به‌نام خداوند بخشنده و مهربان

تمرین سوم: زمان اجرا
درس: تحلیل سیستم داده‌های حجیم

استاد: محمدعلی نعمت‌بخش
دستیاران: فاطمه ابراهیمی، پریسا لطیفی، امیر سرتیپی

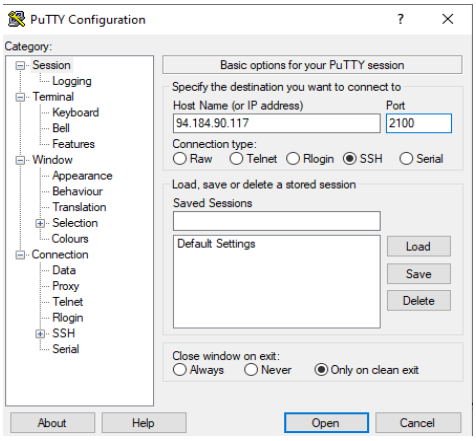
نام و نام‌خانوادگی: نگین شمس
آدرس گیت: https://github.com/NeginShams/spark-hadoop.git

- **لطفا پاسخ تمارین حتما در سامانه‌ی کوئرا ارسال شود.**
- **لطفا پاسخ‌های خود را در خود سند سوال نوشته و در قالب یک فایل PDF ارسال کنید.**
- **نام سند ارسالی HW-{homework number}-{Name Family}-{student number}**
- **تمامی فایل‌های مورد نیاز این تمرین در این لینک قابل دسترس است.**
- **خروجی از هر مرحله‌ی تمرین را در سند خود بارگذاری کنید.**

در این تمرین هدف ما مقایسه زمان اجرا در هدوپ و اسپارک است.
برای این منظور ۴ فایل داده متنی با حجم‌های ۱، ۵، ۱۰ و ۱۲ گیگابایتی در اختیار شما قرار گرفته است که انتظار می‌رود با نوشتن برنامه‌ی شمارش کلمات عملیات نگاشت-کاهش را برای داده‌ها بر روی هدوپ و اسپارک انجام دهید. نتایج را گزارش و مقایسه‌ای بین آنها انجام دهید.
آدرس فایل‌ها:/user/ebrahimi/hw3-data
نمونه‌ی دستور اسپارک را با client mode هم امتحان کرده و تفاوت حالت cluster و client را بیان کنید.

- **اتصال به cluster**

مطابق شکل ۱ برای اتصال به cluster از نرم‌افزار putty استفاده شده است.



شکل ۱: اتصال به Cluster با استفاده از putty

- **هدوپ**

پس از ورود، در مسیر جاری دو فایل جدید با نام‌های mapper.py و reducer.py ایجاد شده است که به‌ترتیب وظیفه انجام عملیات نگاشت و کاهش را بر عهده دارند. در کد مربوط به mapper ورودی از طریق STDIN خوانده می‌شود سپس فاصله‌های اضافی از ورودی حذف شده و ورودی به‌وسیله تابع split به کلمات جداگانه تقسیم می‌شوند و داخل یک لیست قرار می‌گیرند. در پایان تاپل نهایی که شامل یک کلید (رشته‌ی مربوط به کلمه) و یک مقدار (عدد ۱) است، توسط STDOUT به‌عنوان خروجی تولید می‌شود. شکل ۲ نشان‌دهنده ساختار فایل mapper.py است. برای اجرای موفق عملیات نگاشت-کاهش با استفاده از زبان پایتون، لازم است آدرس مفسر پایتون در سیستم به‌صورت قابل مشاهده در شکل ۲ به ابتدای فایل اضافه شود. با توجه به این‌که در صورت سوال به حذف کلمات توقف اشاره نشده است، بنابراین کلمات توقف نیز شمرده می‌شوند. در صورت تمایل می‌توان با استفاده از کتابخانه‌های موجود و یا استفاده از یک فایل خارجی که حاوی کلمات توقف انگلیسی است، می‌توان در ابتدا این کلمات را از متن حذف نمود.

negin_shams@MasterPC: ~

```
  GNU nano 4.8
#!/usr/bin/python
import sys
import re

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        new_word = re.sub(r'[\W_]+', '',word)
        print('%s\t%s' % (new_word, 1))
```

شکل ۲: محتوای فایل mapper.py

برای انجام عملیات کاهش از فایل reducer.py استفاده شده است. محتویات این فایل در شکل ۳ نشان داده شده است. در این فایل، خروجی تولید شده توسط mapper با استفاده از STDIN به‌صورت خط به خط خوانده می‌شود. سپس با استفاده از تابع strip کلمات جدا شده و در متغیر line قرار می‌گیرد. سپس برای هر کلمه تعداد تکرارهای آن در متن محاسبه می‌شود. برای این کار از یک حلقه استفاده شده است که در آن برای هر کلمه تعداد رخدادها جمع می‌شوند. در انتها پاسخ‌ها توسط STDOUT در خروجی نوشته می‌شوند.

