Foursquare data analysis

Team - 03

Mahantesh Nagendrappa Patil
Priyanka Rao
Vyomesh Patikkal Veettil

Table of Contents

1. Introduction

We are doing the analysis of Foursquare Dataset. From the project we plan to study how to work on Hadoop and do the data analysis. We expect to produce following use-case results from our project:

i) Top N checkins

ii) Mean weighted average of ratings of each venue

iii) Based on ratings [also single node vs Multinode]

a. Rating given to most checkin(visited) venues
b. Rating of least checkin(visited) venues
c. Rating with 5.0 checkin venues

iv) Busiest Hour Analysis

v) Geohash Algorithm

a. Most visited checkin location
b. Closest checkin place

2. Datasets

We took foursquare dataset from Internet Archieve from the below link
https://archive.org/details/201309_foursquare_dataset_umn
Data Size: 1368376 KB

Number of data files used: 4

Dataset content: 2153471 users, 1143092 venues, 1021970 checkins and 2809581 user ratings

File names: User.dat, Venue.dat, Ratings.dat, Checkins.dat

Column names: User.dat : User details

user_id	latitude	longitude

Venues.dat: Venue details

venue_id	latitude	longitude

Ratings.dat: Ratings given to venue by user

user_id	venue_id	rating

Checkins.dat: Checkin records

checkin_id	user_id	venue_id	latitude	longitude	created_at

3. Design Techniques

As part of the process of Data Analytics we first followed the process of ETL [Extract Transform Load]. During this process we removed the tabular form of the Foursquare data and converted to all the tables in csv format. During the analysis of data in Geohash we had to remove the rows whose latitude and altitude columns were empty, as these columns were not a mandatory field in the foursquare dataset. For doing all these process we used unix commands like sed, awk, etc.

Tools/Technology

Cloudera Hadoop, Mapreduce, Java, Python, Shell script, Excel sheet-to plot graph

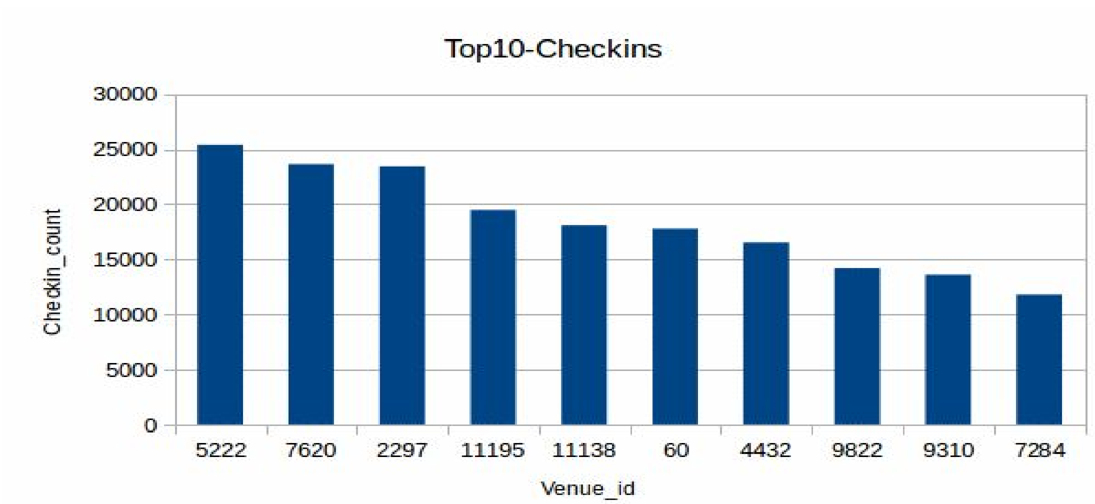
4. Implementation and Results

Here we exhibit process for our each use case and its results.

