MapReduce

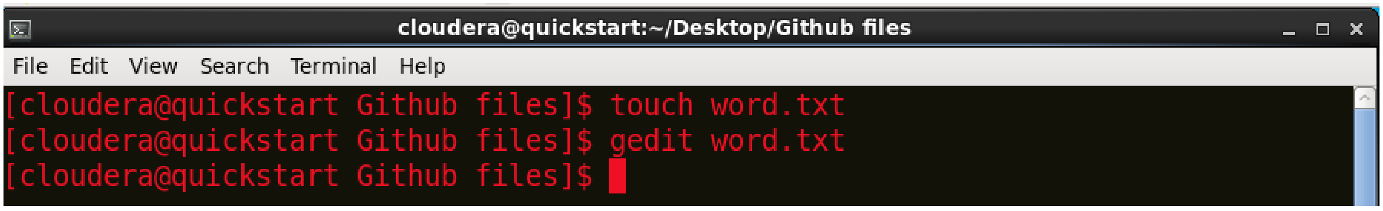
Codes and Instructions available at:
https://drive.google.com/drive/folders/1Ut9ZZqEpMpH8WXJt7rXi-qCAE6lVa6HC?usp=sharing

Mount the shared folder using the command:
`mount -t vboxsf SharedFolder /home/cloudera/Desktop/Windows`

Step 1: Create the Text File

Create a file with the name word.txt and add some data to it.

```
touch word.txt
gedit word.txt
```


cloudera@quickstart:~/Desktop/Github files
File Edit View Search Terminal Help

```
[cloudera@quickstart Github files]$ touch word.txt
[cloudera@quickstart Github files]$ gedit word.txt
[cloudera@quickstart Github files]$ 
```

Step 2: Create mapper.py to implement mapper logic

This script is a Python implementation of a mapper for a Hadoop MapReduce job. The purpose of the mapper is to process input data and emit intermediate key-value pairs that will be used as input for the subsequent reduce phase. In the context of the Word Count example, this script is designed to count the occurrences of each word in the input data.

Here's a detailed explanation of the script:

```python
#!/usr/bin/env python

# import sys because we need to read and write data to STDIN and STDOUT
import sys

# reading entire line from STDIN (standard input)
for line in sys.stdin:
    # to remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()

    # we are looping over the words array and printing the word
    # with the count of 1 to the STDOUT
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        print '%s\t%s' % (word, 1)
```

Let's break down each part of the script:

1. `#!/usr/bin/env python`: This is called a shebang line. It specifies the interpreter to be used to execute the script. In this case, it's using the "env" command to locate and use the system's default Python interpreter.
2. `import sys`: This line imports the `sys` module, which provides access to system-specific parameters and functions, including reading from and writing to standard input and output.
3. `for line in sys.stdin:`: This loop reads lines from standard input (STDIN) one by one.
4. `line = line.strip()`: The `strip()` method is used to remove any leading or trailing whitespace characters from the line.
5. `words = line.split()`: This line splits the cleaned line into a list of words. By default, `split()` splits the line using spaces as the delimiter, resulting in individual words.
6. The inner loop `for word in words:` iterates over each word in the list of words extracted from the line.
7. `print '%s\t%s' % (word, 1)`: For each word encountered, this line prints the word as the key and the value `1` separated by a tab character (`\t`) to STDOUT. The tab character acts as a delimiter between the key and value.

