Geohash Intro

By Phil Whelan on December 15, 2011

Occasionally I come across a nugget of technology that is so simple and elegant it makes me glad to be a Software Engineer. Today's such nugget is the Geohash system, which is a simple way to encode latitude and longitude and grouping nearby points on the globe with varying resolutions. It was created by Gustavo Niemeyer.

Simple Example

We have a location on the world map (see the red dot) and we want to represent this point as a single value Geohash.

First we divide the world into two halves with a vertical line and give each half a binary value of either 0 or 1. Our point on the map is in the "0″ half.

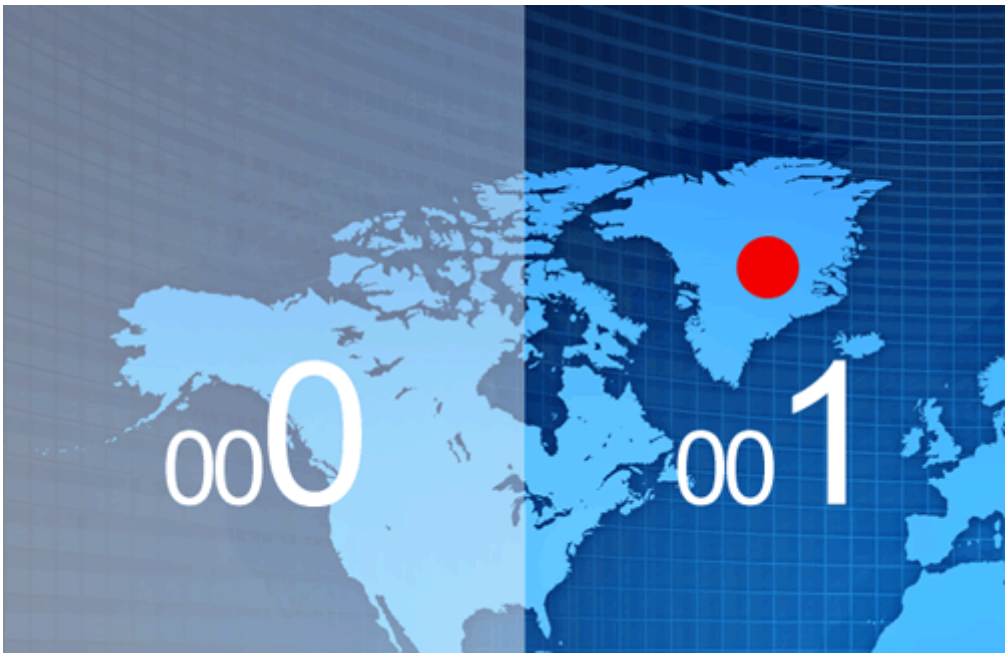
At this point we could be very unhelpful and say that the Geohash is "0″. This is lowest resolution Geohash. Obviously this is not much use if you are trying to locate a party on Saturday night and your friend tells you that the party is on the west half of planet. If he does, then take the hint and stay home.

Let’s improve the resolution of our Geohash…

Next we sub-divide our “0″ half of planet into two halves with a horizontal line and assign each half a “0″ or “1″.

We can see our red dot is located in the “0″ half, so we have a higher resolution Geohash of “00″ (“0″ from the first division, and “0″ from the sub-division).

Great! Your chances of attending the party have doubled, but we will need to raise the resolution further to pin-point more accurately.



Another sub-division gives us a location of “001″ (“0″ from the first division, and “0″ from the first sub-division and “1″ from this second sub-division).

I hope that you are getting the idea by now.

Note that each sub-division switches between dividing vertically and horizontally, dividing longitude and latitude alternately. This interweaves the longitude and latitude information to give a single value.

We can keep sub-dividing the space until we get to street-level or beyond. At that point our Geohash will be a bit longer and will look something like this…

```
00101101010111000110001100011101111000111
```

This binary can be represented as alphanumeric characters (32 bit encoded). Each 5 bits is converted to one character.

```
00101  10101  01110  00110  00110  00110  11110  00111
```

Which comes out as

```
  5      p      f      6      6      6      y      7
```

So our Geohash for the party on Saturday night is “5pf666y7″.

That’s great, but how does that help us?

Why Use A GeoHash?

Single Simple String Representation

A Geohash is fairly short and concise and does not have to be a precise location. It looks similar to a URL shortener string, such as http://bit.ly/**12C5kv**, and can be passed around on social networks such as Twitter.

Grouping Of Points

This is my favorite part about the Geohash system. Geohashs can easily to grouped. No advanced algorithms needed.