i) Top N check-ins

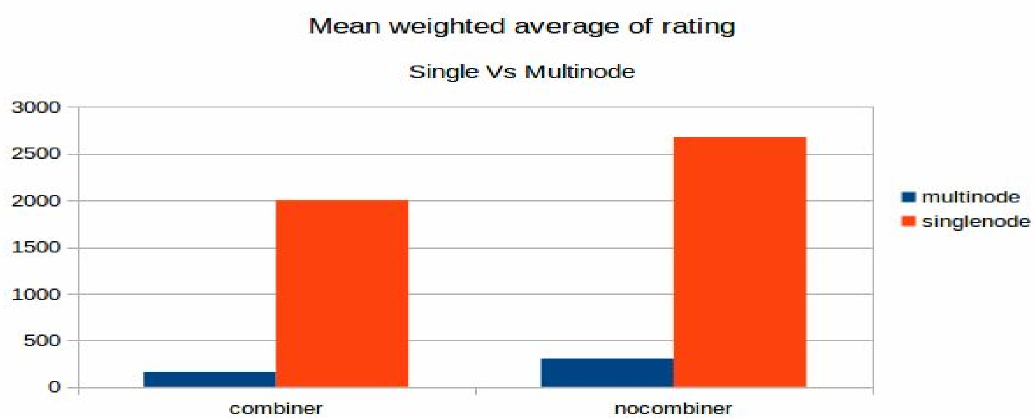
Our aim is to get Top N(eg. top 100) venues that are visited by users

We extract venue column from check-ins file. Map emits (venue_id ,1) Reduce will sum the count for each venue id (key). We will then select top 100 venues from the output file.



ii) Mean weighted average of ratings given to each venue :

Each venue is rated by user; each venue is given different rates from 1 to 5 , which is stored in ratings file. We set venue id as key and Map emits rates for each key. eg(v_id,2,2,3,4,5). The reduce function finds mean for each key and updates in output file. (eg: v_id,rate_avg).We have also compared the result using combiner and without combiner.

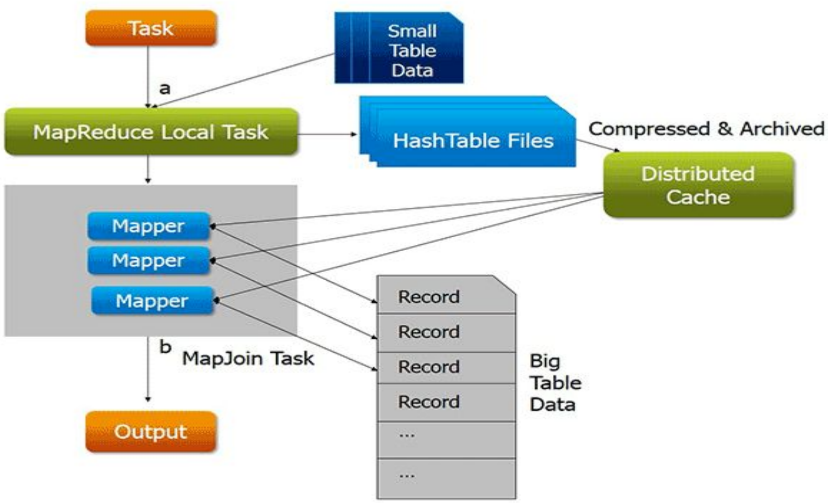


With Combiner	Without Combiner
FILE: Number of bytes read=119288448	FILE: Number of bytes read=284459586
FILE: Number of bytes written=14605349	FILE: Number of bytes written =352513928
HDFS: Number of bytes read=162955640	HDFS: Number of bytes read=162955640
HDFS: Number of bytes written =18033601	HDFS: Number of bytes written =18032904
HDFS: Number of read operations=13	HDFS: Number of read operations=13
HDFS: Number of write operations=4	HDFS: Number of write operations=4
Map-Reduce Framework	Map-Reduce Framework
Map input records=2809580	Map input records=2809580
Map output records=2809580	Map output records=2809580
Map output bytes=61810760	Map output bytes=61810760
Map output materialized bytes=26137121	Map output materialized bytes=67429926
Input split bytes=81	Input split bytes=81
Combine input records=2809580	Reduce input groups=1140494
Combine output records=1204130	Reduce shuffle bytes=67429926
Reduce input groups=1140494	Reduce input records=2809580
Reduce shuffle bytes=26137121	Reduce output records=1140494
Reduce input records=1204130	Spilled Records=8428740
Reduce output records=1140494	Shuffled Maps =1
Spilled Records=3612390	Failed Shuffles=0
Shuffled Maps =1	Merged Map outputs=1
Failed Shuffles=0	**GC time elapsed (ms)=305**
Merged Map outputs=1	CPU time spent (ms)=0

