Batch Data Analytics

201374125

Middleware Configuration

This report uses Apache Spark Resilient Distributed Datasets (RDDs) with operations run on a Google Dataproc cluster. The Dataproc cluster was created in a minimal form, with one master node and two worker nodes, each running `n1-standard-2` which has 2 vCPUs, 7.5GB memory, and 100GB disk space. The script itself is written in python using `pyspark`, and can either be ran locally by executing `$SPARK_HOME/bin/spark-submit a1_spark.py`, or using the Dataproc cluster with `gcloud` command line utilities: `gcloud dataproc jobs submit pyspark a1_spark.py`. The cluster name, region and credentials used in this command are all preset using the `gcloud` tool. The CSV datafile is hosted directly on a Google Cloud server when running through Dataproc.

Data Analytic Design

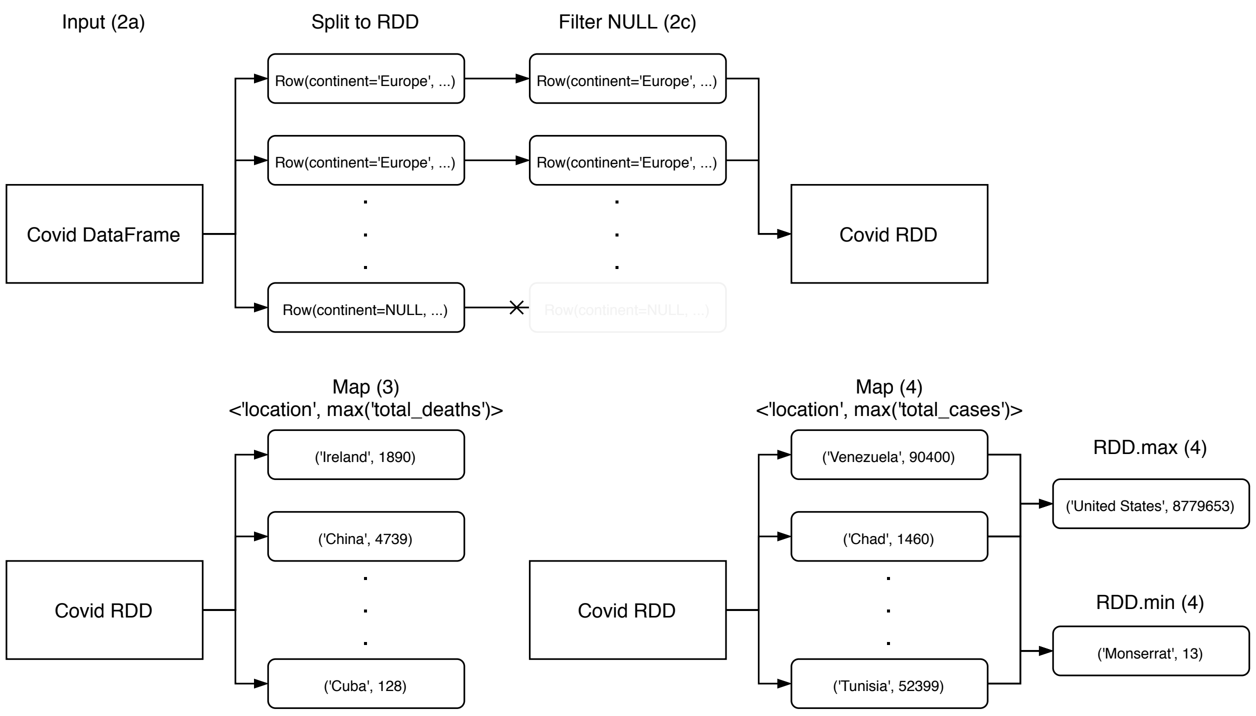
The `pyspark` SQL module provided `SparkSession` which has the ability to create a `Dataframe` directly from `.csv` files using the `spark.read.csv()` function where `spark = SparkSession(...)`. The `DataFrame` is typically used for structured processing and resembles tables from database systems. The head of the dataframe may be viewed using `df.show()`, and the schema printed with `df.printSchema()`. `DataFrame`content can be easily moved to a `pyspark.RDD` of `Row` (rows of data in the dataframe) by calling `DataFrame.rdd`. From this `rdd` any `rdd` function may be performed on each row.

The `rdd.filter` function was used to remove any empty values present in any row by converting each `Row` into a dictionary and checking for empty values `rdd.filter(lambda row: None not in row.asDict().values())`.

