIBRARY_PATH** has these installation paths (/usr/local/lib) by issuing

```
export LD_LIBRARY_PATH=/usr/local/lib
```

3 Build Retail Memory Broker

Retail Memory Broker (RMB) is responsible for shared memory management. In the Spark-HPC repository, the code is located under “alps” directory.

In the “alps” directory, there is a “README.md” to describe how to build and the list of dependent libraries required.

To start, create a sub-directory called “build” under “alps”. In the created “build” directory, issue:

```
cmake ..  -DCMAKE_BUILD_TYPE=Release
```

then issue in the same "build" directory:

```
make
```

To build the entire code base. You can also issue "`make VERBOSE=1`" to check that the build process really invokes g++ compiler with "-O3" compilation flag.

To speed up compilation process, you can also issue "`make -j 8`" to invoke multithreaded compilation.

Once the build is completed successfully, the shared library "libalps.so" is located under "./build/src" directory. Copy this runtime to a shared-library installation location, such as /usr/local/lib.

The shared-library "libalps.so" requires at least boost library with version 1.5.6.

In order to check that all of the dependent shared libraries have been resolved correctly, issue:

```
ldd ./build/src/libalps.so
```

Instead of specifying LD_LIBRARY_PATH, the alternative way to force the boost library (or other dependent libraries) to be loaded from /usr/local/lib rather than the one in the default /usr/lib or /usr/lib64, , is to create "boost.conf" under /etc/ld.so.conf.d, and in this file, specify /usr/local/lib to be the loading path.

At the end of the build process, the RMB utility tool called "globalheap-util" is built under "./build/src/globalheap".

Finally, to install of the shared library and the RMB utility tool globalheap-util to a system-wide directory, issue under the "build" directory:

```
sudo make install
```

The result is that "libalps.so" and "globalheap-util" will be installed to /usr/local/lib and /usr/local/bin correspondingly.

Once the installation steps above are completed, check by invoking "which globalheap-util", which should lead to the answer that the utility is located under /usr/local/bin.

4 The Utility Routines Provided by Retail Memory Broker

After the build and installation process of Retail Memory Broker described in Section 3, we can then invoke the Retail Memory Broker utility called: globalheap-util, which is located in /usr/local/bin.

On the big-memory machine that uses tmpfs to back the shared-memory, before we create a shared-memory region, two steps need to be done first, under the current user account (that is, the account that is to launch Spark and use shared-memory region managed by Retail Memory Broker):

(1) issue: `touch /dev/shm/@@lockfile@@`, to create a global lock file. (Note: `@@lockfile@@` should be in `/dev/shm` even if the global heap is located in a different directory)

(2) issue: `mkdir /dev/shm/nvm`, to create a directory under the default tmpfs called /dev/shm. The user is allowed to create mount points with tmpfs, as well as the path is ended with "nvm".

To create a shared-memory region under /dev/shm, issue:

```
globalheap-util --loglevel=info create /dev/shm/nvm/global0 --size=32G
```

"--loglevel=info" is optional. Note that before you create the global heap, make sure to remove all files in "/dev/shm/nvm" or run:

```
globalheap-util remove /dev/shm/nvm/global0
```

The total memory size specified via "--size" should be multiple of 8 GB's, as the zone size that is defined to be 8 GB.

To report the usage of the created shared memory region, issue:

```
globalheap-util report /dev/shm/nvm/global0
```

Since there can be multiple of 8 GB zones in the region, a finer report can be found by issuing:

```
globalheap-util report /dev/shm/nvm/global0 --perzone=true
```

to display the per-zone memory consumption.

To format the shared-memory region after some job run, issue:

```
globalheap-util format /dev/shm/nvm/global0
```

There is no need to format the global heap after each job run, if it can be sure that there is sufficient shared-memory for next job run's shuffle data.

To remove the shared-memory region:

```
globalheap-util remove /dev/shm/nvm/global0
```

5 Spark-HPC Package Compilation

In the Spark HPC repository, after the retail memory broker is compiled and deployed, at the top "firesteel" directory, issue:

```
mvn -DskipTests -Dmaven.test.skip=true -Dmaven.site.skip=true clean
package
```

The outcome of the compilation is the JAR file located at firsteel/target/firesteel-2.0.0-SNAPSHOT.jar. This jar file contains both the shuffle engine and the off-heap memory store.

The above compiled firesteel JAR package needs to be installed to the local .m2 repository in order for the other Spark applications that depends on this firesteel JAR package to be correctly compiled, the complete mvn command is the following:

```
mvn -DskipTests -Dmaven.test.skip=true -Dmaven.site.skip=true clean
package install
```

The command:

```
jar xvf ./target/firesteel-2.0.0-SNAPSHOT.jar
```

can be issued on a temporary directory, to inspect whether the two shared libraries are packaged correctly: “libjnishmoffheapstore.so” and “libjnishmshuffle.so”. These two shared libraries should be located at the top of the directory where the above “jar xvf” command is issued.

