SpatialSearch

{{{#!wiki red/solid ❌😞 **The most up to date information about spatial search at the Solr Reference Guide instead:** https://cwiki.apache.org/confluence/display/solr/Spatial+Search. Some details, examples, and other info is still here, however. ❌😞
}}}

⚠️ Solr3.1

Spatial Search

Introduction

Many applications wish to combine location data with text data. This is often called spatial search or geo-spatial search. Most of these applications need to do several things:

1. Represent spatial data in the index
2. Filter by some spatial concept such as a bounding box or other shape
3. Sort by distance
4. Score/boost by distance

NOTE: Unless otherwise specified, all units of distance are kilometers and points are in degrees of latitude,longitude.

New Solr 4 Spatial Field Types

Lucene 4 has a new spatial module that replaces the older one described below. The Solr adapters for it are documented here: SolrAdaptersForLuceneSpatial4. The rest of this document is about the still-supported older approach.

QuickStart

If you haven't already, download Solr, start the example server and index the example data as shown in the solr tutorial. With the Solr server running, you should be able to click on the example links and see real responses.

In the example data, certain documents have a field called "store" (with a fieldType named "location" implemented via LatLonType). Some of the points in the example data are:

```
<field name="store">45.17614,-93.87341</field>  <!-- Buffalo store -->
<field name="store">40.7143,-74.006</field>     <!-- NYC store -->
<field name="store">37.7752,-122.4232</field>   <!-- San Francisco store -->
```

Schema Configuration

This requires a location field type in schema.xml

```
  <fieldType name="location" class="solr.LatLonType" subFieldSuffix="_coordinate"/>
```

and also a dynamic field type matching the suffix to store the data points:

```
  <dynamicField name="*_coordinate"  type="tdouble" indexed="true"  stored="false"/>
```

geofilt - The distance filter

Now let's assume that we are at **45.15,-93.85** (which happens to be 3.437 km from the Buffalo store). We can use a **geofilt** filter to find all products (documents in our index) with the field **store** within **5km** of our position:

- *&fq=\{!geofilt pt=45.15,-93.85 sfield=store d=5\}

Sure enough, we find 8 products at the Buffalo store:

```
...
  "response":{"numFound":8,"start":0,"docs":[
      {
        "name":"Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133",
        "store":"45.17614,-93.87341"},
      {
        "name":"Maxtor DiamondMax 11 - hard drive - 500 GB - SATA-300",
        "store":"45.17614,-93.87341"},
...
```

Spatial Query Parameters

The main spatial search related queries, **geofilt**, **bbox**, and **geodist** default to looking for normal request parameters, so any of **pt**, **sfield**, and **dist** may be factored out and only specified once in a request (even if multiple spatial queries are used).

Examples:

- *&fq=\{!geofilt sfield=store\}&pt=45.15,-93.85&d=5
- *&fq=\{!geofilt\}&sfield=store&pt=45.15,-93.85&d=5

bbox - Bounding-box filter

Exact distance calculations can be somewhat expensive and it can often make sense to use a quick approximation instead. The **bbox** filter is guaranteed to encompass all of the points of interest, but it may also include other points that are slightly outside of the required distance. For our standard LatLonType, this is implemented as a bounding box - a box made up of a range of latitudes and longitudes that encompasses the circle of radius **d** (i.e. it will select the same or slightly more documents than **geofilt** will).

The parameters are exactly the same as **geofilt**, so the following request will still match everything in the Buffalo store:

- *&fq=\{!bbox\}&sfield=store&pt=45.15,-93.85&d=5

Because the bounding box is less selective, if we change our distance to 3km it will still include the Buffalo store (which is actually 3.437 km away). If we used the more accurate **geofilt** at 3km, these documents would not match. There are many scenarios when the bounding box can make sense though - especially if you are sorting by some other criteria anyway, or sorting by distance itself.

Since the LatLonType field also supports field queries and range queries, one can manually create their own bounding box rather than using bbox:

- ...&q=*:*&fq=store:[45,-94 TO 46,-93]

geodist - The distance function

The **geodist(param1,param2,param3)** function supports (optional) parameters:

- param1: the sfield
- param2: the latitude (pt)
- param3: the longitude (pt)

geodist is a function query that yields the calculated distance. This gives the flexibility to do a number of interesting things, such as sorting by the distance (Solr can sort by any function query), or combining the distance with the relevancy score, such as boosting by the inverse of the distance.