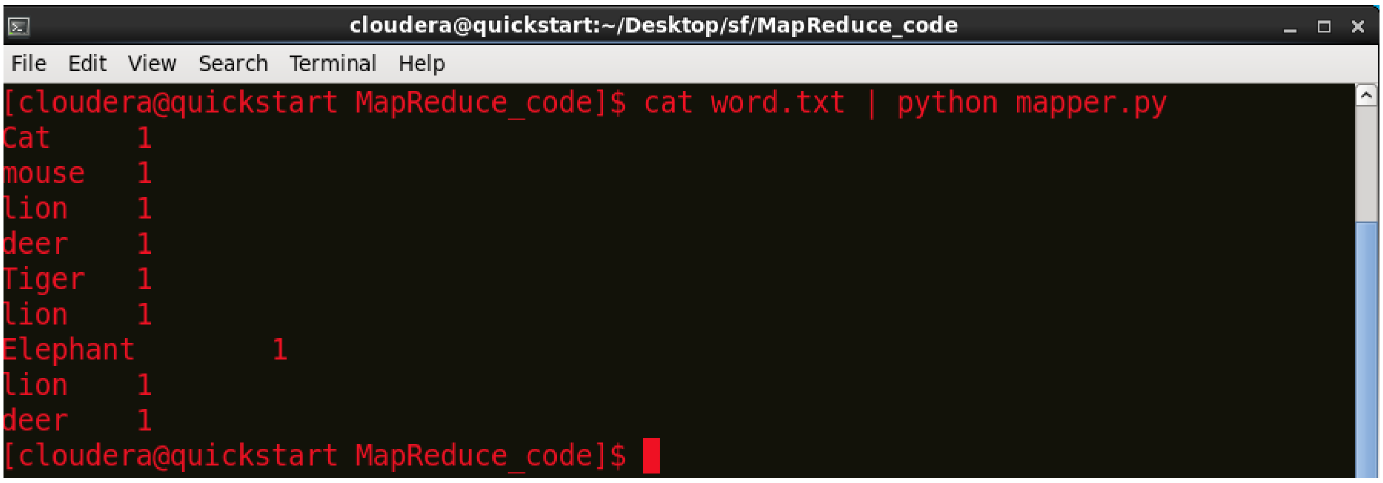
For example, if the input line is `"Hello world and hello again"`, the mapper would emit the following key-value pairs:

```
Hello   1
world   1
and     1
hello   1
again   1
```

These intermediate key-value pairs are then grouped and shuffled by the Hadoop framework before being passed to the reducer for further processing. The reducer will aggregate the values associated with each key (word) and compute the final word counts.

Test mapper.py locally

`cat word.txt | python mapper.py`


cloudera@quickstart:~/Desktop/sf/MapReduce_code
File Edit View Search Terminal Help

```
[cloudera@quickstart MapReduce_code]$ cat word.txt | python mapper.py
Cat     1
mouse   1
lion    1
deer    1
Tiger   1
lion    1
Elephant        1
lion    1
deer    1
[cloudera@quickstart MapReduce_code]$ 
```

Step 3: Create reducer.py to implement reducer logic

Certainly! This script is a Python implementation of a reducer for a Hadoop MapReduce job. The purpose of the reducer is to aggregate and process the intermediate key-value pairs emitted by the mapper and generate the final output, which in the case of Word Count, represents the count of occurrences for each unique word.

