Big Data Analytics Techniques and Applications

Homework 3

Name: 陳以瑄 Student ID:109705001

1. The process of building up Hadoop environment

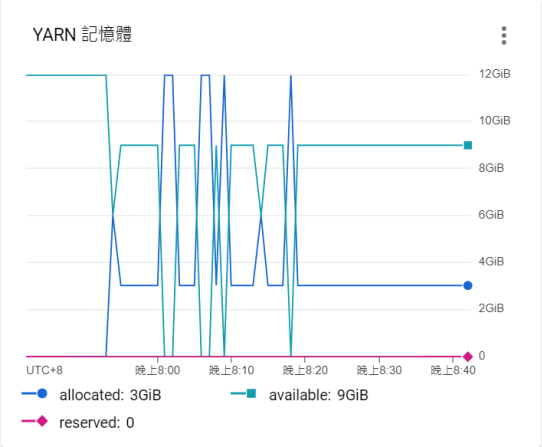
   I use Google Cloud Platform and follow TA's instructions to use Dataproc API. And this is my setting:

區域	asia-east1
區域	asia-east1-a
自動調度資源	關閉
Dataproc Metastore	無
排定刪除作業	關閉
主要節點	標準 (1 個主要節點，N 個工作站)
機器類型	n1-standard-4
GPU 數量	0
主要磁碟類型	pd-standard
主要磁碟大小	500GB
本機 **SSD** 數	0
工作站節點數	2
機器類型	n1-standard-2
GPU 數量	0
主要磁碟類型	pd-standard
主要磁碟大小	500GB
本機 **SSD** 數	0
次要工作站節點	0

   Then, I write my code through the "Web Interfaces >> Jupyter".

   And the code is under file "GCS", using PySpark kernel.

2. The usage of YARN memory (Attach some line plots).
   I wrote my homework in 2 days
   First day



   Second day

3. Descriptions of how you solve each question in detail.
   Q1: Implement a program to calculate the average occurrences of each word in a sentence in the attached article.
   - Source code

     Part 1. Set up the configuration for PySpark to connect to Hadoop

```
from pyspark.sql import SparkSession
from pyspark import SparkConf
from pyspark.sql.functions import col
import pandas as pd
```

```
conf = SparkConf()
conf.setMaster("yarn")
conf.set("spark.hadoop.fs.defaultFS", "10.140.0.4:9866")
conf.set("spark.hadoop.yarn.resourcemanager.hostname", "10.140.0.4")
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

     Part 2. Read data

```
1 ori_txt = spark.read.text(\
2 'gs://[redacted]/Youvegottofindwhatyoulove.txt')
3 ori_txt = ori_txt.filter(col("value").isNotNull()).filter(col("value") != "")
```

     Part 3. Word counts

```
1 word = ori_txt.rdd.flatMap(lambda line: line.value.split())
2 word_counts = word.map(lambda w: (w.lower(), 1))\
3               .reduceByKey(lambda count1, count2: count1 + count2)
4 top30 = word_counts.takeOrdered(30, lambda x: -x[1])
5 sentences = ori_txt.rdd.flatMap(lambda x: x.value.split(". ")).count()
6 avg_counts = [(word, count / sentences) for word, count in top30]
7 df_top30 = pd.DataFrame(avg_counts, columns =['word','avg_counts'])
8 display(df_top30)
```

   - How I solve
     1. Read the .txt into Spark DataFrame (Part2 line 1).
     2. Since the original .txt have empty line between lines, so I filter those empty lines by .filter() function (Part2 line 3).
     3. Then split the words by .flatMap() function (Part3 line 1).

4. Use .map() function to produce key-value pairs. Since I think the letter case should be ignore when doing word-counting, I use .lower() function to change the key to lowercase (Part3 line 2).
5. Use .reduceByKey() function to aggregate each word's occurrences (Part3 line 3).
6. Use .takeOrdered() function to select and sort the words that have top 30 counts(Part3 line4).
7. Since average occurrences is the word occurrences divided by the number of sentences, I use .flatMap() function to calculate the number of sentences (Part3 line5).
8. Then count the average occurrences (Part3 line6).
9. Save the result as dataframe and display it.

- Result

	word	avg_counts						
0	the	0.680851	10	my	0.212766	20	as	0.106383
1	i	0.609929	11	you	0.205674	21	what	0.106383
2	to	0.503546	12	is	0.198582	22	out	0.099291
3	and	0.468085	13	had	0.156028	23	but	0.099291
4	was	0.333333	14	with	0.127660	24	be	0.092199
5	a	0.326241	15	for	0.120567	25	from	0.092199
6	it	0.319149	16	so	0.120567	26	on	0.092199
7	of	0.290780	17	have	0.120567	27	me	0.085106
8	that	0.269504	18	your	0.113475	28	when	0.085106
9	in	0.241135	19	all	0.113475	29	at	0.078014

As the table shows, most of them are function words, such as "the", "was", "and", etc. Also, the word in different tense, such as "have" and "had", are separate in two different categories. If I did some NLP preprocessing, like remove stop words and doing lemmatization, the result will be more content words.

Q2: In YARN cluster mode, implement a program to calculate the average amount in credit card trips and cash trips for different numbers of passengers, ranging from one to four passengers in 2018/10 NYC Yellow Taxi trip data.

- Source code

  Part1. Set up the configuration for PySpark to connect to Hadoop