Here's an example of sorting by distance ascending:

- ...&q=*:*&sfield=store&pt=45.15,-93.85&sort=geodist() asc

Or you could use the distance function as the main query (or part of it) to get the distance as the document score:

- ...&q=\{!func\}geodist()&sfield=store&pt=45.15,-93.85&sort=score asc

The geodist function can have the points specified as function arguments, or can default to looking at the **pt** and **sfield** global request parameters.

Or you could combine geodist() with geofilt (or bbox) to limit the results and sort them by distance (50km):

- *&fq=\{!geofilt\}&sfield=store&pt=45.15,-93.85&d=50&sort=geodist() asc

This returns the as the score - the closest distance for 2 points that the user wants to check near (Denver and San Francisco):

- ...&sort=min(geodist(store,37.7,-122.4),geodist(store,39.7,-105))%20asc

Or

- ...&q=\{!func\}min(geodist(store,37.7,-122.4),geodist(store,39.7,-105))&sort=score%20asc

In order to return the number of results that match using a facet:

- ...&sfield=store&pt=45.15,-93.85&facet.query=\{!geofilt d=10 key=d10\}&facet.query=\{!geofilt d=20 key=d20\}&facet.query=\{!geofilt d=50 key=d50\}

Returning the distance

⚠️ Solr4.0

You can use the pseudo-field feature to return the distance along with the stored fields of each document by adding **fl=geodist()** to the request. Use an alias like **fl=*dist*:geodist()** to make the distance come back in the ***dist*** pseudo-field instead. Here is an example of sorting by distance ascending and returning the distance for each document in ***dist***.

- ...&q=*:*&sfield=store&pt=45.15,-93.85&sort=geodist() asc&fl=*dist*:geodist()

As a temporary workaround for older Solr versions, it's possible to obtain distances by using geodist or geofilt as the only scoring part of the main query.

- ...&sfield=store&pt=45.15,-93.85&sort=score%20asc&q=\{!func\}geodist()

Other Use Cases

How to combine with a sub-query to expand results

It is possible to filter by other criteria with an OR clause. Here is an example that says return by Jacksonville, FL or within 50 km from 45.15,-93.85:

- *&fq=(state:"FL" AND city:"Jacksonville") OR _query_:"\{!geofilt\}"&sfield=store&pt=45.15,-93.85&d=50&sort=geodist() asc

Note: you can't try this example with the example schema since the "state" and "city" fields haven't been defined.

How to facet by distance

Faceting by distance can be done using the frange QParser. Unfortunately, right now, it is a bit inefficient, but it likely will be fine in most situations. Note: frange is actually slower than geofilt.

- &q=*:*&sfield=store&pt=45.15,-93.85&facet.query={!frange l=0 u=5}geodist()&facet.query={!frange l=5.001 u=3000}geodist()
- ...&sfield=store&pt=45.15,-93.85&facet.query=\{!geofilt d=10 key=d10\}&facet.query=\{!geofilt d=20 key=d20\}&facet.query=\{!geofilt d=50 key=d50\}

How to boost closest results

It is possible also boost the score of a query by closest by factoring your function into the score of your main query...

- An example using the 'boost' parser with an arbitrary query...
  - ..&q={!boost b=recip(geodist(),2,200,20)}canon&fq={!geofilt}&sfield=store&pt=45.15,-93.85&d=50&sort=score desc
- An example using the 'boost' param with edismax...
  - *&fq=\{!geofilt\}&sfield=store&pt=45.15,-93.85&d=50&boost=recip(geodist(),2,200,20)&sort=score desc
- An older example using 'bf' and dismax...
  - *&fq=\{!geofilt\}&sfield=store&pt=45.15,-93.85&d=50&bf=recip(geodist(),2,200,20)&sort=score desc

Advanced Spatial Search

Solr also supports other spatial capabilities beyond just latitude and longitude. For example, a PointType can be used to represent a point in an n-dimensional space. This can be useful, for instance, for searching in CAD drawings or blueprints. Solr also supports other distance measures. See the FunctionQuery page for more information and look for hsin, ghhsin and others.