negin_shams@MasterPC: ~

```
 GNU nano 4.8
#!/usr/bin/python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
        line = line.strip()
        word, count = line.split('\t', 1)
        # convert string to int for count
        try:
                count = int(count)
        except ValueError:
                continue

        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

if current_word == word:
        print('%s\t%s' % (current_word, current_count))
```

شکل ۳: محتوای فایل reducer.py

سپس مطابق شکل ۴ با استفاده از دستور h -put فایل‌های mapper و reducer در فضای HDFS بارگذاری می‌شود.

```
negin_shams@MasterPC:~$ h -put mapper.py
negin_shams@MasterPC:~$ h -put reducer.py
negin_shams@MasterPC:~$
```

شکل ۴: انتقال فایل‌ها به فضای HDFS

هم‌چنین فایل‌های متنی نیز با استفاده از این دستور در فضای HDFS قرار می‌گیرد. سپس با دستور h -chmod 777 به فایل‌هایmapper و reducer دسترسی لازم داده می‌شود.

با توجه به این‌که برنامه‌های map و reduce به زبان پایتون نوشته شده است، لازم است برای اجرای برنامه از یک فایل jar به‌نام hadoop-streaming-2.7.3 استفاده شود. با استفاده از این فایل می‌توان عملیات نگاشت-کاهش را بر روی داده‌ها اجرا نمود. فایل مذکور در مسیر /home/hduser/hadoop/hadoop/share/hadoop/tools/lib قرار دارد.

- ✓ فایل یک گیگابایتی

برای شمارش کلمات موجود در فایل از دستور نشان داده‌شده در شکل ۵ استفاده شده است:

```
negin_shams@MasterPC:~$ hadoop jar /home/hduser/hadoop/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.10.1.jar \
> -input file1G.txt \
> -output output \
> -mapper mapper.py \
> -reducer reducer.py \
> -file mapper.py \
> -file reducer.py
```

شکل ۵: دستور اجرای عملیات نگاشت-کاهش

پس از وارد کردن دستور فوق، عملیات نگاشت-کاهش بر روی داده‌ها آغاز می‌شود. در پایان در صورت اجرای موفق عبارت completed successfully در ترمینال چاپ می‌شود. شکل ۶ نشان دهنده‌ی بخشی از محتوای صفحه‌ی ترمینال در زمان اجرای این دستور می‌باشد.

```
22/04/09 18:40:44 INFO mapreduce.Job:  map 97% reduce 21%
22/04/09 18:40:45 INFO mapreduce.Job:  map 100% reduce 21%
22/04/09 18:40:47 INFO mapreduce.Job:  map 100% reduce 29%
22/04/09 18:41:05 INFO mapreduce.Job:  map 100% reduce 34%
22/04/09 18:41:11 INFO mapreduce.Job:  map 100% reduce 39%
22/04/09 18:41:17 INFO mapreduce.Job:  map 100% reduce 44%
22/04/09 18:41:23 INFO mapreduce.Job:  map 100% reduce 49%
22/04/09 18:41:29 INFO mapreduce.Job:  map 100% reduce 54%
22/04/09 18:41:35 INFO mapreduce.Job:  map 100% reduce 59%
22/04/09 18:41:41 INFO mapreduce.Job:  map 100% reduce 63%
22/04/09 18:41:47 INFO mapreduce.Job:  map 100% reduce 67%
22/04/09 18:41:53 INFO mapreduce.Job:  map 100% reduce 69%
22/04/09 18:41:59 INFO mapreduce.Job:  map 100% reduce 71%
22/04/09 18:42:05 INFO mapreduce.Job:  map 100% reduce 73%
22/04/09 18:42:11 INFO mapreduce.Job:  map 100% reduce 74%
22/04/09 18:42:17 INFO mapreduce.Job:  map 100% reduce 76%
22/04/09 18:42:23 INFO mapreduce.Job:  map 100% reduce 79%
22/04/09 18:42:29 INFO mapreduce.Job:  map 100% reduce 81%
22/04/09 18:42:35 INFO mapreduce.Job:  map 100% reduce 83%
22/04/09 18:42:41 INFO mapreduce.Job:  map 100% reduce 85%
22/04/09 18:42:47 INFO mapreduce.Job:  map 100% reduce 87%
22/04/09 18:42:53 INFO mapreduce.Job:  map 100% reduce 89%
22/04/09 18:42:59 INFO mapreduce.Job:  map 100% reduce 91%
22/04/09 18:43:05 INFO mapreduce.Job:  map 100% reduce 93%
22/04/09 18:43:11 INFO mapreduce.Job:  map 100% reduce 94%
22/04/09 18:43:17 INFO mapreduce.Job:  map 100% reduce 97%
22/04/09 18:43:23 INFO mapreduce.Job:  map 100% reduce 99%
22/04/09 18:43:27 INFO mapreduce.Job:  map 100% reduce 100%
22/04/09 18:43:27 INFO mapreduce.Job: Job job_1649179500517_0339 completed successfully
```

شکل ۶: محتوای صفحه در زمان اجرای دستور stream

هم‌چنین اطلاعاتی مانند مدت زمان اجرا نیز در خروجی نمایش داده می‌شود. شکل ۷ نشان دهنده‌ی بخشی از این اطلاعات می‌باشد.