Note that current firesteel build is dependent on the spark-core package that has been installed on local .m2 repository. Therefore, we will need to have “mvn install” step described in Section 1 to allow spark-core 2.0 to be installed on the local .m2 repository.

After the successful full package compilation is completed, we need to go down to “firesteel/build” directory, to issue:

```
sudo make install
```

This installation will install “libshm_management.so” to /usr/local/lib. This is the shared library that the shuffle engine and off-heap memory store are dependent on. We need “sudo” as the deployed shared-library is at /usr/local/lib.

To make sure that “libshm_management.so” is linked to the correct dependent shared library packages such as **tcmalloc**, **boost** and the Retail Memory Broker **alps**, we can use:

```
ldd libshm_management.so
```

to check whether the linkage is correct.

6 User Shell Environment Variables

At this time, the entire Spark HPC package and their dependent packages have been installed to the machine successfully, the shell environment of the user that is to launch Spark application on the machine will need to have the following variables set.

- `JAVA_HOME` – the directory where Java JDK is installed.
- `LD_LIBRARY_PATH` – to specify the shared library loading path. Since ALPS and firesteel runtime libraries are installed by default at `/usr/local/lib`, we should have `/usr/local/lib` to be added to this environment.

7 Creating Spark-HPC Virtual Node Cluster on a Single Multicore machine

On a multicore big-memory machine such as HPE Superdome X (8 sockets, 240 cores) and DL580 (4 sockets, 60 cores), we need different Spark Executors bound to different sockets in terms of CPU and memory. This section explains the installation procedure on how to set up a virtual node cluster to run Spark application on a single multicore machine with CPU/memory binding enforced. A virtual node cluster emulates a multi-node Spark cluster on a single machine. Each virtual node will have:

- A node directory.
- A set of cores, in the same CPU socket.
- A working memory residing local to the socket that contains the specified cores.

7.1 Virtual Nodes-Related Files within the Spark-HPC Distribution

In the Spark HPC repository, under the directory of "`deployment/virtualnode`", which contains the following subdirectories:

- `playbooks` – installation scripts, powered by Ansible
- `scripts` – auxiliary installation scripts
- `templates` – templates for configuration files

7.2 User-Defined Variables

The file `playbooks/variables.yml` contains a list of variables that will guide the Spark-HPC installation. Some important variables are:

- `install_root` – the root directory where the virtual nodes will be installed. This will be referred to as the `<INSTALL_ROOT>` for the multi-node environment.
- `num_workers` – number of virtual worker nodes to create
- `worker_memory` – how much memory capacity to be assigned to each worker
- `executor_memory` – how much memory to be assigned to each executor
- `driver_memory` – how much memory to be assigned to the driver application
- `work_dir` – directory where the temporary files of the Spark master and workers will be stored.
- `master_tmpfs_size` – The size in GB of the work directory for the master virtual node.
- `worker_tmpfs_size` – The size in GB of the work directory for the worker virtual node.

The value setting for the above variables have been already populated in `variables.yml`. We can just make modification based on the actual machine configuration. For example, on an eight-socket HPE DL-980 machine with 2TB memory and 80 core (each socket containing 10 cores), we can set:

```
install_root = /var/tmp/spark2.0hpcplatform

num_workers = 8

worker_memory = 64g

executor_memory = 48g

driver_memory = 24g

work_dir = /var/tmp

master_tmpfs_size = 10g

worker_tmpfs_size = 100g
```

Note that `executor_memory` will have to be no larger than `worker_memory`.

7.3 Templates

The `templates` directory contains the templates of the Spark configuration files that will be deployed during the installation. These files will be customized based on the values set to the variables described above.

The following directories are found under `templates`:

- `master` – Configuration files for the Spark master node
- `worker0` – Configuration files for the Spark worker nodes
- `scripts` – Utility scripts that will be installed under the Spark installation directory
- `spark` – Configuration files to be deployed under the `SPARK_HOME` directory provided by the user. Note that some configuration files under this directory will be overwritten by the installation, and a backup copy saved.

7.4 Virtual Node Cluster Creation

After reviewing the above variables, and making any necessary changes to the configuration files templates (for example, to add a Spark configuration parameter), you can start the following virtual node creation steps.

Before you begin the installation, you need to ensure that you can connect to the machine's localhost and main IP address as root via password-less SSH. The following command should not require you to enter a password:

```
ssh localhost
```

You will already have installed your preferred version of Spark under the `SPARK_HOME` directory. Note that the installation of the virtual node cluster will overwrite some files under `<SPARK_HOME>/conf`, in particular, spark-defaults.conf and spark-env.sh. A backup version will be saved by the installation scripts.

The install scripts use Ansible. You will need to install it on your machine via:

```
sudo yum install ansible
```

and follow Section 2.4 to install the pre-requisite libraries and OS packages.

