International Journal of Engineering Research-Online
A Peer Reviewed International Journal
Articles available online http://www.ijoer.in

Vol.3., Issue.3., 2015
(May-June)

RESEARCH ARTICLE

ISSN: 2321-7758

AN EFFICIENT APPROACH TO FIND NEAREST LOCATION USING GEOHASHING ON HADOOP AND PIG

NEERAJ KUMAR DUBEY[1], Dr. SANJAY AGRAWAL[2]

[1]Department of Computer Technology and Application, M.Tech Scholar NITTTR, Bhopal., M.P India
[2]Department of Computer Technology and Application, Professor and Dean of Research NITTTR, Bhopal., M.P, India

NEERAJ KUMAR DUBEY

ABSTRACT

Geospatial query plays an important role in modern era. "Find Nearest location" is one of the most important geospatial query. There are many methods in existence which are used to answer the query "Find Nearest Location" but somehow every methods have some drawbacks and gaps. And due to huge amount of spatial data traditional database system like RDMS are not able to process and store such large amount of data efficiently. Hadoop is an emerging technology which is used to store and process large amount of data efficiently. Pig provides a good platform to process huge amount of data in parallel. We solved the nearest locations problem by using geohashing. We proposed an algorithm for finding nearest location. Here we developed two UDF for pig one for geohashing and another for prefix matching.

Keywords—Geohashing, hadoop, pig, geocoding, UDF, Spatial Data, Spatial Query.

INTRODUCTION

Geospatial queries play an important role in modern day life. Any query which is related to geospatial data is called as geospatial query. Geospatial data can be understood as the data which is related to particular location or defines some place. Geospatial data contains the information about the geography of locations. Due to advancement of recent technology which is based on geospatial data the size of geospatial data has grown very fast. Geospatial data is accessed by geospatial query. Due to huge size of geospatial data there is need of efficient technique to process spatial query. Hadoop is an emerging technology which can process and store large data very efficiently and effectively.

Find nearest location is one of the well-known spatial queries. This is used to find the nearest location from any specified or current location. There are many methods are in existence for finding nearest location like- zip code method, K-D tree method, polygon method, Voronoi diagram etc. but each method are not able to process large data and these are not able to find nearest location with high precision.

Geohashing is a technique which is used to generate geohash code from <latitude, longitude> pair. It is known that every particular location has its own coordinates (<latitude, longitude>). We used geohash code for finding nearest places. We can find more nearer location by using geohash as compare to other methods. We used hadoop technology for processing and storing large spatial data. For the storage purpose Hadoop distributed file system is used and for processing we used pig.

The rest of the paper is organized as follows: section II discusses the background and related work; section III discusses about gaps and drawbacks in existing system; in section IV proposed algorithm is described; and finally last section IV contains the conclusion.

background and related work

Hadoop

Hadoop has emerged as an outstanding platform in the area of Big Data for data processing. It provides a reliable storage and high performance in the area of Big Data [3]. It analysis system that is scalable and build from commodity hardware. Hadoop mainly known by its two components first is its distributed storage component called HDFS (Hadoop Distributed File System). It is a single, consolidated storage platform [3]. In HDFS platform structured and complex data might be combined easily. Second and important component of Hadoop is MapReduce which is a computation framework. This framework also called as processing framework of Hadoop. It exploits the distributed storage architecture of HDFS to provide scalable, reliable parallel processing services for arbitrary algorithms [3].

Geohashing

Geohashing is a technique which is used to convert <latitude, longitude> pair into a unique alpha-numeric code. It is known that every location of the world have a <latitude, longitude> pair. Geohashing converts this <latitude, longitude> pair into a code known as geohash code. Geohash code is alphanumeric code. It is a base 32 code. Base 32 codes are generated by dividing the world into "0" and "1" and after getting a large binary number each 5 bit binary are combined for a specific character.

