M5MS19 Big Data - Coursework Assignment

MD5 sums:

prox-fixed	1cd581d230a137a69c49df00ebc9f02c
prox-mobile	1d7cde7a7cc534051bb21c294bc27e01
bldg-measurements	8269c88fbad665a0ff76af08feb016bb
f2z2-haz	0b24258e88655591c6988d70b7f001e5

Map Reduce

Map Reduce: Question 1

Use the `prox-fixed` and `prox-mobile` datasets to produce a diagram that displays the number of unique prox-ids on each day.

A MapReduce program that extracts the number of unique prox-ids on each day is as follows.

1. Map receives a line from `prox-*.csv`, parses its `date` and `id` values, then writes the key-value pair `(date id, 1)`.
2. Reduce receives the key-value pairs sorted by `date id` (where the intervening space forms part of the key), then counts the number of unique ids that occur on each day. It outputs lines of the format `date n_unique_ids`.

The Reduce step's counting algorithm is explained in the commented Python code for `Q1reducer1.py`.

Table 1 is a diagram specifying the computed number of unique prox-ids on each date across the two datasets. It should be noted that one day, 2016-06-04, has zero unique prox-ids. The number of unique prox-ids detected per day is roughly consistent with the number of offices.

Table 1: (Map Reduce Q1) Number of unique prox-ids by date, according to the combined contents of `prox-fixed.csv` and `prox-mobile.csv`. The 4th-5th and 11th-12th were weekends. The entry for 2016-06-04 was added to the table manually.

DATE	N_UNIQUE_IDS
2016-05-31	114
2016-06-01	113
2016-06-02	115
2016-06-03	114
2016-06-04	0
2016-06-05	1
2016-06-06	115
2016-06-07	115
2016-06-08	115
2016-06-09	116
2016-06-10	114
2016-06-11	2
2016-06-12	1
2016-06-13	115

```
1 cd ~/bd-sp-2017/coursework
```

```
nano Q1mapper1.py
```

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        words = line.strip().split(',')
        if 'timestamp' not in words: # line is not header
                words = [word.strip() for word in words] # strip any \n
                date = words[0][0:10] # remove time from date-time
                id = words[2]
                key = date + ' ' + id
                print key + '\t1' # value of 1
```

```
nano Q1reducer1.py
```

```
#!/usr/bin/env python
import sys

current_date = None
current_id = None
current_unique_ids = 0

print 'DATE N_UNIQUE_IDS' # Header

for line in sys.stdin:
        key, value = line.strip()split('\t')
        date, id = key.split(' ')

        if date != current_date:
                if current_date != None:
                        print current_date + ' ' + str(current_unique_ids)
                current_date = date
                current_id = None
                current_unique_ids = 0

        if id != current_id:
                current_unique_ids += 1
                current_id = id

print current_date + ' ' + str(current_unique_ids)
```

```
hadoop jar $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -libjars $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -input coursework/prox-*.csv \
        -output q1-output \
        -mapper "python Q1mapper1.py" -file Q1mapper1.py \
        -reducer "python Q1reducer1.py" -file Q1reducer1.py

hadoop fs -head q1-output/part* | cat
```

Map Reduce: Question 2

Using the `prox-fixed` dataset, what is the (floor, zone) of the most-visited location in the building?

Assume that the most-visited location corresponds to the (floor, zone) that has the greatest number of prox id readings over the two-week period. The following solution uses two MapReduce procedures so that Hadoop's distributed sorting algorithm can be used to sort (floor, zone) pairs by their number of prox card readings. This makes it applicable to a dataset with a very large number of floors and zones.

A MapReduce procedure for identifying the most-visited location is:

1. Map1 receives a line from `prox-fixed.csv`, parses its `floor` and `zone` values, then writes the key-value pair (`floor zone`, 1).
2. Combine1 receives a key-value pair and sums over the values to create a Mapper-level `n_prox_readings` for each key `floor zone`[1].
3. Reduce1 receives the key-value pairs sorted by key `floor zone` and sums over the values by key to determine each key's `n_prox_readings` across the entire dataset. Reduce1 outputs lines of the format `n_prox_readings,floor,zone`.
4. Map2 receives the output of Reduce 1, pads `n_prox_readings` with zeros so that ShuffleSort effectively sorts by number, then outputs `n_prox_readings floor,zone` (where the space is a tab character).
5. Reduce2 receives the output of Map2 sorted by `n_prox_readings` and outputs `floor,zone,n_prox_readings` for each key-value pair.

Table 2 lists the number of prox card readings over the two-week period by zone. The (floor, zone) of the most-visited location in the building is (2, 1). Referring to the map of Floor 2 provided in the coursework briefing, this area corresponds to an open area linking most of the building's offices, an area that is directly next the main staircase. It seems plausible that this would be the most-visited location in the building: people are likely pass it when entering and leaving Floor 2. The floor's inhabitants are also likely to move through it when visiting one another.

```
cd ~/bd-sp-2017/coursework
nano Q2mapper1.py
```

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        pieces = line.strip().split(', ')

        if 'timestamp' not in pieces:
                floor = pieces[3].strip()
                zone = pieces[4].strip()
                print floor+','+zone+'\t1'
```

```
nano Q2combiner1.py
```

```
#!/usr/bin/env python
import sys

current_fz = None # current floor-zone
n_prox_readings = 0

for line in sys.stdin:
        fz, value = line.split('\t')
        if fz == current_fz:
                n_prox_readings += 1
        else:
                if current_fz != None:
                        print current_fz+'\t'+str(n_prox_readings)
                current_fz = fz
                n_prox_readings = 1

print current_fz+'\t'+str(n_prox_readings)
```

```
nano Q2reducer1.py
```

[1] Using a combiner reduces the amount of data that needs to be transmitted from Mappers to the Reducer.

```
#!/usr/bin/env python
import sys

current_fz = None # current floor-zone
n_prox_readings = 0

for line in sys.stdin:
        fz, value = line.strip().split('\t')
        if fz == current_fz:
                n_prox_readings += int(value)
        else:
                if current_fz != None:
                        print str(n_prox_readings)+','+current_fz
                current_fz = fz
                n_prox_readings = int(value)

print str(n_prox_readings)+','+current_fz
```

```
nano Q2mapper2.py
```

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        n_prox_readings, floor, zone = line.split(',')
        print n_prox_readings.zfill(10) + '\t' + floor + ',' + zone.strip()
```

```
nano Q2reducer2.py
```

```
#!/usr/bin/env python
import sys
print 'FLOOR,ZONE,N_PROX_READINGS'

for line in sys.stdin:
        pieces = line.split('\t')
        print pieces[1].strip()+','+str(int(pieces[0]))
```

```
hadoop jar $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -libjars $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -input coursework/prox-fixed.csv \
        -output q2-output1 \
        -mapper 'python Q2mapper1.py' -file Q2mapper1.py \
        -reducer 'python Q2reducer1.py' -file Q2reducer1.py \
        -combiner 'Q2combiner1.py' -file Q2combiner1.py # NB combiner

hadoop jar $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -libjars $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -input q2-output1/part* \
        -output q2-output2 \
        -mapper 'python Q2mapper2.py' -file Q2mapper2.py \
        -reducer 'python Q2reducer2.py' -file Q2reducer2.py

hadoop fs -cat q2-output2/part* | tail
```

FLOOR	ZONE	N_PROX_READINGS
1	5	1
3	Server Room	29
1	8	41
1	6	50
1	3	60
1	7	91
3	6	221
3	2	239
3	3	310
2	3	343
1	2	801
3	4	971
3	1	1044
2	2	1251
2	6	2296
2	7	3159
1	4	4104
2	4	4231
1	1	4343
2	1	6153

Table 2: (Map Reduce Q2) Number of prox card readings for each (floor, zone) across the two week period.

Map Reduce: Question 3

> *Using both datasets, what is the `prox-id` of the staff member with the greatest number of prox card readings on 2nd June 2016?*

To retrieve the `prox-id` of the staff member with the greatest number of prox card readings on 2nd June 2016, we use the MapReduce procedure:

1. Map1 receives a line from `prox-*.csv` and creates the key-value pair `(prox_id, 1)` if the line's `date` is 2016-06-02, with no pair created otherwise.
2. Reduce1 sums over the values associated with each key to create the count `n_prox_readings`. It outputs `n_prox_readings prox_id` (where the space corresponds to a tab).
3. Map2 reads the file written by Reduce1 and pads each line's `n_prox_readings` with 0s to create `n_prox_readings0`, enabling ShuffleSort to sort the pairs by number of readings, then outputs the key-value pairs `(n_prox_readings0, prox_id)`.
4. Reduce2 receives the sorted entries, strips the leading 0s from `n_prox_readings0`, and outputs a text file with lines of the format `prox_id,n_prox_readings`. We print the head of this file to the shell.

If there were a very large number of employees, it would not be possible to sort them by `n_prox_readings` on one machine. Map2 and Reduce2 exist in response to this issue and allow us to exploit Hadoop's distributed sorting algorithm. It also worth noting that the amount of network traffic could be reduced by incorporating a combiner after Map1 that accumulates the number of readings of a prox-id within each partition. This would mean that less data must be transmitted to the Reduce machine.

The '10' in the code below would need to be scaled based on the maximum number of prox readings per employee per day.

Table 3 presents the output of the above Map Reduce program, truncated to the head and tail values. We can see that id `fresumir001`'s prox card was detected most frequently on 2016-06-02, at 64 detections.