7.4.1 Installing Virtual Node Environment

After the prerequisites have been successfully installed, you can set up the multi-node environment via the following command. In the Spark HPC repository, under the directory of "`deployment/virtualnode`":

```
cd playbooks/
./install-spark.sh <SPARK_HOME>
```

Where `<SPARK_HOME>` is the directory where you installed the version of Spark downloaded from the Apache Spark website, by following Section 1.

This program will create the multi-node environment under the directory indicated in the variable `install_root` in `variables.yml`.

7.4.2 Copy the SPARK-HPC JAR File to the Virtual Node Environment

After installing the virtual node environment, you will need to copy the SPARK-HPC JAR file, which is compiled by following the steps described in Section 5. Under the directory `<INSTALL_ROOT>/lib`, where `<INSTALL_ROOT>` is the value of the variable `install_root` in the file `variables.yml`.

```
sudo cp firesteel-….jar <INSTALL_ROOT>/lib
```

7.4.3 Set Up In-memory Filesystems Required by Virtual Nodes

For best performance, the working directories for each virtual node (either the master or a worker), need to reside on in-memory filesystems local to the socket assigned to the corresponding virtual node. To create these filesystems, run:

```
cd <INSTALL_DIR>

./create_filesystems.sh
```

Where `<INSTALL_DIR>` is the root install directory where Spark-HPC was installed.

This will create filesystems under the directory pointed by the `work_dir` variable. To view the created file system mount points, issue: "df –h". An example of this command output is the following:

```
Filesystem      Size  Used Avail Use% Mounted on
…
tmpfs            10G     0   10G   0% /var/tmp/WORK_DIRS2/store-m0
tmpfs           100G     0  100G   0% /var/tmp/WORK_DIRS2/store-w0
tmpfs           100G     0  100G   0% /var/tmp/WORK_DIRS2/store-w1
tmpfs           100G     0  100G   0% /var/tmp/WORK_DIRS2/store-w2
tmpfs           100G     0  100G   0% /var/tmp/WORK_DIRS2/store-w3
…
```

The report above corresponds to `master_tmpfs_size =10 GB` and `worker_tmpfs_size = 100 GB` that are specified in Section 7.2.

7.4.4 Structure of the Created Virtual Node Cluster

Under the installation directory `{install_root}`, a directory `multicore` is created with the following sub-directory structure:

- `master`: virtual node for the Spark master
- `worker0 … worker`***`N-1`***: virtual nodes for the N workers as defined by the `num_workers` variable

Each virtual node (the master or a worker node) will have the following sub-directories:

- `assembly` – a symbolic link to the `assembly` directory under `spark-2.0`
- `bin` – a copy of the `bin` directory under `spark-2.0`
- `conf` – configuration files for the virtual node
- `logs` – log files generated by the virtual node
- `pids` – files that contain the process ids to keep track of the processes launched on the virtual node
- `sbin` – a copy of the `sbin` directory under `spark-2.0`, with some scripts changed for virtual node operation
- `tools` – a symbolic link to the `tools` directory under `spark-2.0`

7.4.5 Launching the Virtual Node Cluster with Spark-HPC

At this point you can start the Spark-HPC master and worker nodes that run with Spark 2.0, using the command:

```
cd <INSTALL_DIR>
sudo ./start-spark.sh
```

This will start a master process, plus N worker processes, in the virtual node.

You can check that the processes are running using the command:

```
jps
```

The output should show one master process and N worker processes.

Once the launching of the virtual node cluster is done, you can use a web browser to point to the Spark UI monitoring URL to check whether all of the workers associated with the virtual nodes have been launched successfully.

7.4.6 A Simple Testing of the Created Virtual Node Cluster

To verify that virtual node for Spark-HPC is working correctly, you can run the following:

```
cd <INSTALL_DIR>/examples
./test-pi.sh 2>/dev/null
```

This script should display a message of the form:

```
Pi is roughly 3.14368
```

7.5 Additional Useful Commands

Stopping Spark cluster launched in a virtual node:

```
cd <INSTALL_DIR>
sudo ./killJava.sh
```

Remove virtual node log files,

```
cd <INSTALL_DIR>
sudo ./clean-up.sh
```

Remove all of the in-memory file systems created in a virtual node cluster,

```
cd <INSTALL_DIR>
sudo ./delete_filesystems.sh
```

Remove worker temporary files:

```
sudo find <WORK_DIR> -mindepth 3 -delete
```

Where `<WORK_DIR>` is the work directory pointed by the variable `work_dir` in `variables.yml`.

To test the launching of the virtual node cluster, we can test only by one single virtual node:

```
cd <INSTALL_DIR>
sudo ./start-one.sh
```

To test only by two single virtual nodes:

```
cd <INSTALL_DIR>
sudo ./start-two.sh
```