ISIT312 Big Data Management

Session 2, 2022

Exercise 2

Programming MapReduce Applications in Java

In this exercise, you will get familiar with programming of simple MapReduce applications.

Be careful when copying the Linux commands in this document to your working Terminal, because it is error-prone. Maybe you should type those commands by yourself.

Prologue

Login to your system and start VirtualBox.

When ready start a virtual machine `ISIT312-BigDataVM-07-SEP-2020`.

When the virtual machine is running, leftclick at Terminal icon to open Terminal window.

In this practice you can either use Zeppelin interface described in the previous practice or use can use command line interface in Terminal window. If you plan to use Zeppelin then perform the following actions.

(1) Start Zeppelin and create a new notebook named, say, `Lab2`.

(2) Import `Lab1` notebook created and saved in the previous exercise.

(3) Copy the statements processed in `Lab1` into `Lab2` notebook to Start the services of HDFS, YARN, and Job History Server.

If you plan to use command line in terminal window then type all commands at `$` prompt in terminal window.

(1) How to start Hadoop ?

Open a new Terminal window or Zeppelin paragraph and start all Hadoop processes in the following way.

```
$HADOOP_HOME/sbin/start-all.sh

jps
```

When ready minimize the `Terminal` window used to start Hadoop. We shall refer to this window as to "Hadoop window" and we shall use it later on.

(2) Run MapReduce Applications

In this step, we process two MapReduce applications available in the Hadoop installation. Both applications are included in the `hadoop-mapreduce-examples-2.7.3.jar` file in `$HADOOP_HOME/share/hadoop/mapreduce.`

This first application is the computation of Pi by executing the following command (type 3 lines listed blow in one line):

```
$HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-
2.7.3.jar pi 10 20
```

Another application that uses a regular expression to search for all strings that start from a string `dfs`. First create a folder `input` in HDFS in the following way.

```
$HADOOP_HOME/bin/hadoop fs -mkdir input
```

Next, process the following command to copy the files from etc/Hadoop at a local file system into input folder at HDFS.

```
$HADOOP_HOME/bin/hadoop fs -put /etc/hadoop/conf/* input
```

Next process an application that uses a regular expression to search for all strings that start from a string `dfs`.

```
$HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-
2.7.3.jar grep input output 'dfs[a-z.]+'
```

Note, that to perform the above operation, if the `output` folder exists in your HDFS, you need to remove it first.

Check the outcome of this application through listing the contents of a folder `output`.

```
$HADOOP_HOME/bin/hadoop fs -cat output/part*
```

Finally, remove the folders input and output from HDFS.

```
$HADOOP_HOME/bin/hadoop fs -rm input/*
$HADOOP_HOME/bin/hadoop fs -rmdir input
$HADOOP_HOME/bin/hadoop fs -rm output/*
$HADOOP_HOME/bin/hadoop fs -rmdir output
```

(3) YARN user interface

Enter a link `bigdata-virtualbox:8088` into your web browser. Check the list of applications you just submitted and completed.

(4) How to compile `WordCount.java` Java program

Download a file `WordCount.java` and use editor (`gedit`) to browse its code. The functionality of **WordCount** application and its implementation has been thoroughly discussed during the lecture classes.

To set appropriate environment for compilation of a program process the following command.

```
  export HADOOP_CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath)
```

To compile a program process the following command.

```
javac -classpath ${HADOOP_CLASSPATH}  WordCount.java
```

Finally, we have to create jar file in the following way.

```
jar cf WordCount.jar WordCount*.class
```

(5) How to process `WordCount.java` Java program

Next, upload to HDFS home folder any text file. For example we can use a file `WordCount.java`, please see below.

```
$HADOOP_HOME/bin/hadoop fs -put WordCount.java .
```

WordCount application will count the total number occurrences for each word in the file. Finally, process **WordCount** application in the following way.

```
$HADOOP_HOME/bin/hadoop jar WordCount.jar WordCount WordCount.java /user/output/
```

Note, that an input parameter `WordCount.java` determines an input file located in HDFS in `/user/bigdata` folder. An input parameter `/user/output` determines a location of the output files in HDFS.

To see the results from processing of **WordCount** application process the following commands that lists the contents of a file created in HDFS.

```
$HADOOP_HOME/bin/hadoop fs -ls /user/output
```

```
$HADOOP_HOME/bin/hadoop fs -cat /user/output/part-r-00000
```

The first command lists the contents of output folder created by Hadoop and the second command lists the contents of a file with the results of counting.

(6) How to clean after processing of WordCount application

Before we can process **WordCount** application again we have to remove output folder created by Hadoop. Of course, it is also possible to re-process **WordCount** application with a different value of parameter that determines a location of the results.