It is as simple as saying, *which other points have a Geohash that starts with the prefix “5pf66"?*. The longer the prefix the higher resolution we can zoom to.

Zooming And Aggregation

If we want to zoom in and out on a map and show a summary of how many points there are on grid square, then we can use a Geohash prefix length that is relative to the zoom resolution

In this example we use a prefix of 2 characters, which will give us the world map divided into 64 (2×32) grid squares, and a count of the number of points found in each one.

```
SELECT SUBSTR(geohash, 0, 2), COUNT(*) FROM locations GROUP BY SUBSTR(geohash, 0, 2);

"aa", 67
"ab", 456
"ac", 128
"ad", 994
"ae", 12
...
"zx", 0
"zy", 8
"zz", 5
```

A simple index on the **geohash** column of the above **locations** table allows us to focus any queries on specific areas on the map.

Caching At Scale

are question I often ask when looking at using technologies, is "how does it scale"? Can the data be partitioned? Can we cache the results? The Geohash system provides this through simply prefixes.

You can be sure that Google heavily caches all the grid squares on Google Maps at the various resolutions they provide.

All point data that relates to a specific grid square at a specific resolution can be cached by using that grid square's geohash value. Using this single value, rather than 4 non-discrete co-ordinates values (top, left, bottom, right), we can use a key-value datastore (HBase, MongoDB, Cassandra, Memcached, CouchBase, Memcached..) to store and quickly retrieve all information on that grid-square.

From our above example, if we were to update the party location details at Geohash location "5pf666y7″, we could then quickly update the aggregate information at the lower resolutions "5pf666y", "5pf666″, "5pf66″, "5pf6″, "5pf", "5p" and "5″. By doing this pre-computation we could have that data ready to be served up as-is from our key-value store, as the users of our website zoom in-and-out and move around the map.

Finding Nearest Points

An index of geohash values will store the geohashs in alphanumeric order. This means nearest points are the closest strings. This makes the 2D problem a much simplier 1D problem (see "The Gotcha!")

```
22rt841  <-- Far way party
5pf666y  <-- Somebody else's party nearby
5pf666y  <-- Our party
5pf666y  <-- Another nearby party
5r84ew3  <-- Not so close party
```

The Gotcha!

You get extra points if you have already spotted the problem with finding nearest neighbours.

The nearest point can have wildly different Geohash value, if the location is close to a grid-square boundary. This is similar to two nearby houses that reside on opposite sides of an international border (eg. Canada and USA). There will be no commonality in the addresses of these two houses.

In this example above you can see that these 2 locations (green dot and red dot) are pretty close, but they on opposite sides of a boundary. Immediately they will start with opposite binary values. Many subsequent horizontal divisions will see the green dot on the far right (“1″) of the division and the red dot on the far left (“0″). There will be very little correlation in the geohash values of these two neighbouring dots.

The Gotcha Solution

Fear not, the smart people at the institute of super smart people have a solution for you. There is an algorithm for calculating the geohash values of the 8 surrounding grid-squares of a given grid square, which widens our search net in all directions, allowing us to see nearby points that reside on the other side of a grid’s boundary.

Resources

- Spatial Search with Geohashes – David Smiley, MITRE (Lucene Revolution)
- Geohash implementation in JavaScript
- Geohash implementation in Ruby
- Geohash implementation in Java
- Geohash implementation in Python
- Geohash implementation in Perl
- Geohash implementation in Scala
- Geohash implementation in Clojure

Posted in Geospatial | Tagged clojure, geo, geographic, geohash, geospatial, java, javascript, key-value, location, maps, memcached, nearest neighbour, perl, python, ruby, scala, search | Leave a response

Leave a Reply

Name *

Email *

Website

are

Comment

Submit

Notify me of followup comments via e-mail

« Previous Next »

Top Posts

- Quora's Technology Examined
- Install Gitolite To Manage Your Git Repositories
- Homebrew - Intro To The Mac OS X Package Installer
- How To Get Experience Working With Large Datasets
- Map-Reduce With Ruby Using Hadoop
- Zero-Copy. Transfer Data Faster In Ruby
- Quickly Launch A Cassandra Cluster On Amazon EC2
- Summify's Technology Examined

Tags

amazon ec2 android apple cassandra customers data processing entrepreneur entrepreneurship file-server geo geographic geospatial git google hadoop hbase hdfs high scalability homebrew install iphone java javascript location lucene mac osx mac os x memcached mongodb mysql nosql phone postgresql python rackspace redis ruby scala solr startup tornado web-development whirr wikipedia zookeeper