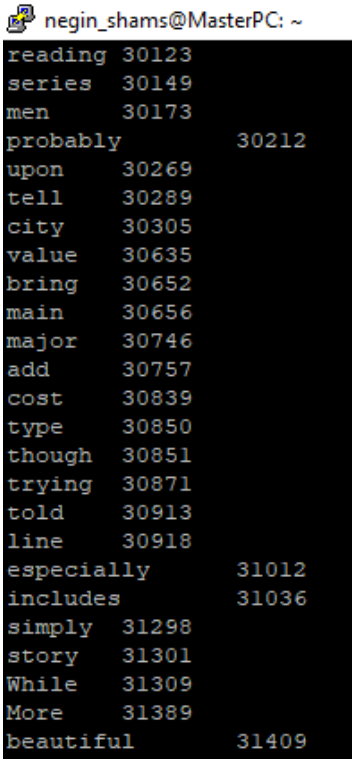
```
        Reduce shuffle bytes=1746546505
        Reduce input records=174516378
        Reduce output records=4007537
        Spilled Records=523549134
        Shuffled Maps =8
        Failed Shuffles=0
        Merged Map outputs=8
        GC time elapsed (ms)=1289
        CPU time spent (ms)=617020
        Physical memory (bytes) snapshot=8161067008
        Virtual memory (bytes) snapshot=47503798272
        Total committed heap usage (bytes)=7543455744
```

شکل ۷: بخشی از اطلاعات مربوط به اجرا

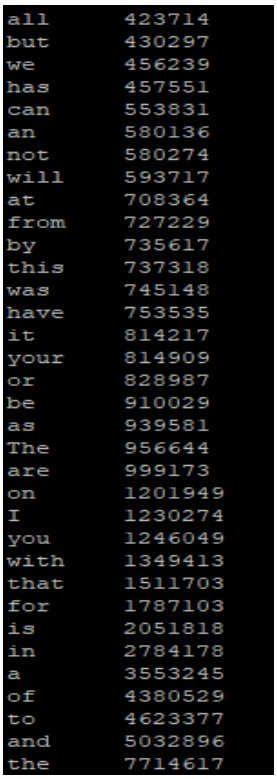
پس از اجرای این دستور یک پوشه‌ی جدید به‌نام output در فضای HDFS ایجاد می‌شود. در این پوشه فایلی به‌نام part-00000 ایجاد می‌شود که حاوی نتایج شمارش کلمات است. هم‌چنین یک فایل SUCCESS_ نیز در این پوشه ایجاد می‌شود. با استفاده از دستور زیر می‌توان نتایج را به‌صورت مرتب‌شده مشاهده نمود.

h -cat /word_count_hw3/output/part-00000 | sort -k 2n

شکل ۸ و شکل ۹ نشان‌دهنده‌ی بخشی از نتایج است.



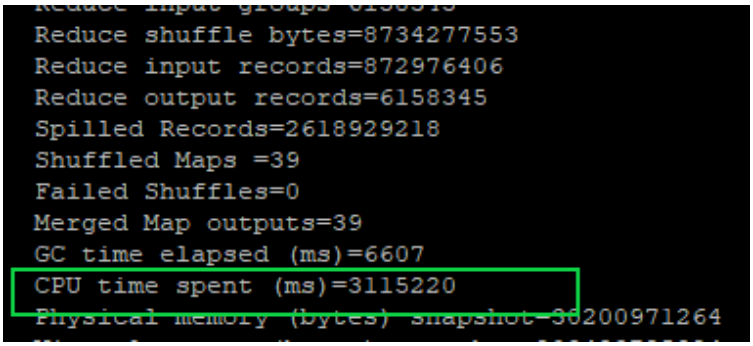
شکل ۸: بخشی از نتایج اجرای برنامه



شکل ۹: بخشی از نتایج اجرای برنامه (ادامه)

- ✓ فایل ۵ گیگابایتی

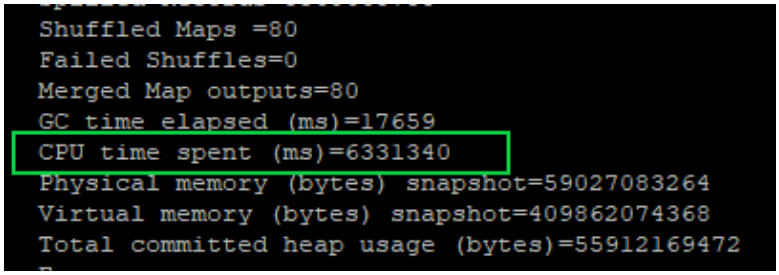
برای این فایل نیز از دستور شکل ۵ استفاده شده است و این‌بار «file5G.txt » به‌عنوان نام فایل قرار می‌گیرد. خروجی نیز در پوشه‌ای به‌نام output2 ذخیره شده است. زمان اجرای این پردازش در شکل ۱۰ قابل مشاهده است.