```
cd ~/bd-sp-2017/coursework
nano Q3mapper1.py
```

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        words = line.strip().split(', ')
        if 'timestamp' not in words: # not header
                date = words[0][0:10]
                prox_id = words[2]
                if date[8:10]=='02': # 2016-06-02
                        print prox_id+'\t1'
```

```
nano Q3reducer1.py
```

```
#!/usr/bin/env python
import sys
current_prox_id = None
n_prox_readings = 0

for line in sys.stdin:
        prox_id, value = line.strip().split('\t')
        if prox_id != current_prox_id: # if id changes
                if current_prox_id != None: # print no. of readings for previous id
                        print current_prox_id+','+str(n_prox_readings)
                current_prox_id = prox_id # update current id
                n_prox_readings = 1
        else:
                n_prox_readings += 1
print current_prox_id+','+str(n_prox_readings)
```

```
nano Q3mapper2.py
```

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        prox_id, n_prox_readings = line.strip().split(',')
        n_prox_readings0 = n_prox_readings.zfill(10)
        print n_prox_readings0+'\t'+prox_id
```

```
nano Q3reducer2.py
```

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        n_prox_readings0, prox_id = line.strip().split('\t')
        print prox_id+','+str(int(n_prox_readings0))
```

```
hadoop jar $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -libjars $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -input coursework/prox-*.csv \
        -output q3-output1 \
        -mapper "python Q3mapper1.py" -file Q3mapper1.py \
        -reducer "python Q3reducer1.py" -file Q3reducer1.py

```

PROX_ID	N_PROX_READINGS
jfrost001	3
earpa001	5
rparade001	6
cstaley001	11
dscozzese001	11
⋮	⋮
eminto001	55
mbramar001	55
mvollan001	55
llagos001	59
fresumir001	64

Table 3: (Map Reduce Q3) Number of card readings for each prox-id on the 2nd of June 2016, across both prox-fixed.csv and prox-mobile.csv.

```
hadoop jar $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
       -libjars $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
       -input q3-output1/part* \
       -output q3-output2 \
       -mapper "python Q3mapper2.py" -file Q3mapper2.py \
       -reducer "python Q3reducer2.py" -file Q3reducer2.py

hadoop fs -cat q3-output2/part* | tail
```

Map Reduce: Question 4

> *Using the `bldg-measurements` dataset, produce a time series plot of the average hourly 'Total Electric Demand Power'. What does this plot indicate about power usage throughout the day?*

We calculate the average hourly Total Electric Demand Power (TEDP) by averaging all TEDP readings within a given hour across all days. In `bldg-measurements.csv`, `timestamp` is of the format HH:MM:SS and corresponds to column 1 while `TEDP` is a float and corresponds to column 9. The following MapReduce program is used:

1. Map1 receives a line from `bldg-measurements.txt`, parses `timestamp`'s hour `HH` and the TEDP value `TEDP`, then outputs key-value pairs `(HH, TEDP)`. ShuffleSort sorts the values by `HH`.
2. Reduce1: for each `HH` value, we compute the hourly TEDP average `HHTEDPAvg` using the recursive algorithm described below. The output of Reduce2 is `HH,HHTEDPAvg`.

We compute the TEDP hourly averages using an on-line algorithm. Assume that the average $\bar{x}_n$ of the first n values in the sequence $x_1, x_2, \ldots$ is known so that $\bar{x}_{n+1}$ can be computed via the recursion:

$$\bar{x}_{n+1} = \frac{n}{n+1}\left(\bar{x}_n + \frac{x_{n+1}}{n}\right) \tag{1}$$

By calculating the averages using Equation 1 we avoid having to store multiple TEDP values. This is desirable because the number of TEDP values corresponding to a given hour could be very large, meaning that they could not all be stored on one machine.

Figure 1 contains the requested plot of average total electric power demand versus hour of the day. As might be expected, power usage is low (≈ 0.12MW) between the hours of 18:00 and 06:00, when presumably very few people are at work. Between 08:00 and 17:00 power usage is much higher: the office is being used by the company's employees - lights are on, elevators are running, and the building's HVAC units are being used by taskmasters Chilly Sally on Floor 1 and Sweaty Joe on Floor 2.

```
cd ~/bd-sp-2017/coursework
nano Q4mapper1.py
```

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
        words = line.strip().split(', ')
        if 'Total Electric Demand Power' not in words: # if not header
                HH = words[0][11:13] # HH of YYYY-MM-DD HH:MM:SS
                TEDP = words[8]
                print HH.strip()+'\t'+TEDP.strip()
```

```
nano Q4reducer1.py
```

```
#!/usr/bin/env python
import sys

current_HH = None
current_HHTEDPAvg = 0
n = 0
print 'HH,HHTEDPAvg'

```

```
for line in sys.stdin:
        HH, TEDP = line.split('\t')
        if HH != current_HH: # if key changes
                if current_HH != None: # print the average
                        print current_HH+','+str(current_HHTEDPAvg)
                current_HH = HH # re-initialize current HH,
                n = 1 # counter,
                current_HHTEDPAvg = float(TEDP) # and average
        else: # update average recursively
                current_HHTEDPAvg = n*(current_HHTEDPAvg + (float(TEDP)/n))/(n+1)
                n += 1

print current_HH+','+str(current_HHTEDPAvg)
```

```
hadoop jar $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -libjars $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
        -input coursework/bldg-* \
        -output q4-output \
        -mapper "python Q4mapper1.py" -file Q4mapper1.py \
        -reducer "python Q4reducer1.py" -file Q4reducer1.py

hadoop fs -get q4-output/part* # then scp and plot in R
```

```
# on local shelll (not bazooka)
scp user255@bazooka.ma.ic.ac.uk:/home/user255/bd-sp-2017/coursework/part-0000 \
        averageHourlyTEDP.csv # transfer with WinSCP on Windows

# rename local part-0000 file to averageHourlyTEDP.csv
```

```
# ! R, local, set working directory to download location !
Data <- read.csv('averageHourlyTEDP.csv',
                        col.names=c('HOUR', 'AVERAGE_HOURLY_TEDP'))
Data$AVERAGE_HOURLY_TEDP <- Data$AVERAGE_HOURLY_TEDP/(10^6)
Data <- rbind(Data, c(24, Data$AVERAGE_HOURLY_TEDP[1]))

require(ggplot2)

