實驗紀錄

一、spark安裝與配置：

1. 進入Apache Spark官網進行Spark的下載，下載完之後解壓，在解壓文件夾所在目錄下打開terminal，將解壓後的文件夾移動到/usr/local/目錄下，並改名爲spark。

2. 安裝pyspark

在 Terminal 輸入：pip install pyspark

3. 配置環境變量

vim ~/.bash\_profile進行編輯，增加環境變量：

export SPARK\_HOME=/Users/wangjingzi/opt/spark

export PATH=$PATH:$SPARK\_HOME/bin

export PYSPARK\_PYTHON=python3

然後保存退出，執行source  ~/.bash\_profile，使之生效。

4. 打開terminal，輸入pyspark，會出現

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 3.2.1

/\_/

Using Python version 3.9.2 (v3.9.2:1a79785e3e, Feb 19 2021 09:06:10)

Spark context Web UI available at http://192.168.0.102:4040

Spark context available as 'sc' (master = local[\*], app id = local-1649003812362).

SparkSession available as 'spark'.

>>>

檢查是否有正確啟動

>>> print(sc.version)

3.2.1

二、體驗spark的執行：對於文件「shakespear.txt」來做字數統計

1. PySpark將會自動使用本地Spark配置創建一個SparkContext, textFile方法將shakespeare.txt加載到一個RDD命名文本

>>> text = sc.textFile("shakespeare.txt")

>>> print(text)

shakespeare.txt MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0

2.首先用tokenize把文本拆分為words, 返回拆分的word list。然後通過給flatMap傳遞tokenize對textRDD進行變換創建了一個wordsRDD。

>>> from operator import add

>>> def tokenize(text):

... return text.split()

...

>>> words = text.flatMap(tokenize)

>>> print(words)

PythonRDD[2] at RDD at PythonRDD.scala:53

3.將每個word map到一個key-value pair，其中key就是word，value是1，然後使用reducer計算每個key的1總數， 可以用toDebugString方法來查看PipelinedRDD是如何被轉換的

>>> wc = words.map(lambda x: (x,1))

>>> print (wc.toDebugString())

b'(2) PythonRDD[3] at RDD at PythonRDD.scala:53

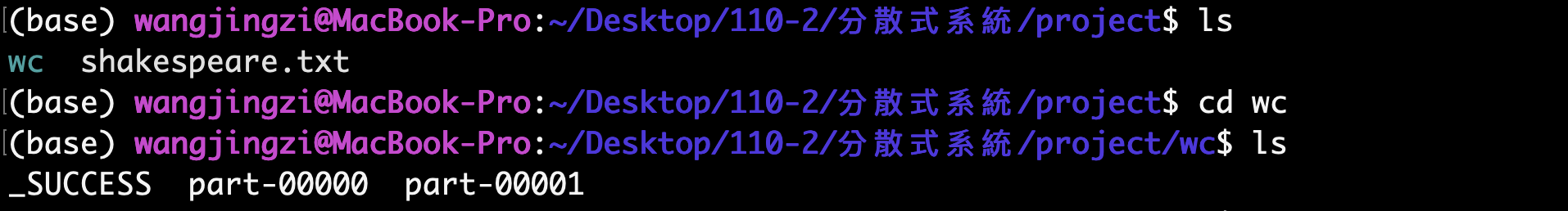
| shakespeare.txt MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0

| shakespeare.txt HadoopRDD[0] at textFile at NativeMethodAccessorImpl.java:0

4.使用reduceByKey進行字數統計，然後把統計結果寫到disk，最終調用saveAsTextFile，這個分布式任務就開始執行

>>> counts = wc.reduceByKey(add)

>>> counts.saveAsTextFile("wc")

5. exit()退出後，會發現該目錄下多了一個wc directory

每個part文件都代表進程計算得到的被保持到disk上的最終RDD。對某一個part文件進行head命令，可以能看到字數統計元組。

三、以standalone單機模式實做簡單範例：

使用本机作为master，搜索該程式同目录下的ontime文件夾的2个CSV file，將航班的延誤時間可視化出來，以下僅展示部分code

import **csv**

import **matplotlib**.**pyplot** as **plt**

import **io**

from **datetime** import **datetime**

from **collections** import **namedtuple**

from **operator** import **add**, **itemgetter**

from **pyspark** import **SparkConf**, **SparkContext**

首先通過setMaster將Spark配置到SparkConf，再創建了SparkContext，使用了配置好的context執行main

if \_\_name\_\_ == "\_\_main\_\_":

*# Configure Spark*

conf = **SparkConf**().**setMaster**("local[\*]")

conf = conf.**setAppName**(APP\_NAME)

sc = **SparkContext**(conf=conf)

**main**(sc)

Load航空公司的csv到RDD，將其split並返回代表每行的tuple，再將collect傳給RDD，使每一行以list的形式從RDD傳回driver，並儲存為dictionary，然後使用sc.broadcast廣播給cluster的每個node。接著相同的對航班的csv做split並且將數據parse為方便處理的格式。有了Flight的RDD，再通過map將RDD轉換為key-value pair，其中key是航空公司的名字，value是到達和出發的延誤時間總和。使用reduceByKey和add可以得到每個航空公司的延誤時間總和，然後RDD被傳遞給driver，按照升序排列，print出來再使用matplotlib進行了可視化。

def **main**(sc):

airlines =**dict**(sc.textFile(“ontime/airlines.csv”).map(**split**).collect()

airline\_lookup = sc.broadcast(airlines)

flights = sc.textFile("ontime/flights.csv").map(**split**).map(**parse**)

delays = flights.map(lambda f: (airline\_lookup.value[f.AIRLINE],

**add**(f.DEPARTURE\_DELAY,f.ARRIVAL\_DELAY)))

delays = delays.reduceByKey(**add**).collect()