Let's go through the script step by step:

```python
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# read the entire line from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # splitting the data on the basis of tab we have provided in mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

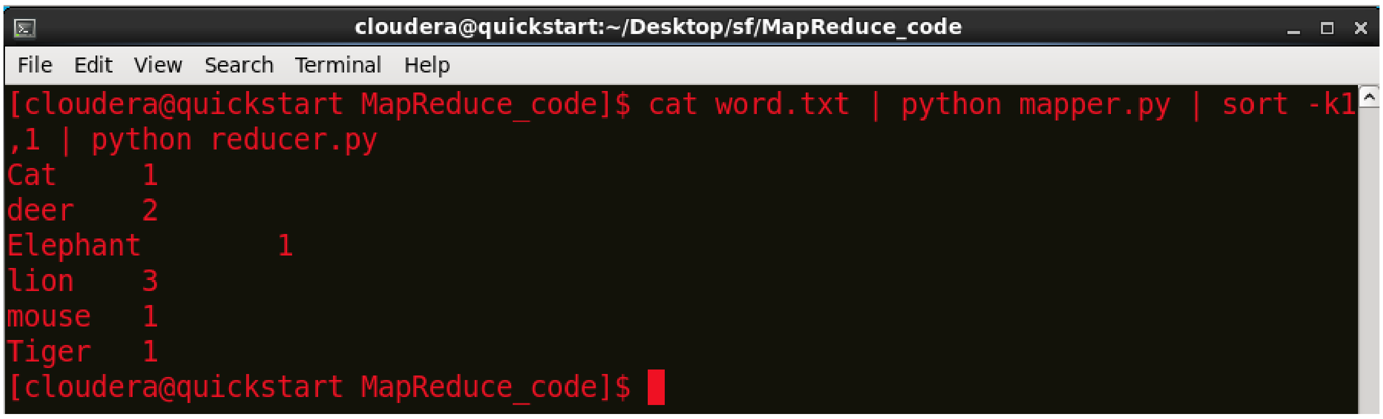
Now let's break down each part of the script:

1. `#!/usr/bin/env python`: Same as in the previous script, this is a shebang line that specifies the Python interpreter to be used.
2. `from operator import itemgetter`: The `itemgetter` module is imported from the `operator` module. This module provides functions for performing item-based operations.
3. `current_word = None`, `current_count = 0`, `word = None`: These variables are used to keep track of the current word being processed, the count associated with that word, and the new word being read from STDIN.
4. The script reads lines from standard input (STDIN) one by one using a loop.
5. `line = line.strip()`: The `strip()` method is used to remove any leading or trailing whitespace characters from the line.
6. `word, count = line.split('\t', 1)`: The line is split into two parts using the tab character `\t` as the delimiter. The `word` is the first part, and the `count` is the second part.
7. `try:` and `except ValueError:`: This code block attempts to convert the `count` from a string to an integer. If the conversion fails (e.g., if `count` is not a valid integer), the script ignores that line and continues to the next iteration of the loop.
8. The script then checks if the current word is the same as the new word being processed. If they are the same, the `current_count` is incremented by the count from the new line.
9. If the current word changes (indicating a new word in the sorted input), the script prints the result for the previous word and resets `current_count` and `current_word`.
10. Finally, the script handles the last word encountered by checking if the `current_word` is the same as the `word` being processed and prints the result if necessary.

The key idea behind the reducer is that it receives intermediate key-value pairs sorted by the key (word) from the mapper. It then aggregates the counts for each word, generating the final output with the word and its total count. The reducer takes advantage of the fact that Hadoop sorts the intermediate data before passing it to the reducer.

Let's test reduer.py locally

```
cat word.txt | python mapper.py | sort -k1,1 | python reducer.py
```


cloudera@quickstart:~/Desktop/sf/MapReduce_code
File Edit View Search Terminal Help

```
[cloudera@quickstart MapReduce_code]$ cat word.txt | python mapper.py | sort -k1
,1 | python reducer.py
Cat     1
deer    2
Elephant        1
lion    3
mouse   1
Tiger   1
[cloudera@quickstart MapReduce_code]$ 
```

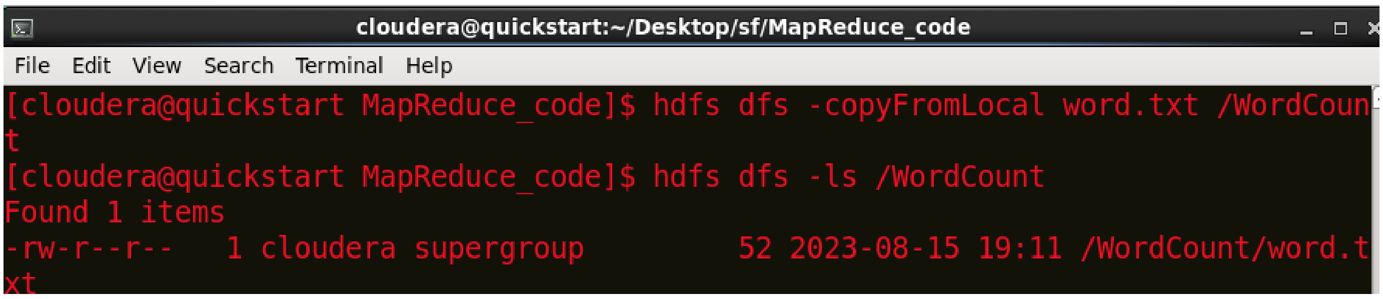
Step 4: Create a WordCount directory in HDFS