pdf('001MapReduceQ4.pdf', height=3, width=5)
ggplot(Data, aes(x=HOUR, y=AVERAGE_HOURLY_TEDP)) +
    geom_vline(xintercept=c(8, 17), linetype='dashed', col='gray') +
    geom_line() +
    geom_point() +
    xlab('Hour') +
    ylab('Average Hourly TEDP [MW]') +
    ylim(0, 0.2) +
    scale_x_continuous(breaks=seq(0, 24, 4), limits = c(0, 24))
dev.off()
```

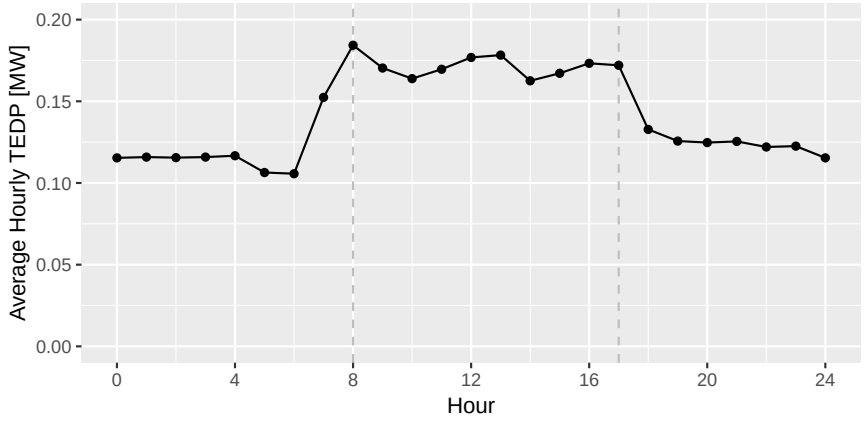

Figure 1: Average hourly total electric demand power (TEDP) based on `bldg-measurements.csv`. The points at HOUR=0 and HOUR=24 share the same value for average TEDP - HOUR=24 was included to emphasize that HOUR=0 follows HOUR=23. The dashed lines are included to highlight the start and end of a typical working day (08:00 to 17:00).

HOUR	14-DAY AVERAGE TEDP [W]
00	115375.296602
01	115155.517601
02	115511.196886
03	114480.489709
04	113902.055684
05	106418.49304
06	105679.310167
07	152401.02978
08	184323.085726
09	170381.002059
10	162903.090721
11	168614.072495
12	176836.336588
13	176149.023414
14	161589.476829
15	166157.204236
16	173261.315624
17	168968.472612
18	130426.649105
19	124922.283107
20	124008.717558
21	123200.905834
22	121309.481237
23	122560.906415

Table 4: (Map Reduce Q4) 14-day average hourly total electric power demand [W]. Computed using `bldg-measurements.csv`.

Spark

Spark: Question 5

> *Parse the prox-fixed.csv data file into an RDD[ProxReading], where ProxReading is defined as*
>
> `case class ProxReading(timeStamp: org.joda.time.DateTime, id: String, floorNum: String, zone: String)`
>
> *In this class,* `timestamp` *corresponds to a joda DateTime object,* `id` *corresponds to prox-id,* `floornum` *corresponds to the floor number,* `zone` *corresponds to the zone id.*

The following Scala commands yield an RDD[ProxReading]. The data is read directly from HDFS as an RDD, meaning that it moves from HDFS's parallel hard-disk replicate-based representation to Spark's parallel RAM-bound replicateless representation.

The final caching step in the code is not necessary. It is included to highlight that caching can be used to store (persist[2]) an RDD in-memory even once further actions are applied to it. This is useful if an RDD is to be re-used regularly or is expensive to recompute in the event of a worker node failure, since the action sequence that creates it does not need to be repeated.

```
import org.joda.time.DateTime
import org.joda.time.format.DateTimeFormat

val proxfixedFile = sc.textFile("hdfs:///user/user255/coursework/prox-fixed.csv")

case class ProxReading(timeStamp: org.joda.time.DateTime,
                       id: String, floorNum: String, zone: String) # case class is immutable

def isHeader(line: String): Boolean = line.contains("timestamp")

def parse(line: String) = {
    val pieces = line.split(", ")
    val formatter = DateTimeFormat.forPattern("yyyy-MM-dd HH:mm:ss")
    val datetime = formatter.parseDateTime(pieces(0))
    val id = pieces(2).replace(" ", "") // remove spaces
    val floornum = pieces(3).replace(" ", "")
    val zone = pieces(4).replace(" ", "")
    ProxReading(datetime, id, floornum, zone)
}

val proxfixedData = proxfixedFile.filter(x => !isHeader(x)).map(parse)

proxfixedData.persist() // proxfixedData.cache()
```

Spark: Question 6

> *Using the prox-fixed dataset, what is the (floor, zone) of the most-visited location in the building across the complete dataset?*

The Scala code below yields a table that is identical to Table 2, as we would hope. The mapping portion of this code takes the RDD constructed in Question 5 and creates an RDD[(String, int)] with key `'floorNum zone'` and value `1`. No distributed sorting occurs until `reduceByKey(_+_)` is called, which sorts the RDD by

[2] A minor difference between `cache()` and `persist()` is that the latter can store an RDD on the hard disk of worker nodes.

key and accumulates each key's values to yield a count of the number of times each floor was visited. The output of this is an RDD[(String, Int)] with as many rows as unique (floor, zone) pairs.

The `collect()` command returns all entries in the RDD to the driver program, which executes on one machine[3]. If the number of unique floor-zones was large, the machine on which the driver program was running might not have enough memory to store this collection and would crash. This may not be desirable. Instead, we could use `.take(n)`, where `n` is a natural number, which would create a collection containing only the first `n` entries of the RDD.

```
val floorZoneCounts = proxfixedData.map(x => (x.floorNum+" "+x.zone, 1)).reduceByKey(_+_)

floorZoneCounts.sortBy(_._2).collect().foreach(println)
```

Spark: Question 7

> *Using both datasets, what is the prox-ID of the staff member with the greatest number of prox card readings on 7th June 2016?*

The code is self-explanatory and is similar to what has been shown so far, with the exception of the `union()` method. This method gathers entries from both `proxmobileID` and `proxfixedID` into a single RDD. Five head entries and five tail entries from the output of the final command are shown in Table 5.

Running similar code that filters for entries from the 2nd June 2016 produces an output that corroborates the result of Question 3 (i.e. Table 3).

```
def is7June(line: String): Boolean = {
  val pieces = line.split(", ")
  val MMdd = pieces(0).substring(5, 10) # e.g. 05-31
  MMdd == "06-07"
}

def parseID(line: String) = {
        val pieces = line.split(", ")
        val id = pieces(2).replace(" ", "")
        (id, 1)
}

val proxmobileFile = sc.textFile("hdfs:///user/user255/coursework/prox-mobile.csv")

val proxmobileID = proxmobileFile.filter(x => !isHeader(x)).filter(is7June).map(parseID)

val proxfixedID = proxfixedFile.filter(x => !isHeader(x)).filter(is7June).map(parseID)

val idCounts = proxmobileID.union(proxfixedID).reduceByKey(_+_)

idCounts.sortBy(_._2).collect().foreach(println)
```

Spark: Question 8

> *Provide a concise summary of your experiences writing Map Reduce programmes and Spark commands for questions 6 and 7. Comment on the differences between the two computational platforms.*

- Spark offered an interactive development environment that made it easy to experiment and identify errors quickly. Development in MapReduce was slower since scripts had to be run via the shell.

[3]When this happens, the RDD is converted to a collection, a local copy of the RDD.

Table 5: Number of prox card readings for each staff id. for the 7th June 2016 across both `prox-fixed.csv` and `prox-mobile.csv`.

ID	PROX CARD READINGS
pyoung001	1
jfrost001	3
earpa001	5
rparade001	6
dscozzese001	11
⋮	⋮
yfinney001	51
sflecha001	56
sfusil001	56
eminto001	57
fresumir001	64

- Scala/Spark was capable of more data processing per line of code than MapReduce, as is shown by the data in Table 7.
- Spark allowed the sequence of data transformations to be graphed, whereas no such functionality exists for MapReduce. Scala's syntax also makes it clear what sequence of transformations is applied to an RDD.
- Spark processed the data more quickly because it relied on in-memory rather than disk-based processing[4].
- Scala is statically typed and emphasizes the use of immutable data types, which arguably provides better guarantees regarding code stability.

Both platforms could read and write to disk via HDFS and offered similar levels of file compatibility. They also provided protocols for fault-tolerance: Spark in the form Resilient Distributed Dataset (RDD), and MapReduce via the JobTracker. Evaluating these innovations in detail is worthwhile.

An RDD consists of partitions of a transformed dataset. These partitions are held in-memory across the cluster's nodes. RDDs can be cached, which reduces the time-to-recovery in the event that a partition of a transformed RDD is lost. By contrast, MapReduce's JobTracker identifies when a worker node has failed to complete a task and reschedules that task to another node. MapReduce cannot write to a RAM cache, meaning that the dataset must be read from the disk again. MapReduce is therefore slower to recover from a task failure than Spark, provided Spark's caching tool is used.

MapReduce has several benefits relative to Spark. Python is more widely known within the data science community[5] than Scala. MapReduce attempts to standardize how data is processed and is surprisingly flexible. If every analysis a characteristic structure, then it is easier to interpret the work of other people. This is a benefit that comes at the expense of constraining the analysis format: MapReduce is probably less suitable for complex data manipulations.

If data does not need to be processed quickly, MapReduce may be cheaper to implement at scale. This is because it processes the data in batches and relies on disk storage, which tends to be cheaper than RAM.

[4] Compute time was not an issue in the questions as the datasets were small. For larger datasets however, Spark's speed advantage would be highly desirable.

[5] Although a Python API for Spark exists!

Table 6: Lines of code per solution for Questions 2, 3, 5, 6, and 7. The Spark solutions required fewer lines of code, even accounting for their dependency (the solution to Question 5). Note that Questions 2 and 6 and Question 3 and 7 request essentially the same output.

Question	Platform	Lines of code
2	Hadoop	31
3	Hadoop	29
6	Spark	2
7	Spark	13
5	Spark	15

Spark: Question 9

> *Construct an RDD from the bldg-measurements.csv data containing the "Date/Time" column (column 1) and the "F_2_Z_1 VAV REHEAT Damper Position" (column 193).*
>
> *What is the date and time of the first occurrence of the F_2_Z_1 VAV REHEAT Damper Position being set to its maximum value of 1.0?*

The final command in the code below prints `damperClass(2016-06-02 15:20:00,1.0)`, indicating that the first time the reheat damper position attains its maximum value is on the 2nd of June 2016 at 15:20:00. The data is sorted after filtering to reduce the amount of sorting that needs to be done.

