Nghi Le
K1898955
Big Data Technologies

Coursework 1

Task 1:

a.) Data from the transportation domain can definitely be classified as Big Data because it satisfy all 5 characteristics of Big data:

- **Volume:** the data is usually gathered from tracking sensors of massive truck, shipments, and any type of vehicle fleets. Hence, the volume is large
- **Velocity:** the data is also usually real-time in order to support on-the-fly analysis and decision-making.
- **Variety:** data can be in many forms. For example, in diagnostics, the data are temperature, RPM of the engine of the vehicles and so on. While in shipping, the data can be the location of the truck or the temperature and humidity in the food shipment.
- **Veracity:** understandably, the data from sensors can be missing due to various failures or corrupted by noises from the outside environment.
- **Value:** transportation data are useful in improving traffic congestion, saving businesses' costs in maintenance and logistics. This adds real value to businesses and society at scale.

b.) Big data has lots of value but also has significant challenges as well:

- **Volume:** because the data is big, it is challenging to store as well. The data infrastructure usually has to be complex and robust enough to handle failure, and to allow fast access for analysis and decision-making.
- **Velocity:** the data speed is high and in real-time so it is difficult to analyze and process them carefully.
- **Variety:** different sources and formats of data requires complex integration, transformation and storage requirements.
- **Veracity:** noisy and missing data is more difficult and costly to handle, and the information from them is not as useful as well.
- **Value:** the costs of handling and processing big data can be very high due to the above-mentioned challenges. Complete cost-analysis should be done to see where it is worth it to implement big data solutions.

Task 2:

a.) After the command is input, the Sqoop client imports the metadata of the table “adult” to examine the table. Then Sqoop will create multiple Map tasks and send them to Hadoop cluster to run in parallel on the mappers. The mappers will now import the data in parallel from the relational database to the distributed file system, with automatic load-balancing between themselves. The “-m8” option hint to Sqoop that 8 mappers should be used for this import job. Also, this import job obeys the query in the command and import the table “adult” with 2 columns “age” and “gender”.

b.) Three features of Sqoop that makes importing data to a distributed file system efficiently are:

1. Launch map tasks and importing them in parallel, which improve speed and efficiency for big data.
2. There are automatic balancing between the mappers to make importing extremely efficient.
3. There is also the ability to compress data to make them smaller in size for importing.

Task 3:

a.) A requirement for a function to be used in a combiner is that it has to be commutative and associative, in order for the results after both the combiner and reducer to be consistent. A mean function cannot be used in the combiner for this reason, since it is not associative. For example, mean(5, mean(7,9)) is not equal to mean(mean(5,7),9).

b.)

```
# I will answer these questions on the first 10 lines of the 'adult' dataset

from mrjob.job import MRJob

class MRmyjob(MRJob):

    def mapper(self, _, line):
            # Split data by ,
            data=line.split(", ")
            if len(data)>=2:
                # Assign the first term of the data to age
                age = data[0].strip()
                # Assign the second term of the data to work_class
                work_class = data[1]
                # Yield every age as value and each work_class as key
                yield work_class, age


    def reducer(self, work_class, list_of_values):
            # Convert the values to integer for arithmetic
            a=[int(i) for i in list_of_values]
            # Take the average
            avg_a = float(sum(a)) / len(a)
            # Yield the average, group by work_class
            yield work_class, avg_a

if __name__== "__main__":
    MRmyjob.run()
```

c.)

```
from mrjob.job import MRJob

class MRmyjob(MRJob):

    def mapper(self, _, line):
            # Split data by ,
            data=line.split(", ")
            if len(data)>=2:
                # Assign the first term of the data to age
                age = data[0].strip()
                # Assign the second term of the data to work_class
                work_class = data[1]
                # Yield every age as value and each work_class as key
                yield work_class, age

    # Combiner code:
    def combiner(self, work_class, list_of_values):
            # These steps are very much the same as the steps in reducer below
            # Convert the values to integer for arithmetic
            a=[int(i) for i in list_of_values]
            # Take the average
            avg_a = float(sum(a)) / len(a)
            # Yield the average, group by work_class
            yield work_class, avg_a

    def reducer(self, work_class, list_of_values):
            # Convert the values to integer for arithmetic
            a=[int(i) for i in list_of_values]
            # Take the average
            avg_a = float(sum(a)) / len(a)
            # Yield the average, group by work_class
            yield work_class, avg_a

if __name__== "__main__":
    MRmyjob.run()
```

d.) The output without using the combiner is:

```
"State-gov"  39.0
"Private"    39.714285714285715
"Self-emp-not-inc"  51.0
```

The output with the combiner is:

```
"State-gov"  39.0
"Private"    39.75
"Self-emp-not-inc"  51.0
```

The output with the combiner is wrong, evident by the "Private" key. The reason is that some of the key-value pairs was combined and take the mean of beforehand within the mapper itself, before being combined and take the mean of by the reducer.

Task 4:

a.) My code:

```
from mrjob.job import MRJob

class MRmyjob(MRJob):

    def mapper(self, _, line):
            # Split data
            data = line.split(', ')
            # Get the key
            join_id = data[0].strip()
            yield join_id, (data[1],data[2])

    def reducer(self, key, values):
            # Append to a list
            a = [i for i in values]
            # yield key and the combined list of a for each key
            yield key, a


if __name__== "__main__":
    MRmyjob.run()
```

Sample output:

```
"24547"     [["43", " Self-emp-inc"], ["Some-college", "Married-civ-spouse"]]
"24548"     [["32", " Private"], ["HS-grad", "Married-civ-spouse"]]
"24549"     [["37", " State-gov"], ["Assoc-acdm", "Divorced"]]
"2455"      [["48", " Private"], ["HS-grad", "Divorced"]]
"24550"     [["48", " Private"], ["Some-college", "Married-civ-spouse"]]
```