Aims

This exercise aims to get you to:

- Compile, run, and debug MapReduce tasks via Hadoop Streaming
- Compile, run, and debug MapReduce tasks via MRJob
- Apply the design pattern "in-mapper combining" you have learned in Chapter 2.2 to MapReduce programming

One Tip on Hadoop File System Shell

Following are the three commands which appear same but have minute differences:

```
1. hadoop fs {args}

2. hadoop dfs {args}

3. hdfs dfs {args}
```

The first command: fs relates to a generic file system that can point to any file system like local, HDFS etc. So this can be used when you are dealing with different file systems such as Local FS, HFTP FS, S3 FS, and others.

The second command: dfs is very specific to HDFS. It would work for operations relates to HDFS. This has been **deprecated** and we should use hdfs dfs instead.

The third command: It is the same as 2nd. It would work for all the operations related to HDFS and is the recommended command instead of hadoop dfs.

Thus, in our labs, it is always recommended to use **`hdfs dfs {args}`**.

Hadoop Streaming

Hadoop streaming is a utility that comes with the Hadoop distribution. The utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper/reducer.

1. In Lab1, you tested an application of word count. In Linux, wc command can be used to find out **number of lines**, **word count**, **byte and characters count** in the files specified in the file arguments. Test wc command:

```
$ wc $HADOOP_HOME/etc/hadoop/*.xml
```

2. Run a streaming task with /bin/cat as the mapper and /bin/wc as the reducer:

```
$ hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
    -input input \
    -output output2 \
    -mapper /bin/cat \
    -reducer /bin/wc
```

3. Check out the output:

```
$ hdfs dfs -cat output2/*
```

4. To specify the number of reducers, for example two, use:

```
$ hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
  -D mapreduce.job.reduces=2 \
  -input input \
  -output output3 \
  -mapper /bin/cat \
  -reducer /usr/bin/wc
```

Specifying Python Scripts as Mapper/Reducer

Next, you will learn to run Map/Reduce jobs with Python script as the mapper/reducer.

1. Create a folder "Lab2", and put all your codes written in this week's lab in this folder. Next, create a file named mapper.py and copy the codes below into the file.

```
#!/usr/bin/python3

import sys

for line in sys.stdin:
   line = line.strip()

   words = line.split()

   for word in words:
      print (word + "\t" + "1")
```

Make sure this file has the execution permission:

```
$ chmod +x mapper.py
```

2. Similarly, create a file named reducer.py and copy the codes below into the file.

```
#!/usr/bin/python3
import sys

results = {}
for line in sys.stdin:
    word, frequency = line.strip().split('\t', 1)
    results[word] = results.get(word, 0) + int(frequency)
words = list(results.keys())

for word in words:
    print(word, results[word])
```

Also, make sure this file has the execution permission:

```
$ chmod +x reducer.py
```

Compare the above code with that provided in slide 38 of Lecture 2.1. What are the differences? What problem the above approach may encounter?

3. Run the application:

```
$ hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \

        -input input \
        -output python_output \
        -mapper mapper.py \
```

```
        -reducer reducer.py \
        -file mapper.py \
        -file reducer.py
```

4. Check out the output:

```
$ hdfs dfs -cat python_output/*
```

5. Use a combiner:

```
$ hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
        -input input \
        -output python_output \
        -mapper mapper.py \
        -reducer reducer.py \
        -combiner combiner.py \
        -file mapper.py \
        -file reducer.py \
        -file combiner.py
```

Try to Write Your First Hadoop Streaming Job

1. Download the test file "pg100.txt" to your home folder from https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/66890, and put it to HDFS:

```
$ hdfs dfs -rm input/*
$ hdfs dfs -put ~/pg100.txt input
```

2. Now please write your first MapReduce job with Hadoop Streaming to accomplish the following task:

Output the number of words that start with each letter. This means that for every letter we want to count the total number of words that start with that letter. In your implementation, please first convert all words to lower case. You can ignore all non-alphabetic characters. Create "mapper. py" and "reducer.py" scripts in the folder "LetterCount" to finish this task.

Hint: In the (key, value) output, each letter is the key, and its count is the value.