GC time elapsed (ms)=163 Total committed heap usage (bytes) =692584448 File Input Format Counters Bytes Read=81477820 File Output Format Counters Bytes Written=18033601	Total committed heap usage (bytes)=687341568 File Input Format Counters Bytes Read=81477820 File Output Format Counters Bytes Written=18032904

iii) Based on ratings [also single node vs Multinode]

Map side Join works well when one of the file is small. This small file is stored in hashmap .The setup function of map is called once for each task. We override this function to store the small file data into hashmap. Below figure explains Map side Join



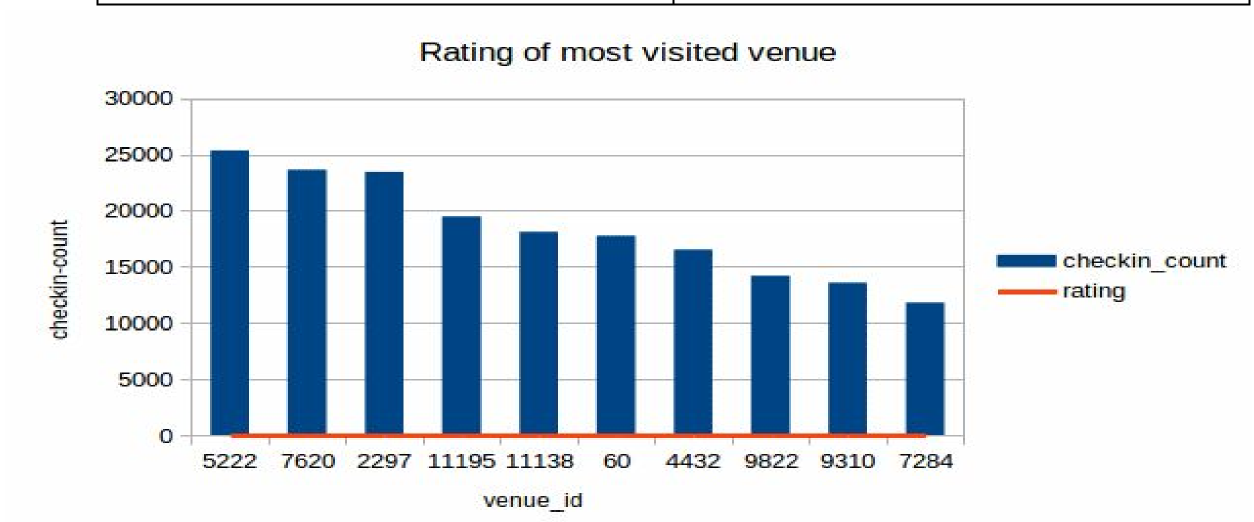
Input: The TopN venues which are stored in topN.txt, Rate_avg file(contains mean weighted average rating for each venue). A join is made based on the column venue_id.
TopN.txt (small file) [columns] - venue_id, checkin-count
Rate_avg.txt (large file) [columns] - user_id, venue_id, rate-avg
<u>Map join Process:</u>

1. Venues are read from file and stored in hash map where (key : venue_id, values: checkin-count).
2. The map then reads Rate_avg file and for each venue_id that is present in hashmap the corresponding rate is chosen.
3. The output file contains <venue_id, checkin_count,rate-avg >

<u>a. Rating given to most checkin(visited) venues</u>

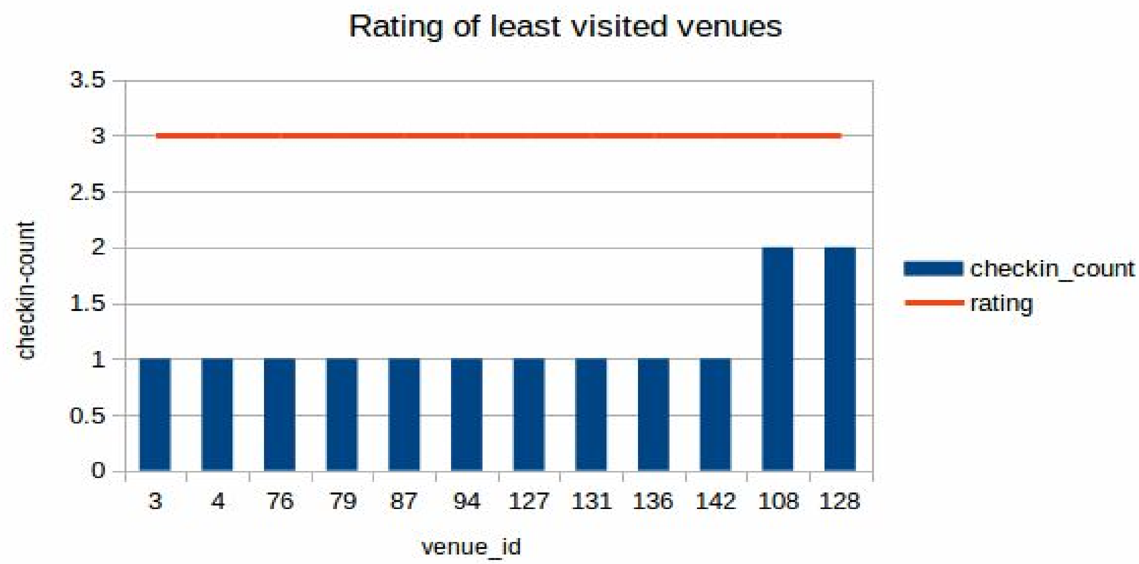
Based on the output we can say that all the top 100 venues visited have rating of 2.0 (Negative correlation).