شکل ۱۰: زمان اجرای پردازش فایل ۵ گیگابایتی

✓ فایل ۱۰ گیگابایتی

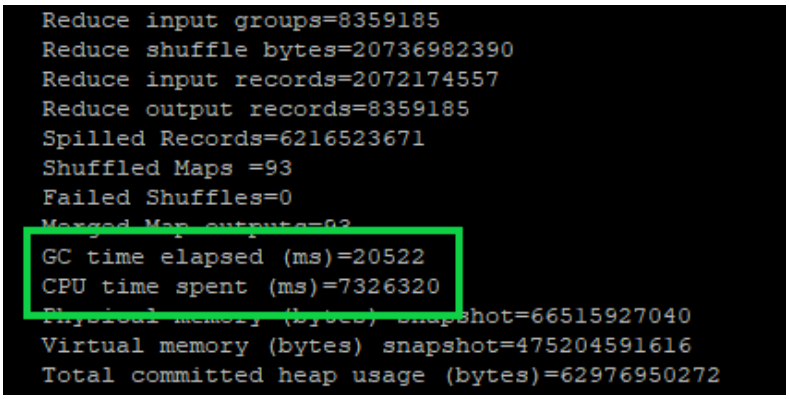
خروجی حاصل از پردازش این فایل در پوشه‌ی output3 در فضای HDFS قرار گرفته است و مدت زمان صرف‌شده برای اجرای آن در شکل ۱۱ قابل مشاهده است.



شکل ۱۱: زمان اجرای فایل ۱۰ گیگابایتی

✓ فایل ۱۲ گیگابایتی

دستور وارد شده برای اجرای برنامه بر روی این فایل نیز مشابه موارد قبلی است. خروجی مربوط به اجرا با این فایل در پوشه‌ای به‌نام output4 قرار دارد و محتوای آن با استفاده از دستور h -cat قابل مشاهده است. شکل ۱۲ نشان‌دهنده زمان اجرای صرف‌شده برای این فایل است.



شکل ۱۲: زمان اجرای فایل ۱۲ گیگابایتی

- **اسپارک (client)**

در این بخش ابتدا یک فایل پایتون به‌نام pyspark_example ایجاد شده است که حاوی برنامه‌ی شمارش کلمات است. محتوای این برنامه در شکل۱۳ قابل مشاهده است. در تابع main این برنامه یک session ایجاد می‌شود. سپس با استفاده از تابع flatMap هر کلمه به‌صورت یک جفت (word, ۱) تبدیل می‌شود و در مرحله بعد با استفاده از reduceByKey می‌توان یک تابع لامبدا تعریف نمود که تعداد رخداد هر کلید را محاسبه می‌کند. سپس با استفاده از تابع saveAsTextfile می‌توان نتایج را در پوشه مورد نظر ذخیره نمود. با توجه به این‌که در اسپارک بر خلاف هدوپ، زمان اجرای برنامه پس از اتمام در صفحه چاپ نمی‌شود، بنابراین در خود برنامه با استفاده از کتاب‌خانه «timeit» می‌توان یک‌بار زمان را در آغاز برنامه و یک‌بار در پایان آن محاسبه نمود و سپس تفاضل این دو زمان را به‌عنوان زمان اجرا در خروجی چاپ کرد.

```
import sys
import timeit

from pyspark import SparkContext, SparkConf

if __name__ == "__main__":
        start = timeit.timeit()

        sc = SparkContext("local","PySpark Word Count")

        words = sc.textFile("file1G.txt").flatMap(lambda line: line.split(" "))

        wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)

        # save the counts to output
        wordCounts.saveAsTextFile("spark_output/")
        end = timeit.timeit()
        print(end - start)
```

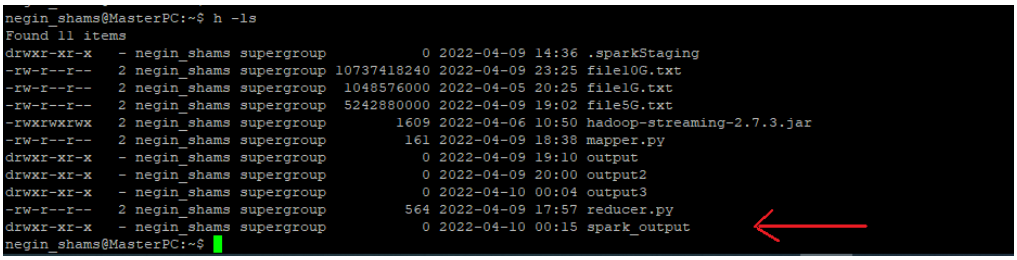
شکل ۱۳: محتوای فایل شمارش کلمات

سپس با استفاده از دستور نشان داده شده در شکل ۱۴ می‌توان عملیات نگاشت-کاهش را آغاز نمود.

```
shams@MasterPC:~$ spark-submit   --master yarn   --deploy-mode client  pyspark_example.py
```