Pig

Pig provides an engine for executing and processing data in parallel on Hadoop. It uses a language which is known as pig Latin. Pig Latin includes number of operators like join, sort, merge, etc. which is used like traditional data operators. Apart from these operators pig also provides some facilities in which user can wright their own function for reading, processing, and writing data called as UDF (User Defined Function). Pig UDF for geohash code and UDF for prefix matching is developed for our work. UDF for geohashing is used to convert <latitude, longitude> pair into geohash code and UDF for prefix matching is used for matching of geohash codes. Pig is an open source project of apache. This means users can download pig source or binary free, use it for themselves, and contribute to it. Pig runs on hadoop it means the execution engine of pig runs on hadoop. Pig itself performs various mapreduce optimization activities. Pig runs on hadoop and it makes use of both the Hadoop Distributed File System (HDFS) for storage, MapReduce for processing.

Finding Nearest Location using ZIP code

In these days almost every business has a functionality of "Find Location" on their websites which gives nearest locations for a given address or Zip code. For answer the query of "Find Location", zip code method performs matching. It matches with zip code of all places which is stored in database one by one and the zip code which falls under the same zip code [3] , it provide that zip code as closest locations. So those places whose zip will match with particular locations zip will be displayed. It is clear that places which will fall under the same zip would belong to same region and it would be near to each other [3].

Finding the Nearest Locations Using Polygons

Finding nearest location using polygons is just an advancement of zip code method. In this method a zip code is represented by "polygon" in some geospatial framework. A polygon is an area which is defined on the map and it has certain defined boundaries [3]. One can also define its own set of polygons on the original polygons in some geospatial framework. So a new polygon can be a zip or it may be extended by any much at any point. So now the shape created after defining the polygon will be mattered. It is very simple with an address because just a polygon has to be created, which is a circle around one point [3]. Then queries can be done which will be based on all the new points that have been defined in the newly created polygon. This method is mostly adopted by sites. They display only those points or locations that fall within the defined boundary. And they can themselves define this polygon boundary which can be of 5km or 10km. so in polygon method for nearest location we can define any set of polygon of our choice with some specified boundaries and can find our desired location in that boundary [3].

Finding Nearest Location Using Voronoi Diagram

A Voronoi diagram can be used to find the nearest locations. It divides space into number of regions. To make Voronoi diagram first of all number of points must be specified, which are called as sites or seeds [3]. And then on the basis of those seeds a region is made around each site. Those regions

describes that all points in that region will be nearest point to that particular site than any other site. The defined regions are called as Voronoi cells [15].

Finding Nearest Location using K-D Tree

We can find out nearest point of interest using k-d tree. The searching procedure performed in k-d tree is more efficient and the time complexity for searching is log(n) [3]. The goal of finding nearest neighbor by k-d tree is done by searching a point in the tree that is close to a user input point. The searching process don in k-d tree by following ways.

1.Algorithm starts with root vertex and run on tree recursively, in the same way that it would if the search point were being added it means it goes left sub-tree or right sub-tree depending on whether the search point is less than or greater than the current vertex in the split dimension.

2. Once the algorithm reached at a leaf vertex then that vertex point is saved as the "current best".

3.The algorithm relax the recursion of the tree by performing the following steps at each vertex:

3.1. If the current vertex is nearer than the current best then that vertex will be declared as current best.

3.2 The algorithm also checks whether there could be any points on the other side of dividing plane that is more nearer to the search point than the current best. This process is done by intersecting the dividing hyper-plane with a hyper-sphere around the search point that has radius which is equal to the current nearest best. Usually the hyper-planes are all axis-aligned this is implemented by performing a simple comparison to see whether difference of distance between the dividing coordinate of the search point and current vertex is less than distance from search point to the current best.

3.2.1. If the hyper-sphere crosses over the plane then there could be a possibility of having nearer points on the other side of the plane by which the algorithm would step down to the other branch of the tree from the current vertex looking for closer points. This step will be performed in recursive step for entire search.

3.2.2. If the hyper-sphere doesn't intersect the plane which is used for divide, then the algorithm continues step up the tree and the whole branch on the other side of that vertex is eliminated.

4. When the algorithm finishes this process for the root vertex then it considered that searching process is complete.

Gaps in existing methods

As various strategies and algorithm to find nearest neighbors has been seen. They all are works good in particular case or situation but might be inefficient in different situations. So now we will have a case study of gaps of these methods one by one as compared to our proposed method.