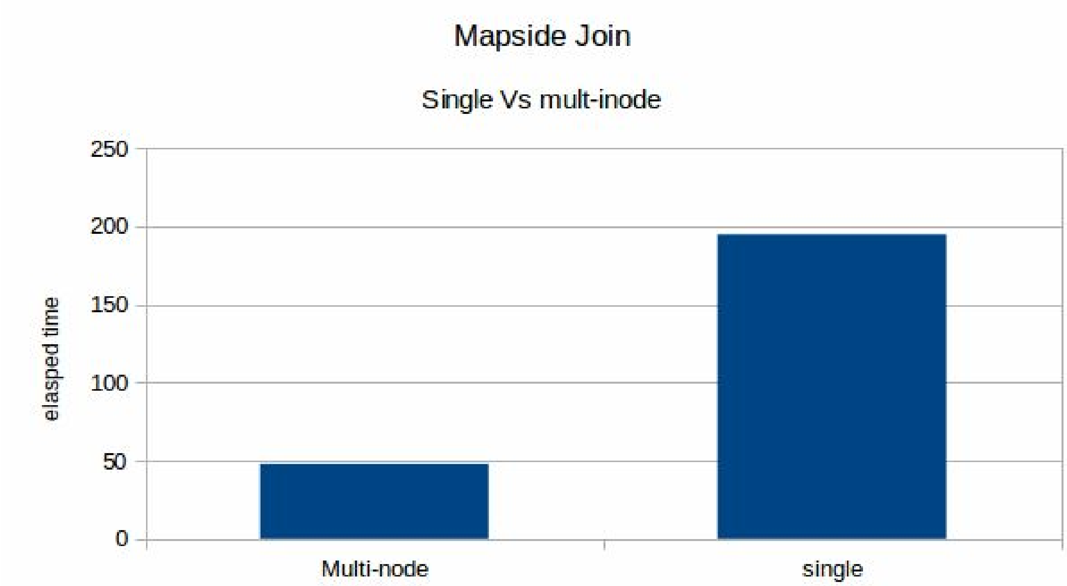
FILE: Number of bytes read=13025850 FILE: Number of bytes written=15046068 HDFS: Number of bytes read=188041930 HDFS: Number of bytes written=1102279 HDFS: Number of read operations=13 HDFS: Number of write operations=4 **Map-Reduce Framework** Map input records=1021967 Map output records=1021967 Map output bytes=15329505 Map output materialized bytes=1428006 Input split bytes=97 Combine input records=1021967 Combine output records=84000 Reduce input groups=84000 Reduce shuffle bytes=1428006	Reduce input records=84000 Reduce output records=84000 Spilled Records=168000 Shuffled Maps =1 Failed Shuffles=0 Merged Map outputs=1 **GC time elapsed (ms)=125** CPU time spent (ms)=0 Physical memory (bytes) snapshot=0 Virtual memory (bytes) snapshot=0 Total committed heap usage (bytes)=670564352 File Input Format Counters Bytes Read=94020965 File Output Format Counters Bytes Written=1102279



b. Rating of least checkin(visited) venues

We use least visited venues as input. We found that Least checked 100 venues have average rating of 3.0.





c. Rating with 5.0 checkin venues

1. Read rate_avg file and store venue_id in hash map whose rate_avg is 5.0
2. Then read the checkin-count file which we got from usecase a. which has venue_id followed by checkin-count.