```
val bldgFile = sc.textFile("hdfs:///user/user255/coursework/bldg-measurements.csv")

case class damperClass(datetime: String, position: Float)

def isHeader(line: String): Boolean = line.contains("timestamp")

def parse(line: String) = {
        val pieces = line.split(", ")
        val datetime = pieces(0)
        val damperPos = pieces(192)
        damperClass(datetime, damperPos.toFloat)
}

val damperData = bldgFile.filter(x => !isHeader(x)).map(parse)

damperData.filter(x => x.position==1).sortBy(_.datetime).take(1).foreach(println)
```

Spark: Question 10

> *A rogue employee is believed to be increasing the Hazium concentration in the building by modifying the Reheat Damper position. By using the Spark package MLlib or other Spark command sequence, demonstrate a statistical association between the Hazium concentration (from f2z2-haz.csv) and the 'F_2_Z_1 VAV REHEAT Damper Position' variable. Provide a concise summary of your statistical findings, using diagrams where appropriate.*

The key theme of this section is that common summary statistics, while useful when exploring Big Data, may not reveal unique structures within a dataset that are relevant to the problem at hand.

Initially we consider the relationship between Hazium concentration and the position of Floor 2 Zone 1's reheat damper. The damper-Hazium concentration Pearson correlation value, 0.178, does not provide com-

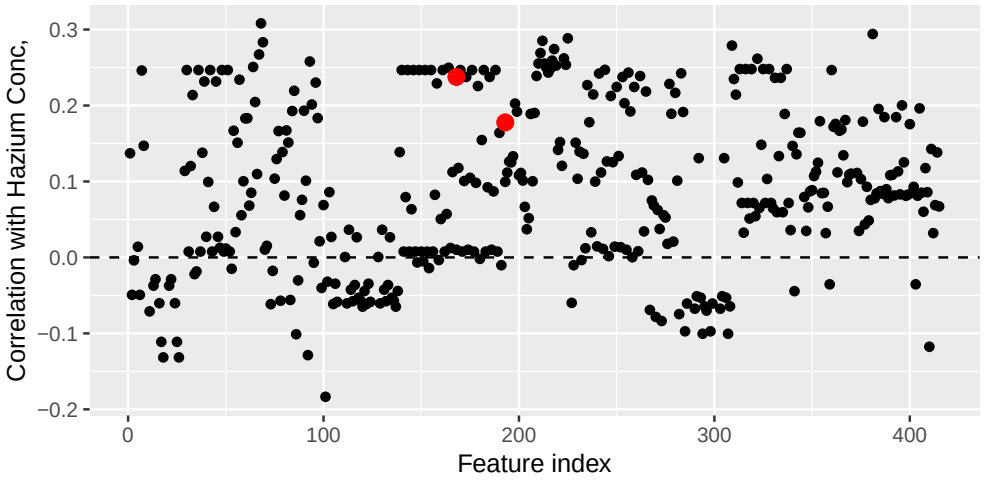

Figure 2: Pearson correlations for Hazium concentration in Floor 2 Zone 2 and the 416 features in the `bldg-measurements` dataset. The left red point corresponds to the Floor 2 Zone 2 thermostat heating setpoint. The right red point corresponds to the Floor 2 Zone 1 reheat damper position.

pelling evidence for a linear zero-lag relationship between Hazium concentration and damper position. This being said, Figure 2, a plot of the Pearson correlation between Hazium concentration and all building measurements, indicates that certain sets of measurements had significantly non-zero correlations with Hazium concentration. Specifically, the largest correlation values are for quantities related to the power supplied to the building's HVAC system (e.g. reheat coil power, heating setpoints and so on), which were all ≈ 0.2.

The time series for the reheat damper position for Floor 2 Zone 1 and heating setpoint for Floor 2 Zone 2, along with that of the Hazium concentration, are plotted in Figures 3 and 4[6]. For Tuesday through Saturday of the first week, the heating is set to a comfortable 21°C during the day and is switched off overnight, as shown by the daily peaks in both HVAC time series. On the Friday of the second week however, the heating setpoint is changed to 24°C and remains at this setting over the weekend. This is reflected by the reheat damper position, which is at its maximum for the duration of the weekend. Furthermore, the time series plots for other HVAC measurements, such as reheat coil powers, indicate that the system worked considerably harder on this weekend than at any other point over the two weeks. It is when the HVAC system abruptly enters this high-power mode that the Hazium concentration skyrockets to dangerous (possibly even fatal) levels. If we assume that the HVAC system is controlled from the Server Room, then we must ask: who in the Server Room on that Friday was unusual?

Filtering the dataset reveals that staff with prox-ids `clais001`, `sflecha001`, `csolos001`, `lbennett001`, and `pyoung001` visit the Server Room 6, 7, 6, 5, and 4 times respectively over the two week period. Most of their visits precede typical HVAC activity. All of these actors, with the exception of `clais001`, were in the Server Room on the Friday of the 10th. Staff member `ncalixto001`, however, makes their first and only visit to the Server Room at 14:03:25 on Friday 06-10, immediately before the heating in Floor 2 Zone 2 is set to 24°C for the entire weekend. If, as under our conjecture, this causes the Hazium concentration to rise, then it seems likely that `ncalixto001` is connected with the attack, which means that they would constitute the factor linking the reheat damper's position with Hazium concentration.

The analysis in this question was supported by using MLlib to compute a 416×416 correlation matrix and Spark to filter the dataset for people visiting the Server Room.

```
1 // Hazium concentration analysis (Question 10)
```

[6]The time-series plots can be created for a Big data set by either subsampling the data or by constructing the plot sequentially. Subsampling in Spark can be done using the RDD methods `sample` or `takeSample`.

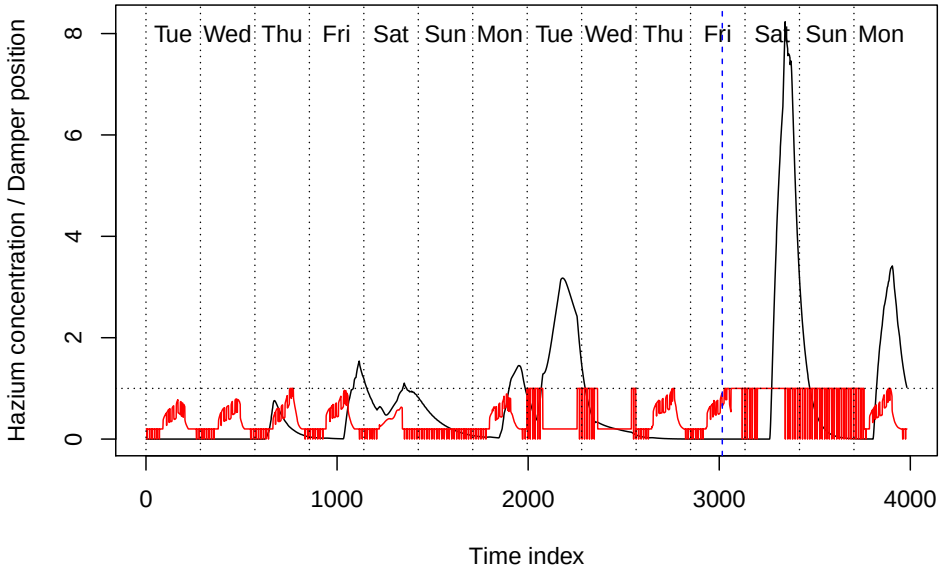

Figure 3: Time series for Hazium concentration (black) and Floor 2 Zone 1's reheat damper position (red). The time of the final visit the Server Room on Friday the 10th of June is indicated by the dashed blue line. The horizontal dashed line denotes the reheat damper's maximum position.

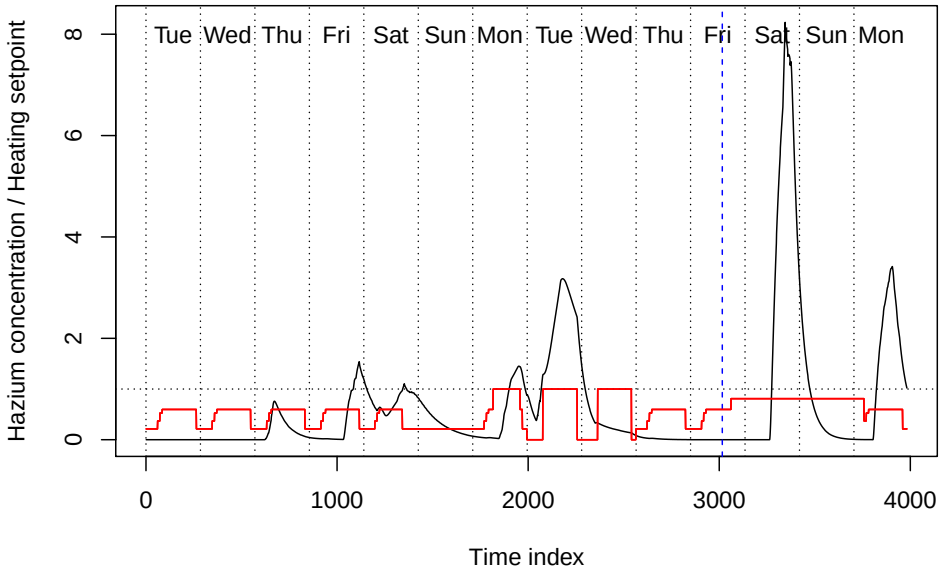

Figure 4: Time series for Floor 2 Zone 2 thermostat heating setpoint (red), translated to have zero minimum (12.6°C) and a range of 1 (such that 1 on the y-axis corresponds to 26.7°C), against Hazium concentration (black). The maximum heating setpoint value is indicated by the horizontal dashed line.