```
hdfs dfs -mkdir /WordCount
```

Step 5: Copy word.txt to this folder in our HDFS with help of copyFromLocal command.

Syntax to copy a file from your local file system to the HDFS is given below:

```
hdfs dfs -copyFromLocal word.txt /WordCount
```


cloudera@quickstart:~/Desktop/sf/MapReduce_code
File Edit View Search Terminal Help

```
[cloudera@quickstart MapReduce_code]$ hdfs dfs -copyFromLocal word.txt /WordCoun
t
[cloudera@quickstart MapReduce_code]$ hdfs dfs -ls /WordCount
Found 1 items
-rw-r--r--   1 cloudera supergroup         52 2023-08-15 19:11 /WordCount/word.t
xt
```

Step 6: Make mapper.py and reducer.py executable

```
chmod 777 mapper.py reducer.py
```

Step 7: Download streaming jar file

Now download the latest hadoop-streaming jar file from https://jar-download.com/artifacts/org.apache.hadoop/hadoop-streaming/2.7.3/source-code. Then place, this Hadoop,-streaming jar file to a place from you can easily access it.

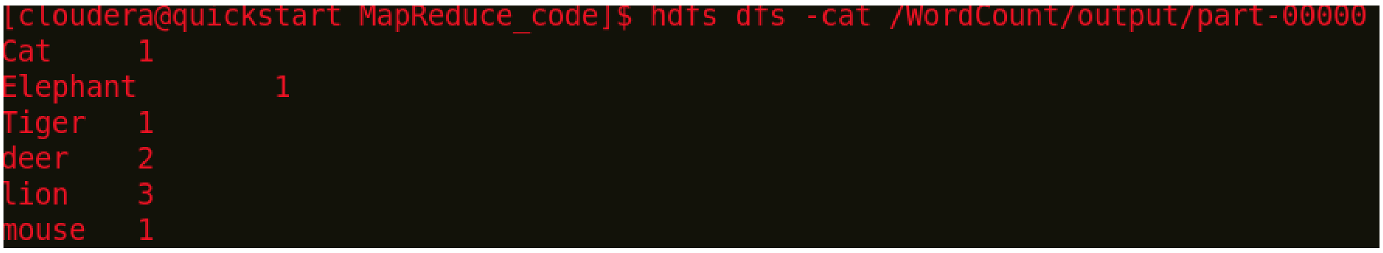
Step 8: Run our python files with the help of the Hadoop streaming utility

```
hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-
streaming-mr1.jar   -input /WordCount/word.txt   -output /WordCount/output
```

```
-mapper /home/cloudera/Desktop/sf/MapReduce_code/mapper.py   -reducer
/home/cloudera/Desktop/sf/MapReduce_code/reducer.py
```

Step 9: Let's see the output

```
hdfs dfs -cat /WordCount/output/part-00000
```



```
[cloudera@quickstart MapReduce_code]$ hdfs dfs -cat /WordCount/output/part-00000
Cat     1
Elephant        1
Tiger   1
deer    2
lion    3
mouse   1
```

References

1. https://www.geeksforgeeks.org/hadoop-streaming-using-python-word-count-problem/
2. https://www.youtube.com/watch?v=tXKzy7IGjkw&list=LL&index=2&t=722s
3. https://iamshobhitagarwal.medium.com/writing-your-first-map-reduce-program-in-hadoop-using-python-e5885331b67b