PART I

1. To emulate a live-stream of the traffic counter dataset, you are required to write a separate Python script that read 10 records (of each counter site - ignore the test site) every seconds from traffic counter data le and stores them as separate files (countdata1, countdata2, countdata3, etc.) in the streaming directory on which your application is listening.

Code -

In [1]:
```
import time
from itertools import groupby, count, islice
```

In [2]:
```
# 1.Read 10 records (of each counter site - ignore the test site) every 5 seconds from traffic counter data file
# and stores them as separate files (countdata1,countdata2, countdata3, etc.) in the streaming directory on which
# your application is listening.
```

In [3]:
```
inputFilePath = '/home/ishan/Desktop/Assignment3/dataset/per-vehicle-records-2020-01-31.csv'
outputDirectoryPath = '/home/ishan/Desktop/Assignment3/streaming/'
```

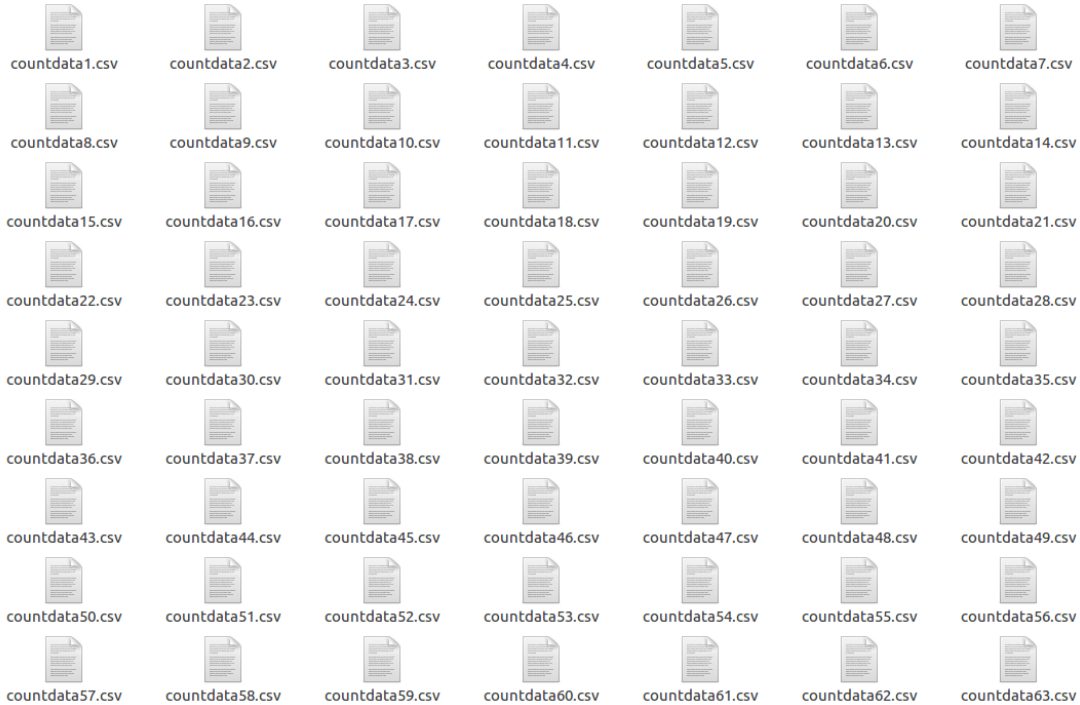
In [4]:
```
def divide_chunks(l, chunk_size):
    c = count()
    for _, g in groupby(l, lambda _: next(c) // chunk_size):
        yield g
```

In [5]:
```
def write_records(records, file_path):
    with open(file_path, 'w') as f:
        f.writelines(record + '\n' for record in records)
```

In [6]:
```
inputLines = (line.rstrip('\n') for line in open(inputFilePath))
# Skipping header
inputLines = islice(inputLines, 1, None)
# Filtering out Test sites
inputLines = filter(lambda x: "test" not in x.split(',')[10].lower(), inputLines)
# Dividing into chunks
inputChunks = divide_chunks(inputLines, 10)
```