شکل ۱۴: دستور اجرا در حالت کلاینت

پس از اجرای این دستور یک پوشه‌ی جدید مطابق *شکل ۱۵* در فضای HDFS ایجاد می‌شود که حاوی نتایج شمارش کلمات متن است.

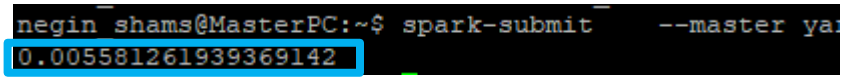


شکل ۱۵: پوشه ایجاد شده پس از اجرای دستور spark-submit

هم‌چنین محتوای این پوشه و مدت زمان اجرای برنامه به‌ترتیب در *شکل ۱۶* و *شکل ۱۷* قابل مشاهده است.

```
negin_shams@MasterPC:~$ h -ls spark_output
Found 9 items
-rw-r--r--   2 negin_shams supergroup          0 2022-04-10 17:30 spark_output/_SUCCESS
-rw-r--r--   2 negin_shams supergroup    9212797 2022-04-10 17:30 spark_output/part-00000
-rw-r--r--   2 negin_shams supergroup    9310556 2022-04-10 17:30 spark_output/part-00001
-rw-r--r--   2 negin_shams supergroup   10275206 2022-04-10 17:30 spark_output/part-00002
-rw-r--r--   2 negin_shams supergroup    9185452 2022-04-10 17:30 spark_output/part-00003
-rw-r--r--   2 negin_shams supergroup    9256671 2022-04-10 17:30 spark_output/part-00004
-rw-r--r--   2 negin_shams supergroup    9205290 2022-04-10 17:30 spark_output/part-00005
-rw-r--r--   2 negin_shams supergroup    9700956 2022-04-10 17:30 spark_output/part-00006
-rw-r--r--   2 negin_shams supergroup    9204285 2022-04-10 17:30 spark_output/part-00007
negin_shams@MasterPC:~$
```