```
import org.apache.spark.mllib.linalg._

// Load and parse datasets
val bldgFile = sc.textFile("hdfs:///user/user255/coursework/bldg-measurements.csv")
val hazFile = sc.textFile("hdfs:///user/user255/coursework/f2z2-haz.csv")

def isNotHeader(line: String): Boolean = !line.contains("timestamp")

def parseHaz(line: String) = {
 val pieces = line.split(", ")
 val timestamp = pieces(0)
 val conc = pieces(1).toDouble
 (timestamp, conc)
}

def parseBldg(line: String) = {
 val pieces = line.split(", ")
 val timestamp = pieces(0)
 val meas = pieces.drop(1).map(x => x.toDouble)
 (timestamp, meas)
}

def parseProx(line: String) = {
 val pieces = line.split(", ")
 val timestamp = pieces(0)
 val id = pieces(2)
 val floor = pieces(3)
 val zone = pieces(4)
 (timestamp, id, floor, zone)
}

def getHeader(line: String) = line.split(", ").drop(1) // drop 'timestamp' only

val bldgHeader = bldgFile.filter(x => !isNotHeader(x)).flatMap(getHeader).toArray
val bldgData = bldgFile.filter(isNotHeader).map(parseBldg)
val hazData = hazFile.filter(isNotHeader).map(parseHaz)

bldgData.persist() // for example
hazData.persist()

// Retain only values that are shared between the data sets
val hazbldgData = hazData.join(bldgData).map(x => Vectors.dense(Array(x._2._1)++x._2._2)) //RDD[Vector]


// Compute and export Hazium - Building measurement correlations
import org.apache.spark.mllib.stat.Statistics

val correlMatrix = Statistics.corr(hazbldgData, "pearson")
val N = correlMatrix.numRows
val hazCorrel = correlMatrix.toArray.slice(1, N) // Hazium correlations
val correlHeader = "F_2_Z_2 Hazium Concentration" +: bldgHeader // prepend to Array

// Write hazCorrel to text file
val headerString = correlHeader.foldLeft("")((a, b) => a + "," + b)
val hazCorrelString = hazCorrel.foldLeft("")((a, b) => a + "," + b.toString)

import java.io._
val pw = new PrintWriter(new File("hazCorrel.csv"))
pw.write(headerString)
pw.write(hazCorrelString)
pw.close()


// Find out who visited the Server Room over the two weeks
val proxFile = sc.textFile("hdfs:///user/user255/coursework/prox-fixed.csv")
val serverRoomData = proxFile.filter(isNotHeader).map(parseProx).filter(x => x._4=="Server Room")

serverRoomData.collect().foreach(println) // All visits to Server Room
```

```
70
71 serverRoomData.map(x => (x._2, 1)).reduceByKey(_+_).collect().foreach(println) // Serv. Room visit counts
```

Spark: Question 11

> *By using the (fixed) proximity location data, determine the IDs of the employees that were in the Server Room prior to the sudden increase in Hazium at the end of the dataset (on the 10th June 2016).*

The Spark output for the employees that were in the Server Room on the 10th of June is presented below.

```
ProxReading(2016-06-10T10:21:57.000+01:00,pyoung001,3,ServerRoom)
ProxReading(2016-06-10T13:58:51.000+01:00,sflecha001,3,ServerRoom)
ProxReading(2016-06-10T14:03:31.000+01:00,ncalixto001,3,ServerRoom)
ProxReading(2016-06-10T14:04:25.000+01:00,lbennett001,3,ServerRoom)
ProxReading(2016-06-10T14:04:25.000+01:00,csolos001,3,ServerRoom)
```