Gaps in Zip code

Zip code method worked according to zip code. If someone wants to find the nearest location with respect to current location then zip code algorithms will show all the location with zip code same with provided zip code. But this method fails in various situations. Zip code method does not provide nearest location it provides range of locations. Zip code algorithm does not give accurate answer. As we know that it works on zip code number so this algorithm just consider the fact the locations which are in same zip code or mostly nearest to each other but it is not correct always. We can understand the failure of zip code method by an example. Let us say that someone is standing at the corner of a city and wants he want to find the nearest ATM. Now zip code method will give all the address of ATM which falls in the same city or have same zip code. But it is not an accurate solution because in this case he is standing at the corner of city and the ATM of other city having different zip code might be closer. It can be understood by figure 1 shows the current location which is shown by red circle and all other green circle shows location of different things like temple, pub, and restaurant in the city. Cities are represented by curvy box. And ATMs are represented by orange colored box.

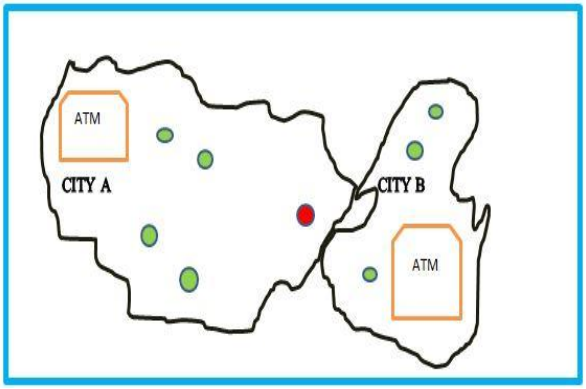

Figure 1: Gaps in zip code method

Figure 2 represents the location which was given as output of the query to find the nearest ATM. So it showed ATM of the city which is in the same zip code and so far from current location while the city B ATM is closer to current location

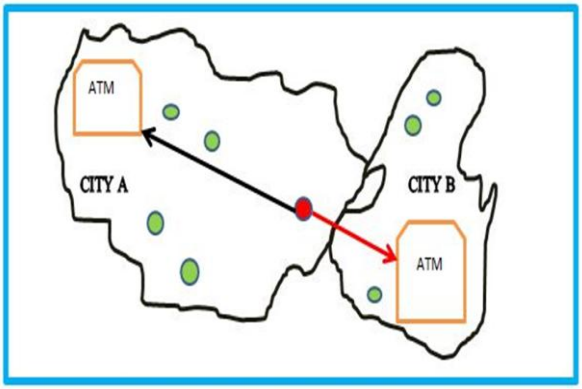

Figure 2: Gaps in zip code method

Gaps in Polygon method

Polygon method considers the zip area of a city as a polygon by this it gives the output based on the polygon area. So if anyone puts a query to find any nearest location then this method will reply based on the polygon in which the current location lies. Polygon method does not take any distance in consideration. The default polygon is the area of city but polygon may be customized. One can also draw his or her own polygon of desired area by this way result can be more precise and correct. If someone wants more precise result then he keeps on decreasing the area of polygon. But this method does not provide guarantee for nearest location. It also possible that someone may think that he can get the point of interest by continuously decreasing the area of polygon but this is impractical. It works better to find the range of location. And in polygon method there would be need of pre well defined map on which this method will be applied; only database of location would not solve this purpose.

Gaps in K-d trees

K-d trees basically works on the points. These points lie in 2D plane. In k-d tree methodology first all 2D points or converted into K-d tree and then generated K-d tree will be used to find the nearest neighbor of any point. The height of K-d tree is log(n) always by which searching process gets improved. But the problem with K-d tree method is that it cannot be suitable for search nearest point of interest as we desire, because K-d tree works on points on 2D plane from which it calculates distance between points. These points can be understood as (longitude, latitude) pair of a location but these points cannot be represented in a 2D plane, the reason is distance between two particular longitudes does not remain same always.

So this algorithm is not suitable there. Moreover if we think to use the actual distances between every two places in the database, then this approach would be infeasible and impractical.

