Brian Weinstein | bmw2148
March 9, 2015

APMA-E4990: Homework 1

Problem 1. Counting scenarios

Case 1

In this case we have 100,000 people, each of whom calls 50 people on average. We thus have, at worst, ~ 5 million groups (keys). This is a manageable set to load into memory.

1. Load the set into memory.
2. Create a unique key for each pair of phone numbers, regardless of who called whom. One way to do this is to sort each {caller, callee} unordered pair so that the minimum of the set is in the first column and the maximum in the second.
3. Initialize an empty list
4. Loop through the table
    a. At each newly-encountered key, insert into the list the (caller/callee, duration) pair
    b. At each subsequent instance of each key, find the (caller/callee, duration) element in the list, and add this row's duration value to the one held in the list element
5. At the end of the loop, the list will have all (caller/callee, duration) key/value pairs, with the duration value as the sum of all the associated duration values in the table.

Case 2

In this case we have 10,000,000 people, each of whom calls 100 people on average. We thus have, at worst, ~ 1 billion groups, which is too large a set to load into memory. In this case we'll use a streaming approach.

1. Create a unique key for each pair of phone numbers, either as described in *Case 1*, or by some other method. For example, we could alternatively create a concatenated string of "caller>callee>caller" and aggregate call durations based on a partial match of this identifier.
2. Next, we begin reading through the data set.
3. Our approach is very similar to the one described in *Case 1*, except we no longer create a new list to store the (caller/callee, duration) pairs.
    a. At each instance of a caller/callee key, we add the duration value to the one previously associated with that key. This is easily done using the `awk` command.
4. Once we've read through the set, the last instance of each key in the data set is associated with the total call duration between the pair.

Case 3

In this case we have 300,000,000 people, each of whom calls 200 people on average. We thus have, at worst, ~ 60 trillion groups. Even though this is a huge over-estimation, the size of the data set is limiting in itself — even storing the data set on one machine is out of the question. In this case we'll use MapReduce to aggregate our call durations across multiple machines.

1. In the *Map* stage, we create key/value pairs. We define our key/value pairs to be the (caller/callee, duration) entries, where our caller/callee entries have been manipulated (sorted, concatenated, etc.) by some method described above.

2. We then shuffle the entries around so that all entries of a single key are stored on one machine.
3. In the *Reduce* stage, we sum the duration values for each caller/callee group.

Problem 2. Command line exercises

Output is summarized below. See the `citibike.sh` file for solution script.

Count the number of unique stations

```
329
```

Count the number of unique bikes

```
5699
```

Extract all of the trip start times

```
"2014-02-01 00:00:00"
"2014-02-01 00:00:03"
"2014-02-01 00:00:09"
...
"2014-02-28 23:58:17"
"2014-02-28 23:59:10"
"2014-02-28 23:59:47"
```

Count the number of trips per day

```
12771 2014-02-01
13816 2014-02-02
2600 2014-02-03
...
11188 2014-02-26
12036 2014-02-27
9587 2014-02-28
```

Find the day with the most rides

```
2014-02-02
```

Find the day with the fewest rides

```
2014-02-13
```

Find the id of the bike with the most rides

```
20837
```

*Count the number of riders by gender and birth year ----this really calculates the number of *trips* (not riders) by gender and birth year*

```
   9 "1899","1"
  68 "1900","1"
  11 "1901","1"
…
100 "1996","2"
 164 "1997","1"
  87 "1997","2"
6717 \N,"0"
```

Count the number of trips that start on cross streets that both contain numbers (e.g., "1 Ave & E 15 St", "E 39 St & 2 Ave", ...)

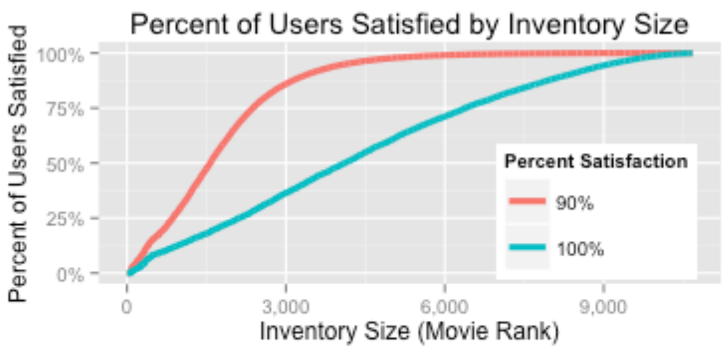
```
90549
```

Compute the average trip duration (seconds)

```
874.516
```

Problem 3. In-memory descriptive statistics

Output graph shown below. See the `eccentricity.R` file for solution script.


Percent of Users Satisfied by Inventory Size
Percent of Users Satisfied
Inventory Size (Movie Rank)
Percent Satisfaction