```
1  def isHeader(line: String): Boolean = line.contains("timestamp")
2
3  def parse(line: String) = {
4                          val pieces = line.split(",")
5                          val formatter = DateTimeFormat.forPattern("yyyy-MM-dd HH:mm:ss")
6                          val datetime = formatter.parseDateTime(pieces(0))
7                          val id = pieces(2).replace(" ", "")
8                          val floornum = pieces(3).replace(" ","")
9                          val zone = pieces(4).replace(" ", "")
10                         ProxReading(datetime,id,floornum,zone)
11 }
12
13 val proxfixedData = proxfixedFile.filter(x => !isHeader(x)).map(parse)
14
15 // Employees in Server Room on Friday before
16 // dangerous Hazium increase at the weekend
17 proxfixedData.filter(x => x.zone=="ServerRoom")
18                         .filter(x => (x.timeStamp.toString("MM-dd") == "06-10"))
19                                 .collect().foreach(println)
20
21 val hazFile = sc.textFile("hdfs:///user/user255/coursework/f2z2-haz.csv")
22
23 case class hazEntry(timestamp: String, conc: Double)
24
25 def parseHaz(line: String) = {
26                         val pieces = line.split(", ")
27                         val timestamp = pieces(0)
28                         val conc = pieces(1).toDouble
29                         hazEntry(timestamp, conc)
30 }
31
32 // Time at which the Hazium concentration becomes dangerous
33 hazFile.filter(x => !isHeader(x)).map(parseHaz).filter(x => (x.conc > 6)).take(1).foreach(println)
```

General

General: Question 12

> *Write a question, that could appear in next year's coursework, which tests a student's understanding of the opportunities and problems with using Big Data technology.*

The following checklist identifies what a student should understand based on the content of this course:

- how to use Spark for basic data manipulation and statistical processing (especially using MLlib),
- how to use Hadoop MapReduce and Distributed File System,
- how Hadoop and Spark differ in the way they store and operate on data,
- what the difficulties of identifying meaningful properties of large datasets are (e.g. important relationships, specific joint distribution structures, missing entries in a time series, incomplete cases),
- the requirements of a Big Data system (fault tolerance, linear scalability, ease of use),
- how statistical procedures can be adapted for Big Data (distributed/sequential computation),
- the common operations applied to Big Data (searching, joining, sorting).

It is late January 2017 and find yourself employed by the Greater London Authority as a data scientist. Your mission: to describe and predict crime in London using the contents of the dataset `LondonCrime.csv`[7]. The features of this dataset are specified below. The data consists of monthly reported crime counts for each of London's boroughs between January 2015 and January 2017 and contains 804 996 row entries.

- **Row ID** - Unique row identifier for row entry.
- **Month** - Month and year of reported count, with the format 'YYYYMM' such that '201701' refers to January 2017 (25 unique values).
- **LSOA Code** - The Office for National Statistics' Lower Layer Super Output Area (LSOA) code, referring to a UK geographical area containing approximately 1500 people. These areas were created to be as socially homogeneous as possible based on tenure of household and dwelling type (4835 unique values).
- **Borough** - Name of London borough (33 unique values).
- **Major Category** - High-level crime category (e.g. Robbery) (9 unique values).
- **Minor Category** - Low-level crime category (e.g. Personal Property) (34 unique values).
- **Count** - Monthly reported count of crime.

This dataset was recovered from a dusty old hard drive and there are concerns that it may contain duplicate entries or incomplete cases.

(i) Describe the benefits of storing the dataset using Hadoop's Distributed File System relative to storing it on a hard disk. Explain the concept of linear scalability.

(ii) Determine whether the `Count` column of LondonCrimeClean.csv contains any non-numeric values by writing a MapReduce procedure that outputs by borough the number of entries in the column that are non-numeric.

This question requires a mapper that returns (Borough, IS_NUMERIC), where IS_NUMERIC indicates whether the count of the input line is numeric, and a reducer that sums over each key's value.

[7] This dataset is available at https://files.datapress.com/london/dataset/recorded-crime-summary-data-london-lsoa-level/2017-01-26T18:50:00/MPS_LSOA_Level_Crime_Current.csv

(iii) Use Spark to identify and remove duplicate rows from the dataset, where a row is classed as duplicated if its Month, LSOA Code, and Minor Category values appear more than once.

Your boss has told you that he expects your report by tomorrow morning, precluding you from grinding through the $\binom{804\,996}{2} \approx 3.24 \times 10^{11}$ pairwise comparisons that seem to be necessary to identify the duplicates. It looks like you will have to find a different approach! Spark's `filter()` method may be useful.

Write the resulting dataset to HDFS under the filename 'LondonCrimeClean.csv'. Explain why Hadoop MapReduce might be inappropriate for the task you executed in this question.

This question asks the student to realize that they can use filtering to identify duplicate entries efficiently - filter on Month and LSOA Code, then compare the Minor Category values against one another to check if there is a value is repeated. If a value is repeated, then the dataset must contain a duplicate. Iterate over all Month-LSOA Code combinations to check the entire dataset for duplicates. This approach requires only $25 \times 4835 \times 34 \approx 68 \times 10^6$ comparison operations (i.e. it provides a 5000 times speedup over pairwise comparison)[8].

(iv) Write a MapReduce procedure that identifies the LSOA code that had the greatest number of reported crimes over the 25-month period.

Your solution should be written as though there were arbitrarily many LSOA codes[9] and should make use of Hadoop's combiner functionality. Explain how the combiner affects the procedure's efficiency.

Map the dataset to (Minor Category, Count), then use combiners to sum over the the counts to output (Minor Category, CombinedCount). The reducer then sums over the CombinedCount values for each value of Minor Category before writing a file of the format MinorCategoryCount\tMinorCategory. This file is then passed through an identity mapper to exploit Hadoop's distributed ShuffleSort procedure. An identity reducer can be used. The combiner reduces the volume of data that must be transmitted between Mappers and Reducer, making the procedure more efficient.

(v) You have been asked to investigate whether bicycle theft was correlated with assault with a weapon. You decide to take the borough of Southwark as a case study.

Using Spark and only the entries associated with Southwark, create an RDD with columns (Month, LSOA Code, Count_Bicycle_Theft, Count_Assault_With_Weapon).

For each LSOA code, Count_Bicycle_Theft is the Count corresponding to Minor Category 'Theft/Taking Of Pedal Cycle' and Count_Assault_With_Weapon is the Count corresponding to Minor Category 'Common Assault'. You should use Spark's `join()` method in a way that only retains LSOA codes for which both Count values are non-zero.

Describe and implement an algorithm for distributed computation of Pearson's correlation coefficient in Spark. Use this algorithm to compute the correlation between Count_Bicycle_Theft and Count_Assault_With_Weapon.

This tests the student's ability to transform the dataset from long format into a wide format by constructing two datasets then performing an inner join on them. It also evaluates their depth of understanding with regards to how Spark can be used to efficiently perform distributed computation.

(vi) Sadiq Khan has requested forecasts for the number of crimes for the Major Category 'Violence Against the Person' for the forthcoming year. To do this, you plan to regress the total crime counts per month for each Major Category for the January 2017 ('201701') onto the crime counts of the six preceding months, for every possible contiguous set of six months in the dataset.

Use Spark to create an appropriate design matrix, then use MLlib to fit a linear model via `LinearRegression()`. Estimate the mean-square error of this model using a train-test split procedure. Apply the model to the

[8] A probabilistic approach to this problem could also be taken by giving each row a key `Month-LSOACode-MinorCategory` then repeatedly shuffling, splitting, and joining the data on key. Rows retained by the join would be duplicates.

[9] This is necessary so that your code is scalable to crime datasets for the whole of England and Wales, for which there are about 35 000 LSOA codes.

feature vector containing crime counts between August 2016 - January 2017 to predict the London-wide Violence Against the Person count for February 2017. These tasks require MLlib's `Vector` and `Matrix` data types to be used.

General: Question 13

Identify, download, and perform a statistical analysis of any suitable data that is available on the internet, and write a one-page summary of your findings. Your analysis should use Hadoop, Spark, or both tools. Please note that the data need not be 'big' - the question is intended to assess your approach to the analysis, and how you utilise Big Data technology in performing a statistical analysis.

For this question I selected the dataset `tips.csv`, which is available as part of the `reshape2` package in R. An entry from this data set is shown in Table 7.

We would like to determine whether the tip ratio, tip/total_bill, is positively correlated with an indicator for day=Sun.

Let $x_1, \ldots, x_N$ denote the tip ratio values and let $y_1, \ldots, y_N$ denote the Sunday indicator for each tip, with $N = 240$. These values are taken as realizations of random variables $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ such that the pairs (X_i, Y_i) are independent and identically distributed. Let ρ denote the sample correlation estimator and set $\rho^* = \rho(x_1, y_1, \ldots, x_N, y_N)$ to be the value of this estimator for the sample, where the order of the arguments matters. Consider the distribution of $\rho(x_1, y_{J_1}, \ldots, x_N, y_{J_N})$, where $(J_1, \ldots J_N)$ is a sample from a discrete uniform distribution over all permutations of the vector $(1, \ldots, N)$. This is the distribution of ρ when all pairs (x_i, y_{J_i}) are equally likely (i.e. when the Y_i are independent of the X_i and $\mathrm{E}[\rho(x_1, y_{J_1}, \ldots, x_N, y_{J_N})] = 0$). Denote it by F_N. By comparing ρ^* with F_N, it is possible to evaluate the plausibility that X_i and Y_i are not independent.

On our dataset, $\rho^* = 0.0672$. An approximation to the distribution of ρ under the null hypothesis is shown by the black curve in Figure 5. This curve is a Monte Carlo approximation $\hat{F}_n$ of the true permutation distribution F_n. $\hat{F}_n$ is obtained by permuting the dataset many times, whereas F_N is the empirical distribution generated by all $N!$ permutations. The p-value of the observation under the null hypothesis, $P(\rho \geq 0.0672)$, is 0.159, meaning that the null hypothesis is retained at the 5% level: there is insufficient evidence to conclude that X_i and Y_i are not independent.

In a distributed context permutation across the entire dataset is not possible. Instead, each machine can permute only the values in its partition. To investigate whether an ecdf constructed using restricted permutations of n values across N/n partitions, $\hat{F}_n$, is a good estimator for F_N, a simulation study was run in R. The dataset was shuffled then split into partitions containing `n_in_partition` values each. To sample ρ, values in each partition were permuted then reduced to the sample correlation coefficient via an algorithm described on the following page. These statistics were aggregated across all (simulated) partitions for a value of ρ[10].

The outcome of the simulation study is shown in Figure 5[11]. We can see that partitions containing very few values (e.g. $n = 2$ or $n = 4$) yield an $\hat{F}_n$ that is a poor approximation of F_N, assuming that $\hat{F}_N$ is a good

[10] This is possible because the correlation coefficient consists of sums.

[11] It is worth noting that this experiment was executed for a partition on only one order of the data set, which also affects $\hat{F}_n$ as an estimator of $\hat{F}_N$. Future studies should assess the variance of $\hat{F}_n$ across different partitions of the dataset.

ID	total_bill	tip	sex	smoker	day	time	size
136	8.51	1.25	Female	No	Thur	Lunch	2

Table 7: An entry from the `tips` dataset.

approximation of F_N. although using even these $\hat{F}_n$ for our hypothesis test would not have affected the test's outcome. For $n \geq 80$, the approximation is acceptable for practical purposes.

Distributed computation of the correlation coefficient proceeds as follows. We have a dataset consisting of N pairs $(x_i, y_i) : i = 1, \ldots, N$. Let $I_{(j)}$ denote the index set associated with the partition $j : j = 1, \ldots, m$, where the elements of $I_{(j)}$ are sampled without replacement from $\{1, \ldots, N\} \setminus I_{(1)}, \ldots I_{(j-1)}$.

1. Shuffle and split the pairs across m partitions according to the $I_{(j)}$.
2. On each partition, compute local sums $x_{(j)} = \sum_{i \in I_j} x_i$, $y_{(j)} = \sum_{i \in I_j} y_i$.
3. Transfer sums $x_{(j)}, y_{(j)}$ to driver program.
4. On the driver program, compute the global means $\bar{x} = \sum_{j=1}^{m} \frac{x_{(j)}}{N}$ and $\bar{y} = \sum_{j=1}^{m} \frac{y_{(j)}}{N}$. N is contained in the sums to mitigate against numerical overflow.
5. Broadcast the global means from the driver program to each worker node.
6. For each partition, compute the local sums-of-squares and sum-of-cross-products

$$S^X_{(j)} = \sum_{i \in I_j} (x_i - \bar{x})^2 \tag{2}$$

$$S^Y_{(j)} = \sum_{i \in I_j} (y_i - \bar{y})^2 \tag{3}$$

$$S^{XY}_{(j)} = \sum_{i \in I_j} (x_i - \bar{x})(y_i - \bar{y}) \tag{4}$$

7. Transfer all $S^X_{(j)}, S^Y_{(j)}, S^{XY}_{(j)}$ to the driver program.
8. Compute the global correlation coefficient according to

$$\rho = \frac{\sum_{j=1}^{m} \frac{1}{N} S^{XY}_{(j)}}{\sqrt{\sum_{j=1}^{m} \frac{1}{N} S^X_{(j)}} \sqrt{\sum_{j=1}^{m} \frac{1}{N} S^Y_{(j)}}} \tag{5}$$

   where again the N is included in the sum as a defensive measure against numerical overflow.

For the distributed Monte Carlo permutation test, computation of $\bar{x}, \bar{y}, \sqrt{\sum_{j=1}^{m} S^X_{(j)}}$, and $\sqrt{\sum_{j=1}^{m} S^Y_{(j)}}$ only need occur once. The $S^{XY}_{(j)}$ are affected by partition-level permutation and must be recomputed each time ρ

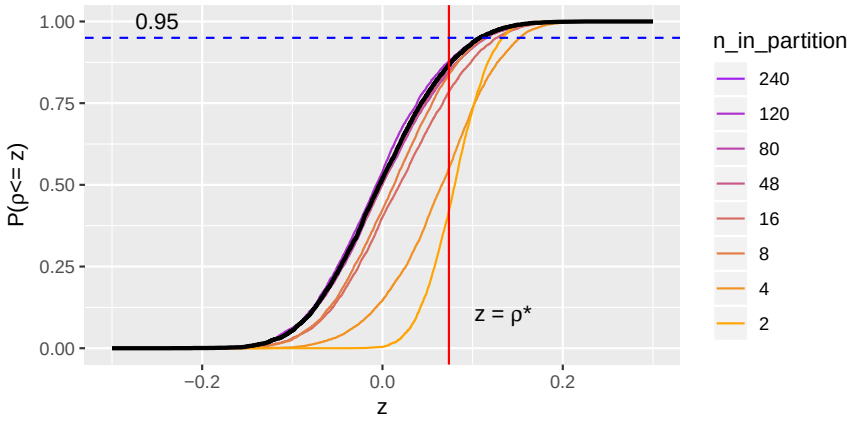

Figure 5: Ecdf of ρ computed over 5e3 Monte Carlo permutations, compared with 95th percentile and observed ρ^*. Comparison with permutation ecdf under partitioning.

is sampled. The full distributed permutation algorithm was not implemented due to time constraints, however distributed computation of the global correlation coefficient was and is presented in the listing below.

Code for computing the tip ratio using Hadoop MapReduce and implementing the distributed correlation coefficient in Spark is provided below.