شکل ۱۶: محتوای پوشه



شکل ۱۷: زمان اجرای فایل یک گیگابایتی

- ✓ فایل ۵ گیگابایتی

مانند مرحله قبل و با تغییر مسیر فایل می‌توان این‌بار عملیات را برای فایل ۵ گیگابایتی نیز تکرار نمود. نتایج این اجرا در پوشه‌ای به‌نام spark_output_2 ذخیره شده و قابل مشاهده می‌باشد. *شکل ۱۸* نشان‌دهنده محتویات این پوشه می‌باشد. هم‌چنین *شکل ۱۹* نیز نشان دهنده مدت زمان اجرای برنامه است (نمایش زمان به‌صورت عدد علمی است و باید در 10^5 ضرب شود).

```
negin_shams@MasterPC:~$ h -ls spark_output2
Found 40 items
-rw-r--r--   2 negin_shams supergroup          0 2022-04-10 00:35 spark_output2/_SUCCESS
-rw-r--r--   2 negin_shams supergroup    2967632 2022-04-10 00:34 spark_output2/part-00000
-rw-r--r--   2 negin_shams supergroup    2947299 2022-04-10 00:34 spark_output2/part-00001
-rw-r--r--   2 negin_shams supergroup    2971870 2022-04-10 00:34 spark_output2/part-00002
-rw-r--r--   2 negin_shams supergroup    2963668 2022-04-10 00:34 spark_output2/part-00003
-rw-r--r--   2 negin_shams supergroup    2957329 2022-04-10 00:34 spark_output2/part-00004
-rw-r--r--   2 negin_shams supergroup    2975016 2022-04-10 00:34 spark_output2/part-00005
-rw-r--r--   2 negin_shams supergroup    3425537 2022-04-10 00:34 spark_output2/part-00006
-rw-r--r--   2 negin_shams supergroup    2960246 2022-04-10 00:34 spark_output2/part-00007
-rw-r--r--   2 negin_shams supergroup    2968850 2022-04-10 00:34 spark_output2/part-00008
-rw-r--r--   2 negin_shams supergroup    2973600 2022-04-10 00:34 spark_output2/part-00009
-rw-r--r--   2 negin_shams supergroup    2956395 2022-04-10 00:34 spark_output2/part-00010
-rw-r--r--   2 negin_shams supergroup    2960027 2022-04-10 00:34 spark_output2/part-00011
-rw-r--r--   2 negin_shams supergroup    2957867 2022-04-10 00:34 spark_output2/part-00012
-rw-r--r--   2 negin_shams supergroup    2970339 2022-04-10 00:34 spark_output2/part-00013
-rw-r--r--   2 negin_shams supergroup    4015112 2022-04-10 00:34 spark_output2/part-00014
-rw-r--r--   2 negin_shams supergroup    2963255 2022-04-10 00:34 spark_output2/part-00015
-rw-r--r--   2 negin_shams supergroup    3065435 2022-04-10 00:34 spark_output2/part-00016
-rw-r--r--   2 negin_shams supergroup    2983420 2022-04-10 00:34 spark_output2/part-00017
-rw-r--r--   2 negin_shams supergroup    2955987 2022-04-10 00:34 spark_output2/part-00018
-rw-r--r--   2 negin_shams supergroup    2969130 2022-04-10 00:34 spark_output2/part-00019
-rw-r--r--   2 negin_shams supergroup    2975146 2022-04-10 00:34 spark_output2/part-00020
-rw-r--r--   2 negin_shams supergroup    2987590 2022-04-10 00:34 spark_output2/part-00021
-rw-r--r--   2 negin_shams supergroup    2957224 2022-04-10 00:34 spark_output2/part-00022
-rw-r--r--   2 negin_shams supergroup    2966523 2022-04-10 00:34 spark_output2/part-00023
-rw-r--r--   2 negin_shams supergroup    2978826 2022-04-10 00:34 spark_output2/part-00024
-rw-r--r--   2 negin_shams supergroup    2962277 2022-04-10 00:34 spark_output2/part-00025
-rw-r--r--   2 negin_shams supergroup    2977503 2022-04-10 00:34 spark_output2/part-00026
-rw-r--r--   2 negin_shams supergroup    2965085 2022-04-10 00:34 spark_output2/part-00027
-rw-r--r--   2 negin_shams supergroup    2966886 2022-04-10 00:34 spark_output2/part-00028
-rw-r--r--   2 negin_shams supergroup    2970102 2022-04-10 00:35 spark_output2/part-00029
-rw-r--r--   2 negin_shams supergroup    2979374 2022-04-10 00:35 spark_output2/part-00030
-rw-r--r--   2 negin_shams supergroup    2973627 2022-04-10 00:35 spark_output2/part-00031
-rw-r--r--   2 negin_shams supergroup    2955028 2022-04-10 00:35 spark_output2/part-00032
-rw-r--r--   2 negin_shams supergroup    2974263 2022-04-10 00:35 spark_output2/part-00033
-rw-r--r--   2 negin_shams supergroup    2982948 2022-04-10 00:35 spark_output2/part-00034
-rw-r--r--   2 negin_shams supergroup    2970874 2022-04-10 00:35 spark_output2/part-00035
-rw-r--r--   2 negin_shams supergroup    2972445 2022-04-10 00:35 spark_output2/part-00036
-rw-r--r--   2 negin_shams supergroup    2964767 2022-04-10 00:35 spark_output2/part-00037
-rw-r--r--   2 negin_shams supergroup    2964570 2022-04-10 00:35 spark_output2/part-00038
```

شکل ۱۸: محتوای پوشه ایجاد شده

```
negin_shams@MasterPC:~$ spark-submit    --master yarn    --deploy-mode client  pyspark_example2.py

2.966210013255477e-05
```