```
import pandas as pd
import numpy as np
from pyspark.sql import SparkSession
from pyspark import SparkConf
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,FloatType
from pyspark.sql.functions import when,col,avg
```

```
conf = SparkConf()
conf.setMaster("yarn")
conf.set("spark.hadoop.fs.defaultFS", "10.140.0.4:9866")
conf.set("spark.hadoop.yarn.resourcemanager.hostname", "10.140.0.4")
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

  Part2. Read data with user define schema

```
customSchema = StructType([
    StructField("VendorID", IntegerType(), True),
    StructField("tpep_pickup_datetime", StringType(), True),
    StructField("tpep_dropoff_datetime", StringType(), True),
    StructField("passenger_count", FloatType(), True),
    StructField("trip_distance", FloatType(), True),
    StructField("RatecodeID", FloatType(), True),
    StructField("store_and_fwd_flag", StringType(), True),
    StructField("PULocationID", IntegerType(), True),
    StructField("DOLocationID", IntegerType(), True),
    StructField("payment_type", StringType(), True),
    StructField("fare_amount", FloatType(), True),
    StructField("extra", FloatType(), True),
    StructField("mta_tax", FloatType(), True),
    StructField("tip_amount", FloatType(), True),
    StructField("tolls_amount", FloatType(), True),
    StructField("improvement_surcharge", FloatType(), True),
    StructField("total_amount", FloatType(), True),
    StructField("congestion_surcharge", FloatType(), True),
    StructField("airport_fee", FloatType(), True),
])
df=spark.read.parquet(\
'gs://dataproc-staging-asia-east1-853194707086-pb19ffdw/yellow_tripdata_2018-10.parquet',\
header=True, schema=customSchema)
```

  Part3. Calculate the average amount

```
new_df = df.select("payment_type", "passenger_count","total_amount")

# payment_type
# 1= Credit card
# 2= Cash
credit_df = new_df.filter(col("payment_type") == "1.0")
cash_df = new_df.filter(col("payment_type") == "2.0")

credit_avg_df = credit_df.groupBy("passenger_count")\
                .agg(avg("total_amount").alias("Credit_avg_amount"))
cash_avg_df = cash_df.groupBy("passenger_count")\
            .agg(avg("total_amount").alias("Cash_avg_amount"))

joined_df = credit_avg_df.join(cash_avg_df, "passenger_count")
filtered_joined_df = joined_df.filter(col("passenger_count") >= 1)\
                    .filter(col("passenger_count") <= 4)\
                    .orderBy(col("passenger_count"), ascending=True)
filtered_joined_df.show()
```