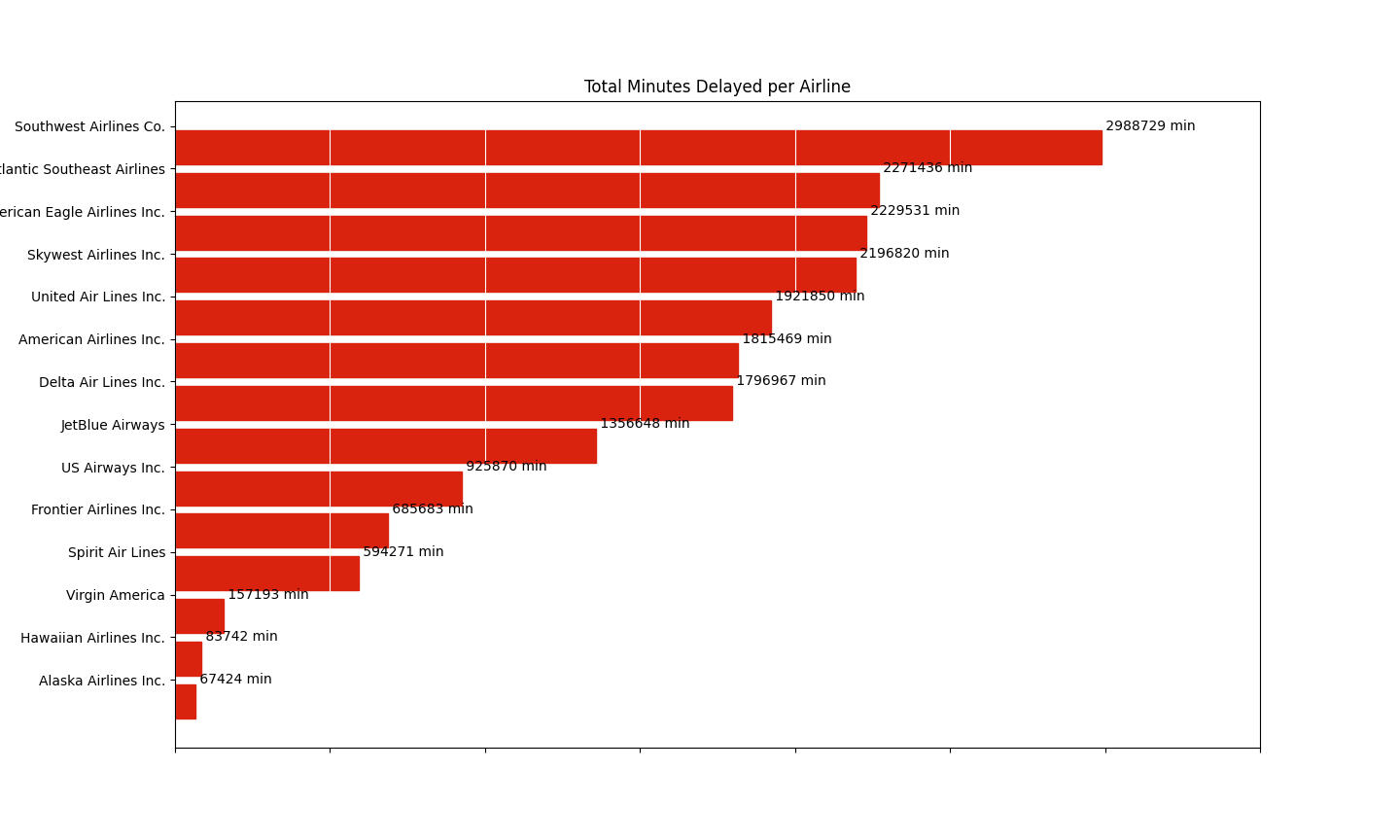
delays = **sorted**(delays, key=**itemgetter**(1))

for d in delays:

**print**("%0.0f minutes delayed\t%s" % (d[1], d[0]))

**plot**(delays)

運行結果如下：



四、使用心得

通過這次實作，我對spark有了基本的瞭解。Spark擁有多種語言的函數式編程API，提供了除map和reduce之外更多的運算符，這些操作是通過一個稱作彈性分布式數據集(resilient distributed datasets, RDDs)的分布式數據框架進行的，RDD是Spark中最基本的數據抽象，它代表一個不可變、可分區、裏面的元素可並行計算的集合。其彈性具體可以體現在自動的進行內存和磁盤數據存儲的切換 ；基於Lineage的容錯，第n個節點出錯，會從第n-1個節點恢復 ；數據partition的高度彈性。

除此之外，本次實作我還體會到了spark的lazy evaluation的特性，這是指spark直到action 動作之前，數據不會先被計算，這裡的action指的是collect，count，reduce等需要拉回產生結果的算子。spark的transform處理的數據，都不會立刻執行，它會根據rdd之間的鏈式進行傳遞，這裡的指的就是transformmap，union，flatmap，groupByKey, join等。例如在第二部分實作中的counts = wc.reduceByKey(add)，此時spark UI上不會有任何job被提交，因爲這個計算沒有任何action算子，實際上根本沒有被計算，這就是lazy特性。只有當counts.saveAsTextFile(「wc」)，這個job才會開始執行，將數據輸出存儲到wc的目錄下。而這樣的特性可以將代碼的程序變成一塊塊的操作鏈，能夠極大的減少中間計算過程提高計算效率，只有真正需要用到的數據才會計算。