CS 348 - Project 3

SQL on Spark!

Step 1: Setting up Hadoop environment

Overview

This handout will demonstrate how to setup HDFS and implement a simple MapReduce job, which can be run on files stored in HDFS. Step 1 will not be graded.

***** Please be mindful of spaces and special characters specifying folder contents such as / or *. This can save you a lot of time by reducing errors. *****

Environment

NOTE: **Purdue Career Account User ID** will be represented here as `$USER` .

We will be using Purdue CS department's Spark cluster for this setup. The master node for this cluster is `jacobi00.cs.purdue.edu`. You can `SSH` into the system through `data.cs.purdue.edu` or through the terminal of the machine that you have currently logged in.

Log in to `data.cs.purdue.edu`,

```
$ ssh $USER@data.cs.purdue.edu
```

Log in to `jacobi00.cs.purdue.edu` (password is the same as your Purdue career account),

```
$ ssh jacobi00
```

If you are connecting from off campus, you may need to set up a VPN connection first. Instructions for setting up a VPN to Purdue's network are here: https://www.itap.purdue.edu/connections/vpn/.

To see a list of all nodes in the cluster, run

```
$ cat /etc/hosts
```

CAUTION: This cluster is temporary. It will be wiped after the project is graded. If you have any code or results that you wish to save, move them to permanent storage on another system.

Basic commands for HDFS

Before you begin project 3, you should know how to:

1. Move files from the cluster master node to HDFS and back.
2. Compile and run a simple MapReduce job. (This is just to see HDFS in action. It is not necessary to know MapReduce for project 3).

If you have used Hadoop before, you may be familiar with these concepts. If you have not, follow the steps in this section.

After logging on to the cluster master node with `SSH`, populate a simple text file:

```
$ printf "aaa\nbbb\nccc\nddd\naaa\nbbb\nccc\nddd\n" > tmp.txt
```

Important Create a your personal HDFS directory:

```
$ hdfs dfs -mkdir /user/$USER
```

Create a directory in your personal HDFS directory to store the file:

```
$ hdfs dfs -mkdir /user/$USER/in
```

Copy the text file from the master node to the new directory:

```
$ hdfs dfs -put ./tmp.txt /user/$USER/in
```

List the directory contents:

```
$ hdfs dfs -ls /user/$USER/in
```

Now that we have loaded a file into HDFS, we would like to run a MapReduce job over it. We have given you `Select.java` with Project 3 handout. Copy the file `Select.java` from your personal machine to the cluster master node. On the master node, navigate to the directory which holds `Select.java` and run the following commands:
Be mindful of spaces and / and * characters

```
$ mkdir select
$ CP=$(hadoop classpath)
$ javac -classpath $CP -d select/ Select.java
$ jar -cvf select.jar -C select .
$ hadoop jar select.jar org.myorg.Select /user/$USER/in /user/$USER/out
```

Notice the last two arguments to the final command. Our MapReduce job will take these arguments to be the source directory (which already exists) and the destination directory (which will be created by our MapReduce job).

Next, check the output:

```
$ hdfs dfs -ls /user/$USER/out
```

You should see an output like this with `SUCCESS` message:

```
Found 2 items
-rw-r--r--   1 $USER supergroup          0 2019-10-15 17:10 /user/manig/out/_SUCCESS
-rw-r--r--   1 $USER supergroup         16 2019-10-15 17:10 /user/manig/out/part-00000
```

To see the contents of the output, run

```
$ hdfs dfs -cat /user/$USER/out/*
```

Sometimes it may not display anything. In that case, use the following:

```
$ hdfs dfs -cat '/user/$USER/out/*'
```

Move the output from HDFS back to your directory on the master node:

```
$ hdfs dfs -getmerge /user/$USER/out ./output.txt
$ cat output.txt
```

Finally, remove the output directory so that you can run your job again. (Hadoop will complain if you try to write to an HDFS directory that already exists.)

```
$ hdfs dfs -rm -r /user/$USER/out
```

(`-rm -r` is used for removing directories in HDFS.)

Dataset

For Project 3, we will be using Movie Lens 1M dataset, taken from GroupLens Research: https://grouplens.org/datasets/movielens/1m/.

CAUTION: There are several datasets listed on the Movie Lens webpage. Be sure you get the 1M dataset.

Download the data:

```
$ curl -O http://files.grouplens.org/datasets/movielens/ml-1m.zip
```

Unzip the folder:

```
$ unzip ml-1m.zip
```

View the contents of the extracted folder:

```
$ cd ml-1m
$ ls
```

Tables:

```
users.dat  ( UserID::Gender::Age::Occupation::Zipcode )
movies.dat  ( MovieID::Title::Genres )
ratings.dat ( UserID::MovieID::Rating::Timestamp )
```

NOTE: Some of the text in `movies.dat`, `users.dat`, `ratings.dat` contain characters that are not UTF-8. These characters may compromise your results. So, as is often the case in the real world, we must put our data through a "cleaning" phase before we query it. Use the following bash command to remove all non-UTF-8 characters from the `movies.dat`, `users.dat`, `ratings.dat` files before you load it into HDFS:

```
$ iconv -f utf-8 -t utf-8 -c movies.dat > movies_utf8.dat
$ iconv -f utf-8 -t utf-8 -c users.dat > users_utf8.dat
$ iconv -f utf-8 -t utf-8 -c ratings.dat > ratings_utf8.dat
```

Copy these formatted files to your HDFS input folders (here, we assume that you are in recently extracted `ml-1m` directory):

```
$ hdfs dfs -put movies_utf8.dat /user/$USER/in
$ hdfs dfs -put userss_utf8.dat /user/$USER/in
$ hdfs dfs -put ratings_utf8.dat /user/$USER/in
```

We have completed step 1 of using HDFS for storing files and running a MapReduce job. We will be using HDFS and the same dataset for the Spark project. We have also cleaned the data for usage.