3. For each venue_id that is present in hashmap get the corresponding checkin-count .
4. Output file contains venue id, checkin-count ,rate. We found very less records containing 5.0 rated venues.

```
File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=105831
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=18035051
        HDFS: Number of bytes written=3130
        HDFS: Number of read operations=6
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
Job Counters
        Launched map tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=3235
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=3235
        Total vcore-seconds taken by all map tasks=3235
        Total megabyte-seconds taken by all map tasks=3312640
Map-Reduce Framework
        Map input records=1140494
        Map output records=100
        Input split bytes=123
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=48
        CPU time spent (ms)=2050
        Physical memory (bytes) snapshot=353361920
        Virtual memory (bytes) snapshot=1545076736
        Total committed heap usage (bytes)=611844096
JoinRecord$MYCOUNTER
        MATCH_COUNT=100
        RECORD_COUNT=1140494
        size=100
File Input Format Counters
        Bytes Read=18033323
File Output Format Counters
        Bytes Written=3130
```

iv) Busiest Hour Analysis

Here we analyzed the busiest hour .This gives us an idea about the time most people prefer to go out. Reducer counts the number of check-ins per each hour.

```
15/06/13 09:54:48 INFO streaming.StreamJob: Output directory: ou
00015015        14
00015670        04
00016817        15
00018631        03
00018909        16
00021424        17
00022250        02
00022922        19
00023021        22
00023036        20
00023127        18
00023198        21
00024585        01
00024744        23
00026357        00
[vpatikka@linux60813 csvwordcount]$
```

From the above output we can see that most preferred checkin hours is midnight with total checkin as 26357.

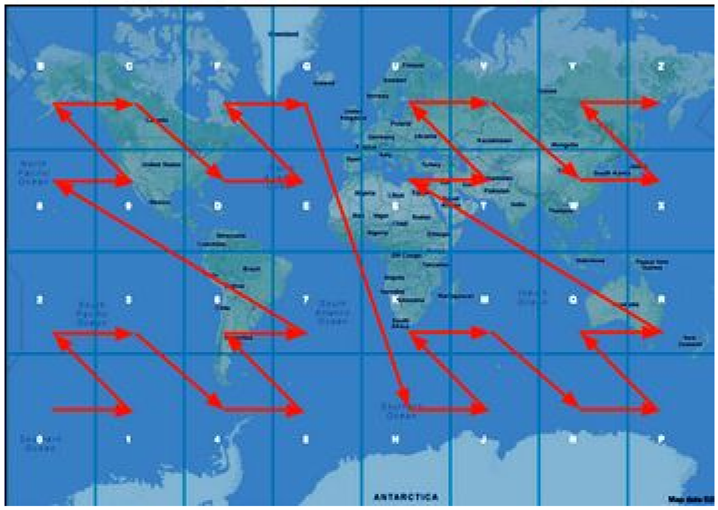
v) Geohash Algorithm

Motivation for Geohash Algorithm

In the dataset table 'checkin.dat' we got columns latitude and longitude. We realized using these geographic coordinates columns we can find the topmost locations among the checkins. Apart from that we can find closest checkin area for a specific location. For doing this initially we used K-means algorithm, but K-means was not efficient in doing this and the result was not proper. The K-means does not tell how much close the checkin area is. It was getting more complicated if we wanted to find a specific checkin area around a radius of distance. We had less time for completing this project report so we researched and came across a new idea called **Geohash Algorithm** rather than digging through K-means.

About Geohash

Geohash is a latitude/longitude geocode system invented by Gustavo Niemeyer when writing the web service at geohash.org, and put into the public domain. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape. Below diagram shows how geohash been encoded in the global map. First, the whole globe is divided in 32, 4 rows and 8 columns, and to each cell is given an alpha-numeric character. The grid is again subdivided multiple times in order to reach closer to the actual point of destination.