In [7]:
```
counter = 0
for chunk in inputChunks:
    counter += 1
    write_records(chunk, '{}/countdata{}.csv'.format(outputDirectoryPath, counter))
    time.sleep(5)
```

Output - (List of files are generated in the directory specified)



2. Prepare the streaming application to read the data streams from the streaming directory using a batch length of seconds.
Code -

```
In [2]: cassandra_host = ['127.0.0.1']
cassandra_port = 9042
cassandra_keyspace = 'assignment3'
```

```
In [3]: inputDirectoryPath = '/home/ishan/Desktop/Assignment3/streaming/'

conf = (SparkConf()
        .setAppName("Assignment3")
        .set('spark.cassandra.connection.host', cassandra_host[0])
        .set('spark.cassandra.connection.port', cassandra_port)
        .set('spark.jars', '/home/ishan/Desktop/Assignment3/jars/spark-cassandra-connector_2.11-2.3.1.jar,/home/isha
        .setMaster('local[*]'))
sc = SparkContext.getOrCreate(conf = conf)
ssc = StreamingContext(sparkContext = sc, batchDuration = 5)
spark = SparkSession.builder.config(conf=conf).getOrCreate()

rows = ssc.textFileStream(inputDirectoryPath)
rows = rows.map(lambda row: row.split(','))
```

PART II
(Computations)

1. Show total number of counts (on each site of M50) by vehicle class.
Code -

```
In [8]: # 1. Show total number of counts (on each site of M50) by vehicle class.
```

```
In [9]: vehicleClassColIndex = 14

vehicleCountPerClass = (rows
                        .map(lambda row: (row[vehicleClassColIndex].strip('"'), 1))
                        .reduceByKey(lambda x, y: x + y))

def saveQ1(time, rdd):
    if not rdd.isEmpty():
        rowRDD = rdd.map(lambda row: Row(time=time, classname=row[0], count=row[1]))
        df = spark.createDataFrame(rowRDD)
        (df.write.format("org.apache.spark.sql.cassandra")
         .mode('append')
         .options(table="q1", keyspace=cassandra_keyspace)
         .save())

vehicleCountPerClass.foreachRDD(saveQ1)
```

Output -

```
cqlsh:assignment3> select * from q1 limit 15;

 time                     | classname | count
--------------------------+-----------+-------
 2020-04-22 20:40:35+0530 |       CAR |     9
 2020-04-22 20:40:35+0530 |       LGV |     1
 2020-04-22 20:39:05+0530 |       CAR |     7
 2020-04-22 20:39:05+0530 |   HGV_ART |     1
 2020-04-22 20:39:05+0530 |   HGV_RIG |     1
 2020-04-22 20:39:05+0530 |       LGV |     1
 2020-04-22 20:44:15+0530 |       CAR |     6
 2020-04-22 20:44:15+0530 |   HGV_ART |     1
 2020-04-22 20:44:15+0530 |       LGV |     3
 2020-04-22 20:40:30+0530 |       CAR |     6
 2020-04-22 20:40:30+0530 |   HGV_ART |     3
 2020-04-22 20:40:30+0530 |   HGV_RIG |     1
 2020-04-22 20:42:35+0530 |       CAR |     5
 2020-04-22 20:42:35+0530 |   HGV_ART |     2
 2020-04-22 20:42:35+0530 |       LGV |     3
```

2. Compute the average speed (on each site on M50) by vehicle class.
Code -

In [10]:
```
# 2. Compute the average speed (on each site on M50) by vehicle class.
```

In [11]:
```
vehicleClassColIndex = 14
speedColIndex = 18

averageSpeedPerClass = (rows
                        .map(lambda row: (row[vehicleClassColIndex].strip('"'), float(row[speedColIndex].strip('"'))))
                        .groupByKey()
                        .map(lambda row: (row[0], list(row[1])))
                        .map(lambda row: (row[0], sum(row[1]) / len(row[1]))))

def saveQ2(time, rdd):
    if not rdd.isEmpty():
        rowRDD = rdd.map(lambda row: Row(time=time, classname=row[0], avg_speed=row[1]))
        df = spark.createDataFrame(rowRDD)
        (df.write.format("org.apache.spark.sql.cassandra")
         .mode('append')
         .options(table="q2", keyspace=cassandra_keyspace)
         .save())

averageSpeedPerClass.foreachRDD(saveQ2)
```