SOLR-2155

SOLR-2155 Refers to an issue in JIRA that uses spatial search techniques based on edge n-gram'ed geohashes with a PrefixTree/Trie search algorithm. SOLR-2155 started out as a patch to Solr trunk, but that part of it is ancient history now. As of September 2011, SOLR-2155 was ported to 3x and was made available as a drop-in add-on to Solr similar to its contrib modules – no patching. **If you are using Solr 3x and want a multi-valued geospatial field for filtering and/or sorting then this is for you.** It has been benchmarked showing great performance too. As the featured notice box on that JIRA issue indicates, the latest code and compiled jar are located here: https://github.com/dsmiley/SOLR-2155

Installation / Configuration

1. Go to the downloads area of the SOLR-2155 GitHub repo to get the latest jar file. Read the README.txt file visible from the GitHub front page. It pretty much repeats these instructions. Put the jar file on Solr's classpath so it's available, similar to how other Solr contrib jars are installed.

2. Then you can create a field type in `schema.xml` like so:

- <fieldType name="geohash" class="solr2155.solr.schema.GeoHashField" length="12" />

3. Add a field in the <fields> section of schema.xml

- <field name="store_geohash" type="geohash" indexed="true" stored="true" **multiValued="true"**/>

The `length` attribute refers to the length of the underlying geohash and thus the precision of the data. Refer to the table at Wikipedia to see what the error distances are for each length.

3. In your solrconfig.xml

- Top level within <config>, suggested to place at bottom:
  - <!-- Optional: alternative query parser to geofilt() – notably allows a specific lat-lon box -->
  - <queryParser name="gh_geofilt" class="solr2155.solr.search.SpatialGeoHashFilterQParser$Plugin" />
  - <!-- Optional: replace built-in geodist() with our own modified one for multi-valued geo sort -->
  - <valueSourceParser name="geodist" class="solr2155.solr.search.function.distance.
    HaversineConstFunction$HaversineValueSourceParser" />
    If you will be sorting, add the following cache into <query> section if you are going to use geodist func
  - <cache name="fieldValueCache"
    - class="solr.FastLRUCache" size="10" initialSize="1" autowarmCount="1"/>

Query

The following parameters are supported for {!geofilt}:

Parameter	Description	Example
pt	The Point to use as the center of the filter. Specified as a comma separated list of doubles. It is lat,lon.	&pt=33.4,29.0
d	The distance from the point to the outer edge of whatever is being used to filter on/ Must be greater than or equal to . Note: For geo hashing approximation is normal.	&d=10.0
sfield	The field defined with solr2155.solr.schema.GeoHashField	&sfield=store_geohash

Included with the code is gh_geofilt. Parameters are sfield, point=lat,long, and radius in meters (not km). Or a box in west,south,east,north order as follows:

- fq={!gh_geofilt sfield=store box="-98,35,-97,36"}

LatLonType

The LatLonType is the current default spatial field. Values for this type are of the form `latitude,longitude`, although behind the scenes, the latitude and longitude are indexed as separate numbers. Fields using LatLonType must be single valued (i.e. multiValued="false"). This field type does distance calculations based on Great Circle (haversine).

In addition to `geofilt`, `geodist` and `bbox`, the LatLonType supports field queries such as `field:10,20` and range queries such as `field:[10,20 TO 30,40]`.

Clustering / Heatmap

For info on spatially aggregating nearby points to reduce the raw coordinate density:
SpatialClustering

Filtering Caveats

For the `bbox` filter, when the bounding box includes a pole, the LatLonType will switch from producing a bounding box to a "bounding bowl" (i.e. a spherical cap) whereby it will include all values that are North or South of the latitude of the would be bounding box (the lower left and the upper right) that is closer to the equator. In other words, we still calculate what the coordinates of the upper right corner and the lower left corner of the box would be just as in all other filtering cases, but we then take the corner that is closest to the equator (since it goes over the pole it may not be the lower left, despite the name) and do a latitude only filter. Obviously, this means there will be more matches than a pure bounding box match, but the query is much easier to construct and will likely be faster, too.

Spatial Options Under Development

SpatialSearchDev – Covers things like Geohash (supports multivalue lat-lon points), other distance functions, etc.