We incorporated the geohash concept in our dataset to easily find out closest checkin location in a specific area. Our code will search the geo-hash code that we generated from the dataset. Geohashing is done such that locations closest to each other will have **Longest Common String** (prefix)**.** The accuracy of geohash depends on number of strings in the geohash. Below table shows accuracy based on number of bits in latitude/longitude and geohash length. It shows the kilometer error at each bit.

geohash length	lat bits	lng bits	lat error	lng error	km error
1	2	3	±23	±23	±2500
2	5	5	± 2.8	± 5.6	±630
3	7	8	± 0.70	± 0.7	±78
4	10	10	± 0.087	± 0.18	±20
5	12	13	± 0.022	± 0.022	±2.4
6	15	15	± 0.0027	± 0.0055	±0.61
7	17	18	±0.00068	±0.00068	±0.076
8	20	20	±0.000085	±0.00017	±0.019

Implementation of Geohash

Developed completely on Python, we used the geohash definition for python from open-source code. Geohash code for each coordinates were generated at the mapper level of Map-Reduce process.

The user location is taken as input and compared against the list of Geohash values generated from the location (latitude , longitude) information in Venue.dat file.

a. Most visited checkin location

Here we convert coordinates to geohash from the checkin data and then take the top checkin location.

```
        File Output Format Counters
                Bytes Written=259534
15/06/13 09:24:05 INFO streaming.StreamJob: Output directory: outputsort-out
00004391        9zvxvsr77172
00004500        djgzzxyd1kxx
00004512        9qqj7nmxncgy
00004539        9q8zn1j0dv3y
00004756        9tbqj6yphh4z
00005850        9q8yndnyz704
00005860        dr5rsr5q8n0y
00005899        c23nb62w20st
00007744        dr5rm0p72j5r
00008096        dqcjqbxu6w67
00009876        dr5regy6rc6y
00010840        9tbq39n4vtkv
00013575        dr5ru2wcynvy
00014489        dr5rsh1g9x31
00015254        dp3wjztvtwnw
[vpatikka@linux60813 csvwordcount]$
```

This [dp3wjztvtwnw] has coordinates of 41.878114, -87.629798 which is a location at Chicago.

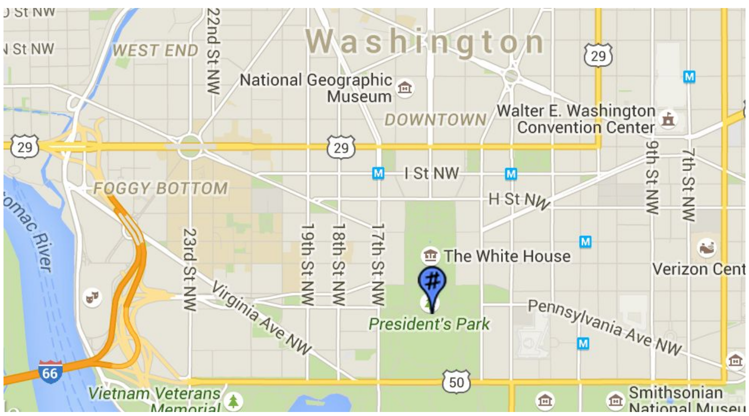
Map taken from http://geohash.org/dp3wjztvtwnw

b. Closest checkin place

Here a random location is kept inside the file and we then find the closest checkin locations through Map-reduce.

```
dqcjqbxu6w67    12
dn5br1nps7d3    1
c20fbrmcm4qj    0
9x0rvscw4wkx    0
9tbq39n4vtkv    0
9t9p7ccc38nr    0
dr5rm0p72j5r    1
9tbq39n4vtkv    0
9tbqj6yphh4z    0
drt2yzpgerb9    1
[vpatikka@linux60813 csvwordcount]$
```

In the above result we can see that all the 12 strings are matching for the given location which means there is a checkin in the present location. Others which has only first character matching has length of 1 which means the next location is 2500 km away [from the above accuracy table]. This [dqcjqbxu6w67] has coordinates of 38.895112, -77.036366 which is a location at President park, Washington.



Map taken from http://geohash.org/dqcjqbxu6w67

5. Related Work/References

1. http://www.myhadoopexamples.com/2014/04/16/hadoop-map-side-join-with-distributed-cache-example
2. https://archive.org/details/201309_foursquare_dataset_umn
3. https://en.wikipedia.org/wiki/Geohash
4. http://www.bigfastblog.com/geohash-intro
5. https://github.com/vinsci/geohash/blob/master/Geohash/geohash.py [partial]
6. Foursquare Dataset credits to: *Mohamed Sarwat, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. LARS*: A Scalable and Efficient Location-Aware Recommender System. in IEEE Transactions on Knowledge and Data Engineering TKDE*
Justin J. Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F. Mokbel. LARS: A Location-Aware Recommender System. in ICDE 2012