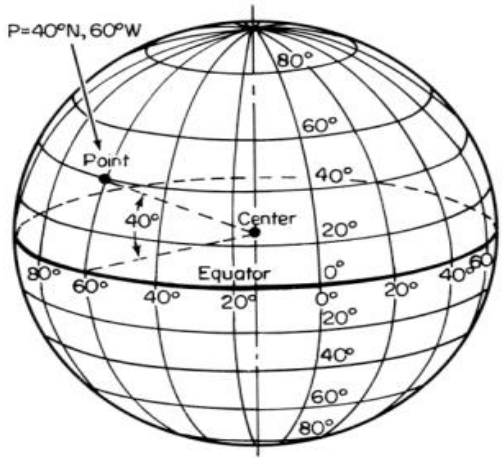

Figure 3: Longitude and latitudes of earth

Gaps in Voronoi diagrams

A Voronoi diagram is a technique of dividing a large space into number of regions. A set of points called as seeds or sites or generator is specified prior and for each seed there will be corresponding regions which consists all points closer to that seeds or generator than to any other. The regions or known as Voronoi cells. Those points can be any location in the city like ATM, school, college, pub etc. When someone queries to find the nearest point of interest from any current location then in Voronoi methods first it will be seen that current location is falling in which region. And second, the point of that region around which the region is drawn is chosen as nearest location. Now let's see the drawback of voronoi diagram algorithm, the first and big drawback of voronoi diagram method is that it is not practical to draw a very big voronoi diagram like the voronoi diagram which is converting whole nation or larger part of nation. By which it is only suitable for smaller area like city or street level. Voronoi diagram method is not suitable for unbounded data. Second drawback of voronoi diagram is- there are so many

categories of point of interest like school, college, hospital, lake, pub, theater, bank, ATM etc. One may put a query to find anyone of that or all of that or some of that. In the case of voronoi diagrams it will be required to draw different voronoi diagram for each and every different category because voronoi diagram cannot shows all the location in a single diagram. So if a voronoi diagram is implemented for a city or for an area then there would be need of so many diagrams for each category of location. So it creates a heavy load on database. So it is an inefficient method. Now it can be seen that none of the method have been implemented with HADOOP PIG yet and they are not suitable for solving in any accurate way. So the given proposed solution will be effective and efficient with respect to time and accuracy both.

IV PROPOSED ALGORITHM

The problem of finding nearest point of interest from a large database containing multiple categories data like school, college, library, restaurant in single query of user is solved by using hadoop and pig. Pig is capable to process large spatial data.

Now every location has its own (latitude longitude) value which is unique. So we will indicate the location by its (longitude, latitude) pair in our implementation. One can find different categories or different types of places which is nearer to his current location as an output of our work.

We developed an algorithm for finding nearest point of interest with use of geohash. We implemented our algorithm on pig latin. An UDF (User Defined Function) for converting (latitude, longitude) pairs is developed after that another UDF for highest common prefix match is developed which will be used for matching of geohash code. And at last our algorithm pick more nearer place from every category like library, school, ATM, park etc.

A. Algorithm for Finding Nearest Location Using Pig Latin

UDF for geohashing and prefix matching is used for finding nearest places. Algorithm 1 &2 describes the process of converting large input file having latitude, longitude pair into another input file which adds a new field called as geohash which is generated by UDF. UDF of geohashing and prefix matching are developed in a project file name as "thesis". So we have to create a jar for accessing these UDF. Following Algorithm describes the process of finding nearest places from current location.

Algorithm1

1. REGISTER A.jar// JAR file which contains the udf of geohashing and prefix matching

2. Load the input file with their schema having latitude, longitude from hdfs to pig flow

3. Generate a new input file with an extra field name as geohash by using UDF of geohashing which converts (latitude, longitude) pair into geohash code.

Algorithm 2

1. Load the current location ((latitude, longitude) pair) or user specified location from hdfs to pig flow.

2. Convert the current location's latitude and longitude into geohash by using UDF.

3. Perform matching between current geohash codes with each geohash code from stored input file by using UDF of prefix matching. And store the result with all fields of input file and one additional field name as count.