```
nano q13mapper.py
```

```
#!/usr/bin/env python
# Compute tip ratio and shuffle the data
import sys
import random

for line in sys.stdin:
        if "total_bill" not in line:
                pieces = line.strip().split(",")
                total_bill = pieces[1]
                tip = pieces[2]
                day = pieces[5]
                if day=="Sun":
                        day = 1
                else:
                        day = 0
                tip_ratio = float(tip)/float(total_bill)
                print str(random.randint(1, 2147483647)).zfill(10)+"\t"+str(tip_ratio)+","+str(day)

# randint is so that the data arrives in Spark pre-shuffled
```

```
nano Q13reducer.py
```

```
#!/usr/bin/env python
import sys

print "TIPRATIO,SUNDAY"

for line in sys.stdin:
        print line
        pieces = line.split("\t")
        print pieces[0].strip()+","+pieces[1].strip()
```

```
hadoop jar $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
    -libjars $HADOOP_STR/hadoop-0.20.2-dev-streaming.jar \
    -input coursework/tips.csv\
    -output q13-output \
    -mapper "python Q13mapper.py" -file Q13mapper.py \
    -reducer "python Q13reducer.py" -file Q13reducer.py

hadoop fs -mv q13-output/part* coursework/tipsMunged.csv # needs to be modified if multiple part files
```

```
nano Q13CorrelationCoefficient.scala
```

```
// Load the data set
val tipsFile = sc.textFile("hdfs:///user/user255/coursework/tipsMunged.csv")

def isHeader(line: String): Boolean = line.contains("TIPRATIO")

def parseTips(line: String) = {
        val pieces = line.split(",")
        (pieces(0).toDouble, pieces(1).toDouble)
}
```

