Running Hadoop Programs on ambari-head.csc.calpoly.edu Cluster

Overview

This document contains information that is specific to running Hadoop programs on our local cluster.

CSLVM Hadoop Cluster

At the moment `hadoop` is installed on the `ambari-head.csc.calpoly.edu` cluster, which besides `ambari-head` contains four more nodes: `ambari-node01`, `ambari-node02`, `ambari-node03`, `ambari-node04`.

All your work with the cluster will take place on `ambari-head.csc.calpoly.edu`.

To successfully run Hadoop jobs on the cluster, you need to know the following.

Location of Hadoop Binaries. The `hadoop`, `hdfs`, and `yarn` binaries are located in `/usr/bin`.

Our `Hadoop` installation is version 2.7.3

Java Hadoop Package. Core Hadoop library is available in the `hadoop-core-1.2.1.jar` jar file. The file can be downloaded from the course web page:

http://users.csc.calpoly.edu/~dekhtyar/369-Winter2019/code/hadoop-core-1.2.1.jar

Note: This file may be out of date. We will provide a new `org.apache.hadoop` implementation once we discover it.

Compiling Hadoop code. To compile a Hadoop job located in file `myJob.java`, place the `hadoop-core-1.2.1.jar` file in the directory where `myJob.java` is located and run the following command from that directory:

```
$ javac -cp hadoop-core-1.2.1.jar myJob.java
```

Note: To stop prolifiration of the `hadoop-core-1.2.1.jar` I recommend creating a ∼`userid/jars` directory, placing the jar file there, and adding the `jars` directory to the classpath.

Making a Jar of your Hadoop Job. Hadoop allows running `.class` files only on local intallations. Our `Hadoop` is installed as a three-node cluster. On such installations, all Hadoop jobs must be packaged as jars.

To create a jar for your Hadoop job, run the following command (after compiling the code):

```
$ jar cvf myJob.jar *.class
```

Alternatively, you can create a simple `manifest.txt` file with the content:

```
Main-Class: myJob
```

and create a jar file using the command:

```
$ jar cvf myJob.jar manifest.txt *.class
```

Removing `.class` files. In some situations your jar files won't execute correctly unless you remove the `.class` files after creating the jar file.

Running your Hadoop Jobs. Once you created the jar file, you can run it as follows.

```
$ hadoop jar myJob.jar myJob <hadoop job arguments>
```

The last argument of the command *before* `<hadoop job arguments>` is the Java class that contains the `public static int main()` method. If you created your jar file with a `manifest.txt` file as as discussed above, you can omit the Java class name:

```
$ hadoop jar myJob.jar <hadoop job arguments>
```

`<hadoop job arguments>` are any of the command-line arguments you need to pass to the `public static int main()` of your program. Often these are the locations of the input files and output directory.

Alternative way to compile and run Hadoop jobs

This is a **better way**. It avoids having to use `org.apache.hadoop` jars, but it does require some extra settings in your `.bashrc` file.

Basic idea. `hadoop` is a launcher of JVMs. It can be used to launch **any** java process, including `javac`. When java compiler is launched this way, the `org.apache.hadoop` package is available at compile time without the need to explicitly include it into the classpath. This requries some setup though.

Please note: the instructions below work on the `cslvm57` (and will work on `cslvm56, cslvm55` if you are given access to those) specifically.

Step 1. Setting up environment. Add the following commands to your `.bashrc` file:

```
export JAVA_HOME=/usr/jdk64/jdk1.8.0_112
export PATH=${JAVA_HOME}/bin:${PATH}
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

After that, execute (from your home directory)

```
$ source .bashrc
```

This will immediately create these variables and will allow you to continue work. `.bashrc` runs automatically upon every login, so from then on, you won't need to worry about these settings.

Step 2. Compilation and creation of jar. The reason why we created `$HADOOP_CLASSPATH` variable and pointed it to Java's `tools.jar` file is because it contains `javac` in it! The Java compiler resides in the following class:

```
com.sun.tools.javac.Main
```

We can run it through `hadoop` as follows:

```
$ hadoop  com.sun.tools.javac.Main myJob.java
```

This results in compilation of `myJob.java` with `org.apache.hadoop` classes present. We can now create the jar as before:

```
$ jar cvf myJob.jar *.class
```

Step 3. Execution. The final step is the same as above:

```
$ hadoop jar myJob.jar myJob <inputParameters>
```

Additional Information

Monitoring your job. Hadoop provides a web service allowing one to monitor the status of Hadoop jobs via a browser. Because cslvm55 does not have pinholes set, this is only possible if you are currently on campus, or running a campus VPN client on your machine.

The URL for the Hadoop monitor is

http://ambari-head.csc.calpoly.edu:8088/cluster

Third party Jars. For more complex Hadoop jobs you often may need third-party Jar files to be used with the `Mapper` and/or `Reducer` code. Because the `Mapper` and `Reducer` jobs run in separate JVMs on the cluster, their classpath environments are different than the classpath envoronment of the `public static int main()`. Fortunately, Hadoop allows us to pass third-party jars to the cluster JVMs using the `-libjars` option.

The full command to run a Hadoop job from myJob.jar file that relies on a third-party jar file `foo.jar` and another third party jar file `bar.jar`:

```
$ hadoop jar myJob.jar myJob -libjars foo.jar,bar.jar <hadoop job arguments>
```