1. How to write a mapper properly?
2. How to write a combiner? Is it necessary?
3. How to write a reducer properly?

Compare your results with the answer provided at:

https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/66889

Install MRJob

As you will install mrjob with pip3, you should first install pip3:

```
$ sudo apt install python3-pip
```

Then, you will be asked to enter the sudo password: `comp9313`

When pip3 is successfully installed, install mrjob with pip3:

```
$ pip3 install mrjob
```

You also need to configure and start YARN since running mrjob on Hadoop requires YARN. Please edit the mapred-site.xml and yarn-site.xml by following Lab 1 instructions. Then, start HDFS and YARN by running “start-dfs.sh” and “start-yarn.sh”.

An Example of MRJob

Create a file called mr_word_count.py in the folder Lab2 and type this into it:

```
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

Download the input file “pg100.txt” from the following link to your home folder (/home/comp9313):

https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/66890

Run the codes locally with:

```
$ python mr_word_count.py ~/pg100.txt
```

You will see the result:

```
Running step 1 of 1...
job output is in /tmp/mr_word_count.comp9313.20210927.075446.090653/output
Streaming final output from /tmp/mr_word_count.comp9313.20210927.075446.090653/output...
"chars" 5340313
"lines" 124787
"words" 904061
Removing temp directory /tmp/mr_word_count.comp9313.20210927.075446.090653...
```

If you want to run in Hadoop, use the “-r hadoop” option, and then you can check the result in the file “output”:

```
$ python mr_word_count.py -r hadoop < pg100.txt > output
```

If your files are in HDFS, you can run like:

```
$ python mr_word_count.py -r hadoop hdfs://localhost:9000/user/comp9313/input
```

There are different ways to run your job, see more details here.

Try to Write Your First MRJob Program

Please write your first mrjob program to complete the above "letter count" task, and compare it with the Hadoop streaming approach.

Improve WordCount by In-Mapper Combining

A better tokenization method:

Use the following codes to tokenize a line of document:

```
import re

words = re.split("[ *$&#/\t\n\f\"\'\\,.:;?!\[\](){}<>~\-_]", line.lower())
```

Documents will be split according to all characters specified (i.e., `" *$&#/\t \n\f"'\,.:;?![](){}<>~-_"`), and higher quality terms will be generated.

Convert all terms to lower case as well (by using lower() method).

Apply this to mapper.py of WordCount we have used. Note that you need to check if the word is an empty string now.

a) Put the input file to HDFS by:

```
$ hdfs dfs -rm input/*
$ hdfs dfs -put ~/pg100.txt input
```

b) Go into the folder Lab2 and use the existing mapper.py and reducer.py scripts.
c) Use the new method to tokenize a line of document
d) Run the application with Hadoop Streaming/mrjob
e) Remember to delete to output folder if it exists in HDFS
f) If you forget the details, please refer to the previous instructions.

Type the following command in the terminal:

```
$ hdfs dfs -cat output/* | head
```

You should see results:

```
comp9313@comp9313-VirtualBox:~/9313_files$ hdfs dfs -cat python_output/* | head
0 2
00 1
000 1
01 1
02 1
03 1
04 1
05 1
08 1
1 362
```

Use the following command:

```
$ hdfs dfs -cat output/* | tail
```

You should see:

```
comp9313@comp9313-VirtualBox:~/9313_files$ hdfs dfs -cat python_output/* | tail
zephyrs 1
zip 1
zipped 1
zir 2
zo 1
zodiac 1
zodiacs 1
zone 1
zounds 24
zwagger 1
```

Apply the in-mapper combing approach:

Based on the previous python scripts, you are required to write an improved version using the "in-mapper combining" approach.

Hadoop Streaming: Create a new script "mapper2.py" in the folder Lab2 and solve this problem. Your results should be the same as generated by mapper.py.

mrjob: Create a new script "mr_word_count2.py" in the folder Lab2 and solve this problem. Your results should be the same as generated by mapper.py.

Hints:

1. Refer to the pseudo-code shown in slide 23 of Chapter 2.2.
2. You can use a dictionary in the mapper script to buffer the partial results for Hadoop streaming.
3. You need to override the methods mapper_init() and mapper_final() in mrjob
4. Do you need to change the reducer?

Solutions of these Problems

I hope that you are able to finish all problems by yourself, since the hints are already given. All the source codes will be published in the course homepage on Friday next week.