```
10
11 val N = tipsFile.count() - 1 // Number of rows
12
13 // Partition the dataset and cache the partitions (full shuffle occurs)
14 val tipData = tipsFile.filter(x => !isHeader(x)).map(parseTips).repartition(2).persist()
15
16 // Compute local sum of x, local sum of y on worker nodes
17 val partxySums = tipData.mapPartitions(values => Iterator(values.foldLeft((0.0, 0.0)){
18                                                 (acc, xy) => (acc._1 + xy._1, acc._2 + xy._2) }))
19
20 // Compute global means using local sums
21 val xBaryBar = partxySums.collect().foldLeft((0.0, 0.0)){
22                             (acc, xysums) => (acc._1 + (xysums._1/N), acc._2 + (xysums._2/N)) }
23 // NB /N in foldLeft is to avoid numerical overflow of large sum
24 // N is only used by the driver program (no need to broadcast)
25
26 // Cache the global means on worker nodes
27 val xbarybar = sc.broadcast((xBaryBar._1, xBaryBar._2)) // broadcast edition of xBaryBar
28
29 // Compute local sums-of-squares and sums-of-cross-products on worker nodes
30 val partxycSquares = tipData.mapPartitions(values => Iterator(values.foldLeft((0.0, 0.0, 0.0)){
31                     (acc, xy) => (acc._1 + math.pow(xy._1 - xbarybar.value._1, 2)/N,
32                                   acc._2 + math.pow(xy._2 - xbarybar.value._2, 2)/N,
33                                   acc._3 + (xy._1 - xbarybar.value._1)*(xy._2 - xbarybar.value._2)/N) }))
34
35 // Compute global sums-of-squares by aggregating partition values on driver program
36 val xycSquares = partxycSquares.collect().foldLeft((0.0, 0.0, 0.0)){
37                                   (acc, xyc) => (acc._1 + xyc._1, acc._2 + xyc._2, acc._3 + xyc._3) }
38
39 // Compute global correlation coefficient on driver program
40 val global_rho = xycSquares._3/(math.sqrt(xycSquares._1)*math.sqrt(xycSquares._2))
```

General: Question 14 (a)

> Write a short (less than one side of A4) synopsis of the paper 'Satistical Paradises and Paradoxes in Big Data (I): Law of Large Populations, Big Data Paradox, and the 2016 US Presidential Election', extracting the key statistical contributions of the paper.

The paper is motivated by the question: 'Is an 80% non-random sample 'better' than a 5% random sample in measurable terms?' Its major statistical contributions towards answering this question are as follows. First, it draws attention to the fact that in non-probabilistic samples, inference for the population mean of some quantity G, denoted $\bar{G}_N$, depends on the population size N and the response mechanism R (such that R_i indicates whether G_i was sampled). Second, it suggests scoring datasets (samples) using the mean-squared error of the sample average $\bar{G}_n$ as an estimator for the population average $\bar{G}_N$. The paper highlights that this is an attractive choice since it is possible to decompose the MSE of the sample mean into the product of three indices, each of which have meaningful interpretations:

$$\mathrm{E}[(\bar{G}_n - \bar{G}_N)^2] = \underbrace{\mathrm{E}_R[\rho^2_{R,G}]}_{D_I} \times \underbrace{\frac{1-f}{f}}_{D_O} \times \underbrace{\sigma^2_G}_{D_U} \tag{6}$$

The expectation is taken with respect to R, the response mechanism[12]. D_I denotes the Defect Index, D_O is the Dropout Odds for response ($f = n/N$ is the fraction of the population surveyed, assumed to be known and non-random), and D_U is the Degree of Uncertainty (i.e. the population variance of G). D_I, D_0, and D_U can be thought of as measures of data quality, data quantity, and problem difficulty respectively. Equation 6 describes how these factors interact and how adjusting them affects the accuracy with which $\bar{G}_N$ can be estimated. This is useful to know when designing mechanisms for constructing Big Data datasets.

D_I cannot be estimated from the sample, since it requires knowledge of non-responses, which we do not have access to by definition. If the actual error of an estimate $\bar{G}_n - \bar{G}_N$ is known, however (such as post-election), then $\rho_{R,G}$ can be estimated and used to inform future analyses, allowing their sensitivity with respect to $\rho_{R,G}$ to be assessed.

A crucial result in the paper is that the ratio between the stochastic error under the actual sampling mechanism and a simple random sampling mechanism grows with $\sqrt{N}$, with constant of proportionality $\rho_{R,G}$:

$$\frac{\bar{G}_n - \bar{G}_N}{\sqrt{\mathrm{Var}_{SRS}(\bar{G}_n)}} = \sqrt{N-1}\rho_{R,G} \tag{7}$$

i.e. the stochastic error $\bar{G}_n - \bar{G}_N$ grows with population size. Meng christens this relationship the Law of Large Populations (LLP): small defects $\rho_{R,G}$ affect the accuracy of $\bar{G}_n$ more aggressively for larger populations, with the error growing at a rate proportional to $\sqrt{N}$. This means that for very large populations, we should be especially conscious of whether it is likely that $\mathrm{E}_R[\rho^2_{R,G}] \neq 0$. Under probabilistic sampling, for which the MSE decreases with n^{-1}, the Defect Index is controlled such that $D_i = O(N^{-1/2})$[13], a useful result in light of the LLP.

The paper introduces a tool that enables its motivating question to be answered: an effective sample size based on equating the MSE of $\bar{G}_n$ under the biased response mechanism with the MSE of $\bar{G}_{n_{eff}}$ under simple random sampling. This equation is used to derive a (very close) upper bound for n_{eff}:

$$n_{eff} \leq \frac{n}{1-f}\frac{1}{ND_I} = \frac{1}{D_O D_I} \tag{8}$$

This bound demonstrates that the effective sample size shrinks dramatically with increasing $ND_I = N\,\mathrm{E}_R[\rho^2_{R,G}]$, meaning that small biases in the sampling mechanism become increasingly catastrophic as population size

[12] This may be deterministic or there may be a distribution over possible $R = (R_1, \ldots, R_N)$. Meng uses a neat trick involving random indices to permit both probabilistic and deterministic response mechanisms to be discussed within a single framework.

[13] i.e. $\lim_{N\to\infty} |D_i||N| < \infty$

increases, as indicated by the LLP. Note that the MSE remains high not because of the variance of $\bar{G}_n$, but because of its bias on account of $\rho_{R,G}$.

The key contribution of this paper is to draw attention to the fact that many Big Data datasets are not probabilistic. This has quantifiable consequences when estimating population statistics: small correlations between response and what is measured produce inaccuracies that are affected by both sample size and population size.

General: Question 14

> Discuss how the key points raised in the paper could be relevant (if at all) to the statistical analysis performed in Question 13, linking to other research if and where appropriate

The key points raised in the paper are relevant to our analysis of tip ratio values because the dataset may be constructed using self-reported visits to the diner.

Take our population to be all visits to the diner that occur over the survey period. Our sample is the entries in the `tips.csv` dataset. The quantity of interest G_i is a visit's tip ratio value and the response variable R_i indicates whether a visit appears in `tips.csv`. Assume that R_i is a fixed value for each visit. A sample estimate $\bar{G}_n$ of $\bar{G}_N$, the population mean tip ratio value, may be biased because people that preferred to tip more on Sundays were particularly keen to volunteer the details of their visits. They would therefore be over-represented in `tips.csv` and $\rho_{R,G}$ would be positive.

$\rho_{R,G}$ cannot be estimated using `tips.csv`, but we can reason about how it will affect $\bar{G}_n$ as an estimator of $\bar{G}_N$. Referring to Equation 6, we can see that the MSE of $\bar{G}_n$ depends on $f = \frac{n}{N}$ and σ_G^2. Assume that σ_G^2 can be estimated without bias from `tips.csv`. This is a generous assumption, since response may also be correlated with deviation from the population mean. With this quantity in hand, however, we can generate a plot of $\mathrm{E}[(\bar{G}_n - \bar{G}_N)^2]$ as a function of the Defect Index $\mathrm{E}_R[\rho_{R,G}^2]$ and dropout odds D_o. This plot is shown in Figure 6 and gives us a feel for the plausible values of $\bar{G}_n$'s MSE. If a range of plausible N can be obtained (for example, by speaking to the diner's manager), then we can focus our attention on the relevant interval of Dropout Odds.

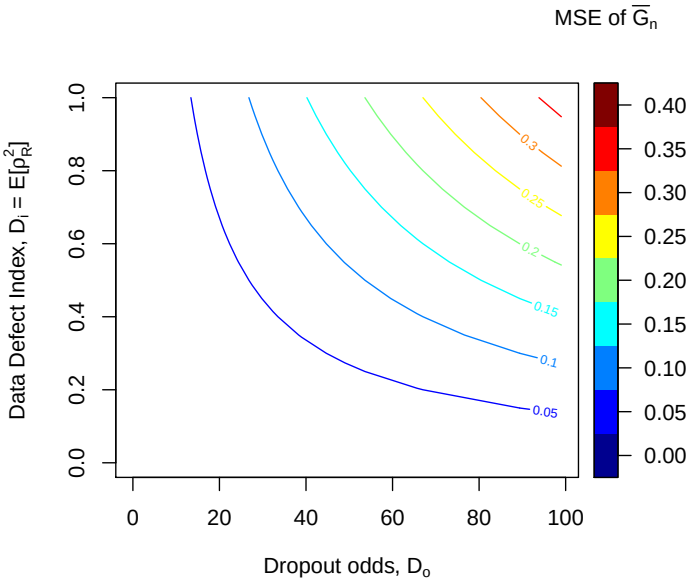

Figure 6: Mean squared error $\mathrm{E}[(\bar{G}_n - \bar{G}_N)^2]$ as a function of possible dropout odds (which depends on the population size relative to $n = 240$) and data defect indices (which are in $[0, 1]$).

We can see that for small values of the MSE (e.g. 0.05), the rate of change of Data Defect index increases as the Dropout Odds approach zero (corresponding to $N = n$). This corroborates what is claimed in the paper: when sampling almost the entire population, even R-mechanisms with $\rho_{R,G} \approx 1$ can yield $\bar{G}_n$ with low MSE. If the Dropout Odds are high, however, the Data Defect index needs to be small if a low MSE is to be achieved.

The analysis pursued in the paper of Question 14 (a) is focused exclusively on comparing datasets based on the performance of the sample mean as an estimator of the population mean. The ordering of datasets implied by Equation 6 is not applicable when other population statistics are of interest, although the guidance surrounding it may be useful as a heuristic. For example, we may want to estimate the population median using a sample median, or to estimate the coefficients of a regression model fit to the entire population. Estimators for these quantities would come with their own peculiarities regarding how their performance changed with the population size N and the response mechanism R. This means that when comparing datasets, it is worthwhile identifying what analysis the dataset is to be used for and to consider how the data's quality and size affect how useful this analysis is.

The sampling bias associated with our dataset could be mitigated using the methods described in Reference (1), inverse sampling and data integration. Both of these methods rely on the collection of a smaller unbiased sample however - if the shop has any records of its own kept, then the methods in this paper could in principle be used.

If we assume instead that `tips.csv` consists of the entire population, then it is possible to explore via simulation how a given response mechanism (or distribution over response mechanisms) affects quality of inference. This might be useful if the diner's owner has obtained the dataset `tips.csv` by exhaustively recording every visit to the diner and would prefer to use a less resource-intensive sampling mechanism in the future. Alternatively, it may be known what the future sampling mechanism is to be, and it is necessary to anticipate how this mechanism will affect the utility of any datasets collected.

References

(1) Kim, J. and Wang, Z. (2018). *Sampling techniques for big data analysis in finite population inference*. Department of Statistics, Iowa State University, `https://arxiv.org/pdf/1801.09728.pdf`. Accessed 20/02/2019.