The highest total deaths for each country was found by first mapping the Rows into key values pairs $< k, v >$ where $k =' location'$ and $v =' total\_deaths'$. `rdd.map` was again used to take the `max` value for each pair.

Finally the highest total cases for each country was found by mapping Rows into key values pairs $k =' location'$, $v =' total\_cases'$. `rdd.map` again used to get max value for each pair. Following this, the `rdd.max` and `rdd.min` functions were used to get the maximum and minimum values for this `RDD`.

The following diagram gives an overview of this dataflow:



Results and Discussion

This DataFrame has 53,087 rows, filtering removed 13,113 rows with empty values, leaving 39,974 rows.

Highest total death cases per country:

Country	Max Total Deaths
United States	226723
Brazil	157946
India	120010
Mexico	89814
United Kingdom	45365
Italy	37700
France	35541
Spain	35298
Peru	34257
Iran	33299
Colombia	30565
Argentina	29730
Russia	26589
South Africa	19053
Chile	14026
Indonesia	13512

Country	Max Total Deaths
Ecuador	12588
Belgium	10921
Iraq	10724
Germany	10183

Max total cases by country:

United States: 8,779,653

Min total cases by country:

Montserrat: 13

Table showing 20 countries:

Country	Max total cases
United States	8779653
India	7990322
Brazil	5439641
Russia	1547774
France	1198695
Spain	1116738
Argentina	1116596
Colombia	1033218
United Kingdom	917575
Mexico	901268
Peru	892497
South Africa	717851
Iran	581824
Italy	564778
Chile	504525
Germany	464239
Iraq	459908
Bangladesh	401586
Indonesia	396454
Philippines	373144

Conclusions and Recommendations

Google Dataproc allows for building large scale clusters with a specified number of worker nodes. For larger scale projects the number of workers could be increased depending on the volume of data. The power of each machine could also be increased to speed up processes on individual partitions, Dataproc has machines with up to 224 cores and 224 GB memory. Disk space could also be increased.

Script Appendix

```python
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession.builder.appName('Covid Dataframe').getOrCreate()
print(spark.sparkContext.appName)

# 2a: Load csv to DataFrame with header=True
# either use local or cloud stored data depending on use of dataproc
# or running locally
try:
    # gcloud dataproc jobs submit pyspark a1_spark.py --cluster=spark-cluster
    bucket = 'dataproc-staging-europe-west1-239259997526-zdg428u9'
    csv_path = f'gs://{bucket}/covid19.csv'
    df = spark.read.csv(csv_path, sep=',', header=True)
except Exception as e:
    # $SPARK_HOME/bin/spark-submit a1_spark.py
    print(e, ": Running Locally.")
    csv_path = './data/covid19.csv'
    df = spark.read.csv(csv_path, sep=',', header=True)

# 2b: Show as table, print schema
df.show()
df.printSchema()

# 2c: RDD Filter function
# convert df to rdd
rdd = df.rdd

# before filter
rdd.toDF().count()  # total rows
rdd.toDF().select(  # NULL per column
    [F.count(F.when(F.col(c).isNull(), c)).alias(c) for c in df.columns]
).show()

# Filter any row containing missing value
rdd = rdd.filter(lambda row: None not in row.asDict().values())
```

```
# after filter
rdd.toDF().count()  # total rows
rdd.toDF().select(  # NULL per column
    [F.count(F.when(F.col(c).isNull(), c)).alias(c) for c in df.columns]
).show()

# 3: Aggregate + GroupBy highest Total Death per country
rdd_totaldeath = rdd.map(lambda x: (x['location'], int(x['total_deaths'])))\
    .groupByKey()\
    .mapValues(list)\
    .map(lambda x: (x[0], max(x[1])))
rdd_totaldeath = spark.createDataFrame(rdd_totaldeath)\
    .orderBy('_2', ascending=False)
print(rdd_totaldeath.show())

# 4: RDD max and min functions
rdd_totalcases = rdd.map(lambda x: (x['location'], int(x['total_cases'])))\
    .groupByKey()\
    .mapValues(list)\
    .map(lambda x: (x[0], max(x[1])))
print(rdd_totalcases.max(lambda x: x[1]))  # max
print(rdd_totalcases.min(lambda x: x[1]))  # min

# (shown list of at least 20 countries)
rdd_totalcases = spark.createDataFrame(rdd_totalcases)\
    .orderBy('_2', ascending=False)
print(rdd_totalcases.show())
```