- How I solve:
    1. Use .select() function to select the columns I need(Part3 line 2).
    2. The data description says that in “payment_type” column, value 1 means pay by credit card and value 2 means pay in cash. So I use .filter() function to separate these two kind of payment in two dataframes (Part3 line 7,8).
    3. Using .groupby() function to split the data in different group by the “passenger _count” column. Then, count the average amount of each group by .agg(avg()) function (Part3 line 10~13). In this case, I just ignore those row that “passenger _count” is “null”.
    4. Merge the result of credit and cash together, then show the result.
- Result

```
+---------------+------------------+------------------+
|passenger_count| Credit_avg_amount|   Cash_avg_amount|
+---------------+------------------+------------------+
|            1.0|18.095631110444963|13.682840694132494|
|            2.0|18.823252006344298|14.728853925008947|
|            3.0|18.398010393293596|14.810038897839949|
|            4.0|  18.6797514241931|15.506981138445854|
+---------------+------------------+------------------+
```

Q3: Referring to Q2, monitor HDFS and YARN metrics through HTTP API.
Please provide screenshots and observations regarding the metrics in your report.

- HDFS metrics
    1. NameNode Metrics

Part1. Summary on GUI

Security is off.

Safemode is off.

63 files and directories, 12 blocks (12 replicated blocks, 0 erasure coded block groups) = 75 total filesystem object(s).

Heap Memory used 125.62 MB of 228.19 MB Heap Memory. Max Heap Memory is 2.9 GB.

Non Heap Memory used 56.96 MB of 58.07 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	968.8 GB
Configured Remote Capacity:	0 B
DFS Used:	9.39 MB (0%)
Non DFS Used:	22.94 GB
DFS Remaining:	945.82 GB (97.63%)
Block Pool Used:	9.39 MB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	2 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Fri Apr 28 16:19:37 +0800 2023
Last Checkpoint Time	Fri Apr 28 15:36:20 +0800 2023
Enabled Erasure Coding Policies	RS-6-3-1024k

Part2. Detail view through NataNode API

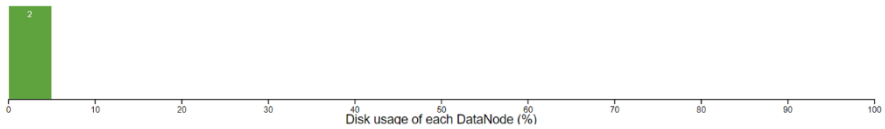

```
{
  "name" : "Hadoop:service=NameNode,name=FSNamesystemState",
  "modelerType" : "org.apache.hadoop.hdfs.server.namenode.FSNamesystem",
  "FilesTotal" : 63,
  "CapacityTotal" : 1040241205248,
  "CapacityUsed" : 9846784,
  "CapacityRemaining" : 1015555563520,
  "ProvidedCapacityTotal" : 0,
  "TotalLoad" : 4,
  "SnapshotStats" : "{\"SnapshottableDirectories\":0,\"Snapshots\":0}",
  "NumEncryptionZones" : 0,
  "FsLockQueueLength" : 0,
  "BlocksTotal" : 12,
  "MaxObjects" : 0,
  "PendingReplicationBlocks" : 0,
  "PendingReconstructionBlocks" : 0,
  "UnderReplicatedBlocks" : 0,
  "LowRedundancyBlocks" : 0,
  "ScheduledReplicationBlocks" : 0,
  "PendingDeletionBlocks" : 0,
  "BlockDeletionStartTime" : 1682667363194,
  "FSState" : "Operational",
  "NumLiveDataNodes" : 2,
  "NumDeadDataNodes" : 0,
  "NumDecomLiveDataNodes" : 0,
  "NumDecomDeadDataNodes" : 0,
  "NumInServiceLiveDataNodes" : 2,
  "VolumeFailuresTotal" : 0,
  "EstimatedCapacityLostTotal" : 0,
  "NumDecommissioningDataNodes" : 0,
  "NumStaleDataNodes" : 0,
  "NumStaleStorages" : 0,
  "CorruptBlocks" : 0,
  "MissingBlocks" : 0,
  "BlockCapacity" : 8388608,
```


Observation:

The capacity remains a lot. No missing or failure.