To delete a folder output we have make it empty first with the following command.

```
$HADOOP_HOME/bin/hadoop fs -rm /user/output/*
```

Then, we can delete the folder with the following command.

```
$HADOOP_HOME/bin/hadoop fs -rmdir /user/output
```

And verify the results with

```
$HADOOP_HOME/bin/hadoop fs -ls /user
```

To remove `WordCount.java` from HDFS process the following command.

```
$HADOOP_HOME/bin/hadoop fs -rm WordCount.java
```

(7) How to process `MinMax.java` Java program

`MinMax` application reads the key-value pairs and for each distinct key it finds the largest value associated with a key. Its functionality is the same as the following SQL statement.

```
SELECT key, MIN(value), MAX(value)
FROM Sequence-of-key-value-pairs
GROUP BY key;
```

Use an editor to examine the contents of a file `MinMax.java`. The application has a very similar structure to WordCount application. The Mapper is almost identical. The Reduce instead of summations find minimal and maximal values associated with each key.

To set appropriate environment for compilation of a program process the following command.

```
export HADOOP_CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath)
```

To compile a program process the following command.

```
javac -classpath ${HADOOP_CLASSPATH}  MinMax.java
```

To create jar file process the following command.

```
jar cf MinMax.jar MinMax*.class
```

Next, upload to HDFS home folder a file `sales.txt` and list its contents in HDFS.

```
$HADOOP_HOME/bin/hadoop fs -put sales.txt .
$HADOOP_HOME/bin/hadoop fs -cat sales.txt
```

Before processing **MinMax** application, make sure that a folder `/user/output` does not exist in HDFS.

```
$HADOOP_HOME/bin/hadoop fs -ls /user
```

If the folder exists then repeat step (2) above.

Finally, start **MinMax** application in the following way.

```
$HADOOP_HOME/bin/hadoop jar MinMax.jar MinMax sales.txt /user/output/
```

To see the results from processing of **MinMax** application process the following command.

```
$HADOOP_HOME/bin/hadoop fs -cat /user/output/part-r-00000
```

Repeat step (2) to remove the results of processing **MinMax** application.

(8) How to process Filter application

An objective of **Filter** application is to select from the contents of a file `sales.txt` all pairs such that amount of sales is greater than a given value. Its functionality is the same as the following SQL statement.

```
SELECT key, value
FROM Sequence-of-key-value-pairs
```

```
WHERE value > given-value;
```

Use an editor to examine the contents of a file `Filter.java`. The application has a simpler structure than WordCount and MinMax applications. The Mapper is almost identical and there is no Reducer. Please see the following line in a Driver `main()`.

```
job.setNumReduceTasks(0); // Set number of reducers to zero
```

To set appropriate environment for compilation of a program process the following command.

```
export HADOOP_CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath)
```

To compile a program process the following command.

```
javac -classpath ${HADOOP_CLASSPATH}  Filter.java
```

To create jar file process the following command.

```
jar cf Filter.jar Filter*.class
```

A file `sales.txt` is already uploaded to HDFS, such that there is no need to do it again.

Before processing **Filter** application, make sure that a folder `/user/output` does not exist in HDFS.

```
$HADOOP_HOME/bin/hadoop fs -ls /user
```

If the folder does exist then repeat step (2) above.

Next, we create a parameterized shell script to process **Filter** application with a parameter that determines the minimal sales from which the transactions are listed. Start gedit editor and open an empty file run in the following way.

```
gedit run.sh
```

Next, type into a file `run.sh` the following line.

```
$HADOOP_HOME/bin/hadoop jar Filter.jar Filter $1 sales.txt /user/output/
```

Note, that `$1` represents a parameter of shell script `run.sh`. In the future we shall use the shell script to process a line listed above and to replace `$1` with a value of actual parameter. Save a file `run.sh` and quit an editor.

Next, we have to grant processing (execute) privileges to a user on a shell script run.sh. To do so process the following command.

```
chmod u+x run.sh
```

Finally, to run **Filter** application process the following command.

```
./run.sh 45
```

Note a value of actual parameter `45` at the end of a command line. The parameter means that we would like to retrieve only sales whose value is greater than 45. When the shell script is processed a value `45` replaces a formal parameter `$1`.

To see the results from processing of **Filter** application process the following command.

```
$HADOOP_HOME/bin/hadoop fs -cat /user/output/part-r-00000
```

Repeat step (2) to remove the results of processing **Filter** application.

(9) How to process `Grep` and `InvIndx` applications ?

`Grep` application finds the words that match a given template provided as a regular expression. You can use command shell scripts `c.sh`, `j.sh`, `load.sh`, `r.sh`, `show.sh`, `clean.sh` in the following way.

To compile the application, process a shell script `c.sh` in the following way.

```
./c.sh
```

To create jar file, process a shell script `j.sh` in the following way.

```
./j.sh
```

To load a file `grep.txt`, process a shell script `load.sh` in the following way.

```
./load.sh grep.txt
```

To run the application process shell script `r.sh` in the following way.

```
./r.sh grep.txt
```

To display the results process a shell script `show.sh` in the following way.

```
./show.sh
```

To remove the results process a shell script `clean.sh` in the following way.

```
./clean.sh
```

It is quite easy to adopt the shell scripts listed above to process an application `InvInd` that creates an inverted index.

End of Exercise 2