Output -

```
cqlsh:assignment3> select * from q2 limit 15;

 time                      | classname | avg_speed
---------------------------+-----------+-----------
 2020-04-22 20:40:35+0530 |       CAR | 100.55556
 2020-04-22 20:40:35+0530 |       LGV |        84
 2020-04-22 20:39:05+0530 |       CAR |  77.71429
 2020-04-22 20:39:05+0530 |   HGV_ART |        70
 2020-04-22 20:39:05+0530 |   HGV_RIG |        58
 2020-04-22 20:39:05+0530 |       LGV |        75
 2020-04-22 20:44:15+0530 |       CAR | 108.33333
 2020-04-22 20:44:15+0530 |   HGV_ART |        88
 2020-04-22 20:44:15+0530 |       LGV |       102
 2020-04-22 20:40:30+0530 |       CAR | 121.33333
 2020-04-22 20:40:30+0530 |   HGV_ART |  89.66667
 2020-04-22 20:40:30+0530 |   HGV_RIG |        93
 2020-04-22 20:42:35+0530 |       CAR |      86.8
 2020-04-22 20:42:35+0530 |   HGV_ART |        86
 2020-04-22 20:42:35+0530 |       LGV |        91
```

3. Find the top 3 busiest counter sites on M50.
Code -

In [12]:
```
# 3. Find the top 3 busiest counter sites on M50.
```

In [13]:
```
counterSitesColIndex = 10

counterSitesPerClass = (rows
                        .map(lambda row: (row[counterSitesColIndex].strip('"'), 1))
                        .reduceByKey(lambda x, y: x + y)
                        .transform(lambda row: row.sortBy(lambda x: x[1], ascending=False)))

def saveQ3(time, rdd):
    if not rdd.isEmpty():
        rowRDD = rdd.map(lambda row: Row(time=time, counter_site=row[0], count=row[1]))
        df = spark.createDataFrame(rowRDD)
        # Limit top 3 values
        (df.limit(3).write.format("org.apache.spark.sql.cassandra")
         .mode('append')
         .options(table="q3", keyspace=cassandra_keyspace)
         .save())

counterSitesPerClass.foreachRDD(saveQ3)
```

Output -

```
cqlsh:assignment3> select * from q3 limit 15;

 time                     | counter_site | count
--------------------------+--------------+-------
 2020-04-22 20:40:35+0530 |    Northbound |    10
 2020-04-22 20:39:05+0530 |  Eastbound 1 |     2
 2020-04-22 20:39:05+0530 |  Eastbound 2 |     3
 2020-04-22 20:39:05+0530 |  Eastbound 3 |     2
 2020-04-22 20:44:15+0530 | Northbound 1 |     3
 2020-04-22 20:44:15+0530 | Southbound 1 |     5
 2020-04-22 20:44:15+0530 | Southbound 2 |     1
 2020-04-22 20:40:30+0530 |    Northbound |     8
 2020-04-22 20:40:30+0530 |    Southbound |     2
 2020-04-22 20:42:35+0530 |  Eastbound 3 |     5
 2020-04-22 20:42:35+0530 |  Westbound 2 |     2
 2020-04-22 20:42:35+0530 |  Westbound 3 |     1
 2020-04-22 20:41:50+0530 |  Westbound 2 |     5
 2020-04-22 20:41:50+0530 |  Westbound 3 |     3
 2020-04-22 20:41:50+0530 |  Westbound 4 |     1
```