4. Group the input records which is obtained by step3 by type (e.g. library, school, airport etc.)

5. Foreach record obtained from step4 do

5.1. DLOCATION = Order each records from step3 by match count DESC;// descending order

5.2. DLIMIT = limit DLOCATION 1;// it will return only one record from each category

5.3. Generate flatten(group), flatten(address), flatten(city);

6. Store the result back into hdfs

B. Flow charts

Flowchart for Geohash Code:

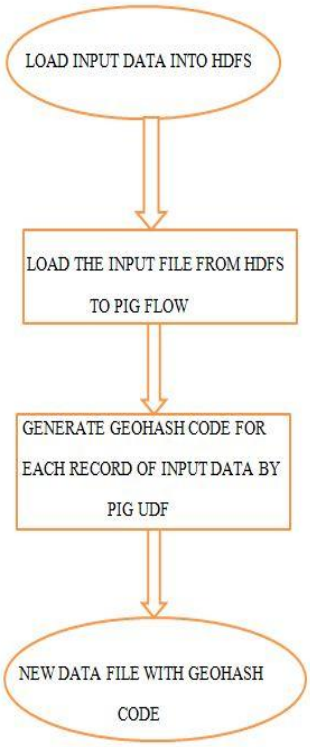

Figure 4 Generation of Geohash code for input file

The flow diagram for geohash code shows the functioning of UDF for geohash in simple way:

1. First, the entire input data file having <latitude, longitude> pair is stored from local machine to hadoop file system.
2. Now the input data file with <latitude, longitude> is loaded from HDFS into pig flow by pig Latin for operation.
3. UDF for geohash code is applied on data file for the generation of geohash code for each <latitude, longitude> pair.
4. Data file with geohash code as another field is stored again in hdfs.

Flowchart for finding nearest places:

The flow diagram 5 for finding nearest places describes the process of finding nearest places from any desired location.

- Data file with geohash code is loaded from hdfs to pig flow.
- User specified <latitude, longitude> pair is converted into geohash code by UDF
- Matching is performed between geohash codes of user specified location with every geohash code present in data file by prefix matching UDF.
- Group each record by type field.
- Data file is sorted on the basis of match count which generated by UDF of prefix matching in descending order.
- Finally records with maximum match count value from each type will be given as output by pig latin.

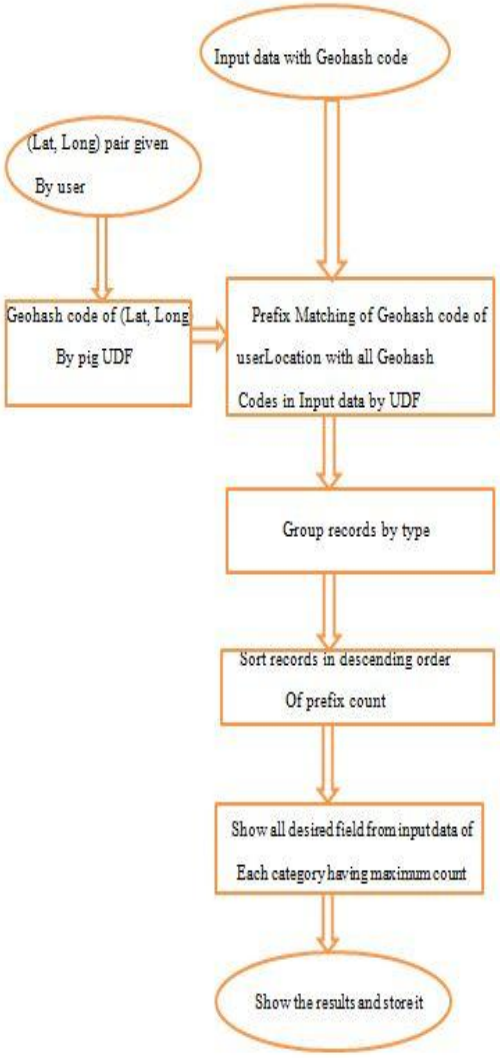

Figure 5 Flowchart for finding nearest location

Conclusion

The amount of geospatial data is growing day by day. Traditional database management system like RDBMS is not compatible to store such huge amount of data. Hadoop is an emerging technology, it provides framework to store and process huge amount of data in efficient way. Pig is a framework which is used to process large amount of

