Experiment no 2

Aim: To implement word count using MapReduce.

Theory:

Pyspark

PySpark is an interface for Apache Spark in Python. It not only allows you to write Spark applications using Python APIs, but also provides the PySpark shell for interactively analyzing your data in a distributed environment. PySpark supports most of Spark's features such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) and Spark Core.

Working of Wordcount Mapreduce

The main method invokes the ToolRunner to run the job based on the configuration information. The map method processes one line at a time, splitting the line on regular expression word boundaries. It emits key/value pairs in the format <word, 1>.

For File0, the map method emits these key/value pairs:
<Hadoop, 1>
<is, 1>
<an, 1>
<elephant, 1>

For File1, the map method emits:
<Hadoop, 1>
<is, 1>
<as, 1>
<yellow, 1>
<as, 1>
<can, 1>
<be, 1>

For File2, the map method emits:
<Oh, 1>
<what, 1>
<a, 1>

<yellow, 1>
<fellow, 1>
<is, 1>
<Hadoop, 1>

The reduce method adds up the number of instances for each key, and then emits them sorted in UTF-8 alphabetical order (all uppercase words, and then all lowercase words). Note that the WordCount code specifies key/value pairs. The Mapper and Reducer classes handle the rest of the processing for us.

<Hadoop, 3>
<Oh, 1>
<a, 1>
<an, 1>
<as, 2>
<be, 1>
<can, 1>
<elephant, 1>
<fellow, 1>
<is, 3>
<what, 1>
<yellow, 2>

Code and Output:

```
[1] !apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

```
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz
```

```
[3] !tar xf spark-3.0.0-bin-hadoop3.2.tgz
```

```
[4] import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"
```

```
[6] !pip install -q findspark
```

```
[7] import findspark
    findspark.init()
    from pyspark.sql import SparkSession
    spark = SparkSession.builder.master("local[*]").getOrCreate()
    from pyspark import SparkContext
    sc = SparkContext.getOrCreate()
    from google.colab import drive
```

```
[9] drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[11] import os
     import urllib.request
     shkspr=urllib.request.urlretrieve('https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt','/content/drive/My Drive/shakespeare.txt')
```

```
[12] Words=sc.textFile("/content/drive/My Drive/shakespeare.txt")
     WordsCount=Words.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1))
     WordsCount.count()
```

```
1418390
```

```
[13] DistinctWordsCount=WordsCount.reduceByKey(lambda a,b: a+b)
     DistinctWordsCount.count()
```

```
67506
```

```
[14] SortedWordsCount=DistinctWordsCount.map(lambda a: (a[1], a[0])).sortByKey()
```

```
[15] SortedWordsCount.top(20)
```

```
[(517065, ''),
 (23242, 'the'),
 (19540, 'I'),
 (18297, 'and'),
 (15623, 'to'),
 (15544, 'of'),
 (12532, 'a'),
 (10824, 'my'),
 (9576, 'in'),
 (9081, 'you'),
 (7851, 'is'),
 (7531, 'that'),
 (7068, 'And'),
 (6948, 'not'),
 (6722, 'with'),
 (6218, 'his'),
 (6009, 'your'),
 (6002, 'be'),
 (5616, 'for'),
 (5236, 'have')]
```

```
[16] Sorted_final=SortedWordsCount.map(lambda a: (a[1], a[0]))
     Sorted_final.top(20)
```

```
[('}', 2),
 ("zwagger'd", 1),
 ('zounds,', 1),
 ('zounds!', 1),
 ('zone,', 1),
 ('zodiacs', 1),
 ('zodiac', 1),
 ('zo', 1),
 ('zir.', 1),
 ('zir,', 1),
 ('zip', 1),
 ('zephyrs', 1),
 ('zenith', 1),
 ('zed!', 1),
 ('zeals,', 1),
 ('zealous', 6),
 ('zeal.', 4),
 ('zeal,', 7),
 ('zeal!', 1),
 ('zeal', 20)]
```

Conclusion:

Thus in this experiment, we installed pyspark and learned how wordcount map and reduce actually works, and implemented a program for the same.