4. Find total number of counts for HGVs on M50.
Code -

In [14]:
```
# 4. Find total number of counts for HGVs on M50.
```

In [15]:
```
vehicleClassColIndex = 14
counterSitesColIndex = 10

countsForHGVs = (rows
                 .filter(lambda row: "HGV" in row[vehicleClassColIndex])
                 .map(lambda row: (row[counterSitesColIndex].strip('"'), 1))
                 .reduceByKey(lambda x, y: x + y))

def saveQ4(time, rdd):
    if not rdd.isEmpty():
        rowRDD = rdd.map(lambda row: Row(time=time, classname=row[0], count=row[1]))
        df = spark.createDataFrame(rowRDD)
        (df.write.format("org.apache.spark.sql.cassandra")
         .mode('append')
         .options(table="q4", keyspace=cassandra_keyspace)
         .save())

countsForHGVs.foreachRDD(saveQ4)
```

Output -

```
cqlsh:assignment3> select * from q4 limit 15;

 time                     | classname    | count
--------------------------+--------------+-------
 2020-04-22 20:39:05+0530 |  Westbound 3 |     1
 2020-04-22 20:39:05+0530 |  Westbound 4 |     1
 2020-04-22 20:44:15+0530 | Southbound 1 |     1
 2020-04-22 20:40:30+0530 |    Northbound |     2
 2020-04-22 20:40:30+0530 |    Southbound |     2
 2020-04-22 20:42:35+0530 |  Westbound 2 |     1
 2020-04-22 20:42:35+0530 |  Westbound 3 |     1
 2020-04-22 20:41:50+0530 |  Westbound 2 |     2
 2020-04-22 20:41:50+0530 |  Westbound 3 |     1
 2020-04-22 20:39:20+0530 | Southbound 1 |     2
 2020-04-22 20:44:10+0530 | Northbound 1 |     2
 2020-04-22 20:44:10+0530 | Southbound 1 |     3
 2020-04-22 20:44:00+0530 |  Westbound 1 |     2
 2020-04-22 20:42:55+0530 |  Westbound 1 |     2
 2020-04-22 20:40:50+0530 |    Northbound |     4
```

PART III
(Cassandra setup and store)

1. Prepare cassandra data structures to store the results.
Code -

In [4]:
```
cassandra_create_queries = [
    "CREATE KEYSPACE IF NOT EXISTS " + cassandra_keyspace + " WITH replication = {'class':'SimpleStrategy', 'replica
    'use ' + cassandra_keyspace + ';',
    'create table if not exists q1 (time text, classname text, count int, primary key(time, classname));',
    'create table if not exists q2 (time text, classname text, avg_speed double, primary key(time, classname));',
    'create table if not exists q3 (time text, counter_site text, count int, primary key(time, counter_site));',
    'create table if not exists q4 (time text, classname text, count int, primary key(time, classname));'
]

def setupCassandra():
    cluster = Cluster(cassandra_host)
    session = cluster.connect()
    for query in cassandra_create_queries:
        print("Executing - {}".format(query))
        session.execute(query)
```

In [5]:
```
setupCassandra()
```

```
/home/ishan/.local/lib/python3.5/site-packages/ipykernel_launcher.py:11: DeprecationWarning: The 'warn' method is d
eprecated, use 'warning' instead
  # This is added back by InteractiveShellApp.init_path()
```

```
Executing - CREATE KEYSPACE IF NOT EXISTS assignment3 WITH replication = {'class':'SimpleStrategy', 'replication_fa
ctor':1};
Executing - use assignment3;
Executing - create table if not exists q1 (time text, classname text, count int, primary key(time, classname));
Executing - create table if not exists q2 (time text, classname text, avg_speed double, primary key(time, classnam
e));
Executing - create table if not exists q3 (time text, counter_site text, count int, primary key(time, counter_sit
e));
Executing - create table if not exists q4 (time text, classname text, count int, primary key(time, classname));
```

2. Prepare code for writing the results into the cassandra tables.
Code -

```
def saveQ1(time, rdd):
    if not rdd.isEmpty():
        rowRDD = rdd.map(lambda row: Row(time=time, classname=row[0], count=row[1]))
        df = spark.createDataFrame(rowRDD)
        (df.write.format("org.apache.spark.sql.cassandra")
         .mode('append')
         .options(table="q1", keyspace=cassandra_keyspace)
         .save())

vehicleCountPerClass.foreachRDD(saveQ1)
```