2. DataNode Metrics

Part1. Summary on GUI



In operation

Show 25 entries Search:

Node	Http Address	Last contact	Last Block Report	Capacity	Blocks	Block pool used	Version
✔cluster-[redacted]	http://cluster-[redacted]	0s	1m	484.4 GB	12	4.7 MB (0%)	3.2.3
✔cluster-[redacted]	http://cluster[redacted]	0s	1m	484.4 GB	12	4.7 MB (0%)	3.2.3

Part2. Detail view through DataNode API

First node


```
"name" : "Hadoop:service=DataNode,name=FSDatasetState",
"modelerType" : "FSDatasetState",
"tag.Context" : "FSDatasetState",
"tag.StorageInfo" : "FSDataset{dirpath='[/hadoop/dfs/data]'}"
"tag.Hostname" : "cluster-hw3-w-0",
"Capacity" : 520120602624,
"DfsUsed" : 4923392,
"Remaining" : 507779571712,
"NumFailedVolumes" : 0,
"LastVolumeFailureDate" : 0,
"EstimatedCapacityLostTotal" : 0,
"CacheUsed" : 0,
"CacheCapacity" : 0,
"NumBlocksCached" : 0,
"NumBlocksFailedToCache" : 0,
"NumBlocksFailedToUnCache" : 0
```

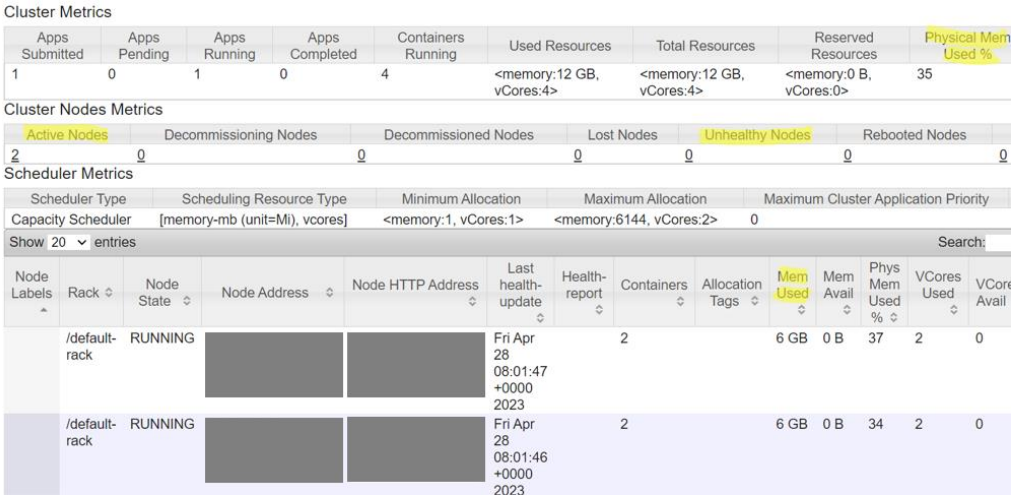

Second node


```
"name" : "Hadoop:service=DataNode,name=FSDatasetState",
"modelerType" : "FSDatasetState",
"tag.Context" : "FSDatasetState",
"tag.StorageInfo" : "FSDataset{dirpath='[/hadoop/dfs/data]'}"
"tag.Hostname" : "cluster-hw3-w-1",
"Capacity" : 520120602624,
"DfsUsed" : 4923392,
"Remaining" : 507775991808,
"NumFailedVolumes" : 0,
"LastVolumeFailureDate" : 0,
"EstimatedCapacityLostTotal" : 0,
"CacheUsed" : 0,
"CacheCapacity" : 0,
"NumBlocksCached" : 0,
"NumBlocksFailedToCache" : 0,
"NumBlocksFailedToUnCache" : 0
```


Observation:

The disk space is enough. No failed volumes.

- YARN metrics



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources	Reserved Resources	Physical Mem Used %
1	0	1	0	4	<memory:12 GB, vCores:4>	<memory:12 GB, vCores:4>	<memory:0 B, vCores:0>	35

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1, vCores:1>	<memory:6144, vCores:2>	0

Show 20 entries Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Allocation Tags	Mem Used	Mem Avail	Phys Mem Used %	VCores Used	VCore Avail
	/default-rack	RUNNING	[redacted]	[redacted]	Fri Apr 28 08:01:47 +0000 2023		2		6 GB	0 B	37	2	0
	/default-rack	RUNNING	[redacted]	[redacted]	Fri Apr 28 08:01:46 +0000 2023		2		6 GB	0 B	34	2	0

Observation:

The cluster is healthy. The apps use the whole memory.