شکل ۱۹: زمان اجرای فایل ۵ گیگابایتی

- ✓ فایل ۱۰ گیگابایتی

نتایج اجرای این برنامه در پوشه spark_output_3 ذخیره شده است و حاوی ۷۹ فایل parquet است. محتوای این پوشه و مدت زمان اجرای برنامه به‌ترتیب در *شکل ۲۰* و *شکل ۲۱* قابل مشاهده است.

```
-rw-r--r--   2 negin_shams supergroup    1982200 2022-04-10 18:15 spark_output_3/part-00049
-rw-r--r--   2 negin_shams supergroup    1984500 2022-04-10 18:15 spark_output_3/part-00050
-rw-r--r--   2 negin_shams supergroup    1980131 2022-04-10 18:15 spark_output_3/part-00051
-rw-r--r--   2 negin_shams supergroup    1989913 2022-04-10 18:15 spark_output_3/part-00052
-rw-r--r--   2 negin_shams supergroup    1973738 2022-04-10 18:15 spark_output_3/part-00053
-rw-r--r--   2 negin_shams supergroup    1995074 2022-04-10 18:15 spark_output_3/part-00054
-rw-r--r--   2 negin_shams supergroup    1984263 2022-04-10 18:15 spark_output_3/part-00055
-rw-r--r--   2 negin_shams supergroup    1981244 2022-04-10 18:15 spark_output_3/part-00056
-rw-r--r--   2 negin_shams supergroup    1980255 2022-04-10 18:15 spark_output_3/part-00057
-rw-r--r--   2 negin_shams supergroup    1972799 2022-04-10 18:15 spark_output_3/part-00058
-rw-r--r--   2 negin_shams supergroup    1988355 2022-04-10 18:15 spark_output_3/part-00059
-rw-r--r--   2 negin_shams supergroup    1982563 2022-04-10 18:15 spark_output_3/part-00060
-rw-r--r--   2 negin_shams supergroup    1984602 2022-04-10 18:15 spark_output_3/part-00061
-rw-r--r--   2 negin_shams supergroup    1995763 2022-04-10 18:15 spark_output_3/part-00062
-rw-r--r--   2 negin_shams supergroup    1986107 2022-04-10 18:15 spark_output_3/part-00063
-rw-r--r--   2 negin_shams supergroup    1984364 2022-04-10 18:15 spark_output_3/part-00064
-rw-r--r--   2 negin_shams supergroup    1978320 2022-04-10 18:15 spark_output_3/part-00065
-rw-r--r--   2 negin_shams supergroup    3041952 2022-04-10 18:15 spark_output_3/part-00066
-rw-r--r--   2 negin_shams supergroup    1974253 2022-04-10 18:15 spark_output_3/part-00067
-rw-r--r--   2 negin_shams supergroup    1987152 2022-04-10 18:15 spark_output_3/part-00068
-rw-r--r--   2 negin_shams supergroup    1989713 2022-04-10 18:15 spark_output_3/part-00069
-rw-r--r--   2 negin_shams supergroup    2459620 2022-04-10 18:15 spark_output_3/part-00070
-rw-r--r--   2 negin_shams supergroup    1989176 2022-04-10 18:15 spark_output_3/part-00071
-rw-r--r--   2 negin_shams supergroup    1979380 2022-04-10 18:15 spark_output_3/part-00072
-rw-r--r--   2 negin_shams supergroup    1987934 2022-04-10 18:15 spark_output_3/part-00073
-rw-r--r--   2 negin_shams supergroup    2057584 2022-04-10 18:15 spark_output_3/part-00074
-rw-r--r--   2 negin_shams supergroup    1983475 2022-04-10 18:15 spark_output_3/part-00075
-rw-r--r--   2 negin_shams supergroup    2007104 2022-04-10 18:15 spark_output_3/part-00076
-rw-r--r--   2 negin_shams supergroup    1974005 2022-04-10 18:15 spark_output_3/part-00077
-rw-r--r--   2 negin_shams supergroup    1983837 2022-04-10 18:15 spark_output_3/part-00078
-rw-r--r--   2 negin_shams supergroup    1997568 2022-04-10 18:15 spark_output_3/part-00079
```

شکل ۲۰: محتوای پوشه ایجاد شده

```
negin_shams@MasterPC:~$ spark-submit    --master yarn    --deploy-mode client  pyspark_example3.py

6.736692739650607e-05
```

شکل ۲۱: زمان اجرای فایل ۱۰ گیگابایتی

- ✓ فایل ۱۲ گیگابایتی
- ✓ نتایج اجرای این برنامه در پوشه spark_output_4 ذخیره شده است و حاوی ۹۲ فایل parquet است. محتوای این پوشه و مدت زمان اجرای برنامه به‌ترتیب در *شکل ۲۲* قابل مشاهده است.

```
-rw-r--r--   2 negin_shams supergroup    1705730 2022-04-10 18:47 spark_output_4/part-00076
-rw-r--r--   2 negin_shams supergroup    1705320 2022-04-10 18:47 spark_output_4/part-00077
-rw-r--r--   2 negin_shams supergroup    1708087 2022-04-10 18:47 spark_output_4/part-00078
-rw-r--r--   2 negin_shams supergroup    1725360 2022-04-10 18:47 spark_output_4/part-00079
-rw-r--r--   2 negin_shams supergroup    1720943 2022-04-10 18:47 spark_output_4/part-00080
-rw-r--r--   2 negin_shams supergroup    1703321 2022-04-10 18:47 spark_output_4/part-00081
-rw-r--r--   2 negin_shams supergroup    1716078 2022-04-10 18:47 spark_output_4/part-00082
-rw-r--r--   2 negin_shams supergroup    1719098 2022-04-10 18:47 spark_output_4/part-00083
-rw-r--r--   2 negin_shams supergroup    1722582 2022-04-10 18:48 spark_output_4/part-00084
-rw-r--r--   2 negin_shams supergroup    1700612 2022-04-10 18:48 spark_output_4/part-00085
-rw-r--r--   2 negin_shams supergroup    1707429 2022-04-10 18:48 spark_output_4/part-00086
-rw-r--r--   2 negin_shams supergroup    1703347 2022-04-10 18:48 spark_output_4/part-00087
-rw-r--r--   2 negin_shams supergroup    1708560 2022-04-10 18:48 spark_output_4/part-00088
-rw-r--r--   2 negin_shams supergroup    1729925 2022-04-10 18:48 spark_output_4/part-00089
-rw-r--r--   2 negin_shams supergroup    1703324 2022-04-10 18:48 spark_output_4/part-00090
-rw-r--r--   2 negin_shams supergroup    1710839 2022-04-10 18:48 spark_output_4/part-00091
-rw-r--r--   2 negin_shams supergroup    1700837 2022-04-10 18:48 spark_output_4/part-00092
negin_shams@MasterPC:~$
```