data in effective way. "Find Nearest Location" is one of most important query on geospatial data. There are many methods for finding nearest location but somehow each have some gaps and drawbacks. We have used geohashing technique for finding nearest location. Huge geospatial data is processed on hadoop cluster by hadoop component pig. We have implemented PIG UDF for geohashing. UDF for geohashing is used for converting <latitude, longitude> pair into alphanumeric code which is called as geohash code. We have used the advantage of geohash to answer the query "Find Nearest Location". A PIG UDF for prefix matching has been developed. UDF for prefix matching has been used for finding nearest location on the basis of match count. Match count can be understand as the number of character matched between geohash code of current location and Geohash code from data file. Some specified fields from each type like library, school etc having highest prefix match is returned as output.

REFERENCES

[1] Kisung Lee, Raghu k Ganti, Mudhakar Shrivasta, Ling Liu. "Efficient Spatial Query Processing for Big Data" 978-1-4503-3131-9/14/11/2014 ACM.

[2]. "Basics of Geohash" [online] en.wikipedia.org/wiki/Geohash

[3]. "Finding Nearest Location with Open Box Query using Geohashing andMapReduce" [online] www.gdeepak.com/thesisme/Finding Nearest Location

[4]. A. Akdogan, U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. "Voronoi-Based Geospatial Query Processing with MapReduce". In CLOUDCOM '10, 2010.

[5]. Afsin, Akdogan, Ugur Demiryurek, Farnoush Banaei Kashani, Cyrus Shahabi "Voronoi-Based Geospatial Query Processing with MapReduce" CloudCom Pg. 9-16, 2010.

[6]. W. Lu, Y. Shen, S. Chen, and B. C. Ooi." Efficient Processing of K Nearest Neighbor Joins Using MapReduce" Proc. VLDB Endow, 5(10), June 2012.

[7] J.M. Patel "Building a Scalable Geospatial Database System" In SIGMOD, 1997

[8]. "Geohash and its format" [online] http://geohash.org/site/tips.html

[9] "What is hadoop" [online] http://hadoop.apache.org/

[10]. "Hadoop fundamentals" [online] http://bigdatauniversity.com/courses/

[11]. C. Zhang, F. Li, and J. Jestes. "Efficient Parallel kNN Joins for Large Data in MapReduce". In EDBT'12, 2012.

[12.] S. Zhang, J. Han, Z. Liu, K. Wang, and S. Feng. "Spatial Queries Evaluation with MapReduce" In GCC '09, 2009.

[13]. "W3C recommendation: Rdf primer" http://www.w3.org/TR/rdf-primer/

[14] Der-Tsai Lee "On -Nearest Neighbor Voronoi Diagrams in the Plane" IEEE Trans. Computers, 1982.

[15]. "Basi of voronoi diagram"[online] en.wikipedia.org/wiki/Voronoi_diagram

[16]. "Basic of pig" http://developer.yahoo.com/hadoop/tutorial/.

[17]. Kai Wang, Jizhong Han, Bibo Tu, Jiao Dai, Wei Zhou, "Accelerating Spatial Data Processing with MapReduce" 1521-9097/10 2010 IEEE DOI 10.1109/ICPADS.2010.76

[18] Y onggang Wang, "Research and Implementation on Spatial Data Storage and Operation Based on Hadoop Platform" 978-1-4244-8515-4/10 20 10 IEEE

[19] Ian De Felipe, "Keyword Search on Spatial Databases" supported in part by NSF grants CNS-0320956, CNS-0220562, HRD-0317692, and IIS-0534530

[20] Ariel Cary et.al, "Experiences on Processing Spatial Data with MapReduce" supported in part by NSF grants IIS-0837716, CNS-0821345, HRD-0833093, EIA-0220562, IIS-0811922, IIP-0829576 and IIS-0534530, and equipment support by Google and IBM