شکل ۲۲: محتوای پوشه ایجاد شده

```
negin_shams@MasterPC:~$ spark-submit    --master yarn    --deploy-mode client  pyspark_example4.py
0.0051714420551434 16
```

شکل ۲۳: زمان اجرای فایل ۱۲ گیگابایتی

- **اسپارک (cluster)**

برای اجرا در حالت cluster لازم است است مقدار مربوط به آرگومان deploy-mode به cluster تغییر کند. *شکل ۲۴* نشان دهنده‌ی دستور استفاده شده برای اجرا در حالت cluster است.

```
shams@MasterPC:~$ spark-submit    --master yarn    --deploy-mode cluster  cluster_pyspark.py
```

شکل ۲۴: دستور لازم برای اجرا در حالت کلاستر

برای این‌که فایل در این حالت اجرا شود باید تغییراتی در کد مربوط به شمارش کلمات ایجاد شود. برای مثال آدرس مربوط به فضای HDFS فایل قرار گیرد. هم‌چنین در پایان برنامه با استفاده از دستور stop سشن متوقف می‌شود.

```
negin_shams@MasterPC: ~
  GNU nano 4.8                                                cluster_pyspark.py
import sys
import pyspark
from pyspark.sql import SparkSession
from pyspark import SparkContext, SparkConf
import timeit

if __name__ == "__main__":
        start = timeit.timeit()
        sc = SparkSession.builder.appName('SparkWordCount').getOrCreate()
        words = sc.sparkContext.textFile("hdfs:/user/negin_shams/word_count_data.txt").flatMap(lambda line: line.split(" "))

        wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)

        # save the counts to output
        wordCounts.saveAsTextFile("spark_output_c1/")
        end = timeit.timeit()
        print(end-start)
        sc.stop()
```

شکل ۲۵: کد استفاده شده برای حالت کلاستر

با توجه به این‌که در حالت کلاستر پس از ارسال درخواست، کلاینت منتظر دریافت پاسخ نمی‌ماند بنابراین نمی‌توان اطلاعات زمان اجرای برنامه را از طریق چاپ کردن زمان‌ها به‌دست آورد.

- **مقایسه‌ی هدوپ و اسپارک**

اگرچه زمان اجرای برنامه‌ها براساس شرایطی مانند بار وارد بر سیستم متفاوت است اما با توجه به زمان‌های به‌دست آمده، به‌طور کلی مدت‌زمان اجرای اسپارک کمتر از هدوپ می‌باشد.

این موضوع به این دلیل است که هدوپ عملیات خواندن و نوشتن داده‌ها را بر روی HDFS انجام می‌دهد در صورتی‌که اسپارک این عملیات را در حافظه اصلی و با استفاده از RDD انجام می‌دهد [۱].

- **مقایسه حالت کلاینت و کلاستر اسپارک**

در حالت کلاستر یک کلاینت درخواست خود را submit می‌کند و سپس این درخواست امکان اجرا بر روی هر یک از کلاینت‌های موجود را خواهد داشت. کلاینت ارسال کننده‌ی درخواست پس از آن می‌تواند به کار دیگری بپردازد. در مواردی که زمان اجرای برنامه طولانی است استفادع از این حالت سودمند خواهد بود. بنابراین کلاینت تنها درخواست را صادر نموده و دیگر نیازی به انتظار برای دریافت پاسخ نخواهد داشت و در این مدت می‌تواند آفلاین باشد یا عملیات دیگری انجام دهد. برخلاف حالت کلاستر، در حالت کلاینت خود کلاینت درایور را راه‌اندازی نموده و تا پایان عملیات در دسترس باقی می‌ماند. در مواردی که نیاز است تا پایان اجرای برنامه اطلاعاتی از وضعیت آن دریافت شود، بهتر است از این حالت استفاده شود. با این حال با توجه به این‌که درایور بر روی ماشین محلی قرار دارد و کلاینت آن را اجرا می‌کند، با خراب شدن آن ماشین برنامه نیز از بین می‌رود. به‌همین دلیل برای برنامه‌های بزرگ معمولا از این حالت استفاده نمی‌شود [۲].

مراجع

[1] https://www.geeksforgeeks.org/difference-between-hadoop-and-spark

[2] https://blog.knoldus.com/cluster-vs-client-execution-modes-for-a-spark-application