erLocator

Location, location, location.

Overview - Objective

Make Erlang more accessible and less daunting.

The demonstration application will:

- Leverage a NoSQL backend
- Use Natively Implemented Functions
- Exhibit eDoc and Rebar
- and provide an interesting feature that might actually be useful.

The application is hosted on GitHub and licensed MIT.

Demo: erlocator.org

Code: github.com/erlocator/erlocator.git

Overview - Concept

Track user locations such that nearby users can be found efficiently.

- Do not perform radius or distance calculations on each data element.
- Do not perform range filtering by latitude and longitude.

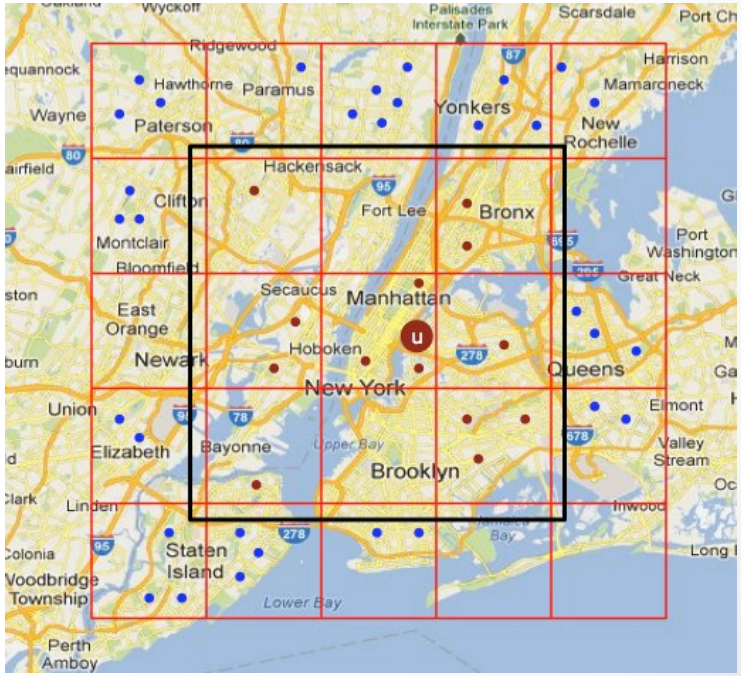
- Store user entries by region.
- Return users within the same and adjacent regions.

Where the user U is in the center region, return the records of users in the 3 x 3 grid of regions surrounding that user.

In the diagram, users represented as red dots are returned, while users in blue are outside the range.

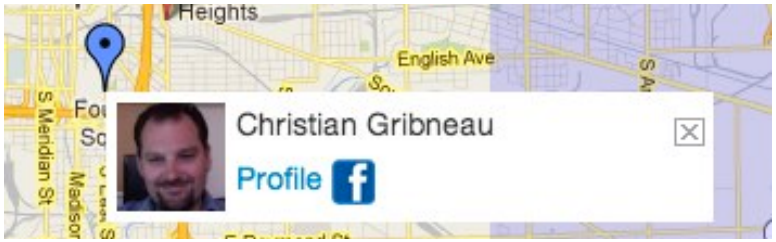
This is exposed as a web api.

There is no security – deploy wisely.

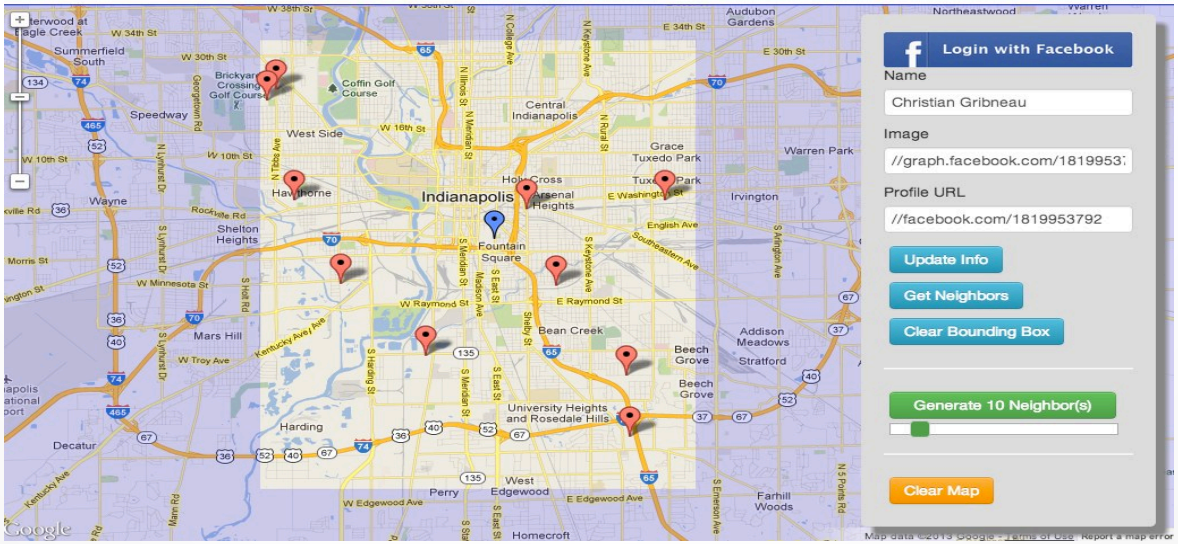


Overview – Front End

- The front end simply demonstrates the back end.
- Components
  - Twitter bootstrap: reasonable design out of the box.
  - Google Maps API: familiar presentation of geolocation data.
  - Facebook javascript API: expose user identity with location.
- Licensed: MIT
- View it on the web
  - http://erlocator.org



Erlocator Demo



Overview – Back End

- A very simple erlang application.
- Core Technology
  - Store users in small regions based on geolocation.
  - Return a set of those regions so that users can find others nearby.
- Demonstrates
  - Geonum for numerical hashes that support bitwise masking.
  - Natively Implemented Functions in C for performance.
  - Redis to store in-memory data structures with Redo.
  - Mochiweb to expose a web api.
  - eDoc to document code.
  - Rebar for builds.
- Licensed: MIT
- Get it on Github
  - erlocator/erlocator
  - erlocator/geonum

Backend - GeoNum

We group user locations within ranges by hashing the latitude and longitude using a numeric version of GeoHash called GeoNum. The hashing algorithm simply divides latitude and longitude in half repetitively, assigning 1 for larger and 0 for smaller values in the initial data. This process continues until the desired precision has been encoded, and the bits of latitude and longitude are interleaved in a single integer padded with 1 in the most significant bit.

We calculate a GeoNum with 25 bits of overall precision. This same hash will be returned for all locations within that region.

LAT 37.7891726

LNG -122.4102559

➔ GeoNum 43721694

GeoNum

1010011011001000111011110

Backend – GeoNum NIF – Erlang

Small routines that are compute intensive can be written in C and used in Erlang applications as Natively Implemented Functions. The NIF in this application is here (GitHub: erlocator/geonum)

To use C functions, init() loads the shared library on module load:

```
erlang:load_nif(SoName, 0)
```

Then we call appropriately named Erlang wrappers. In this example, the NIF is named geonum, and the function is named encode:

```
 %% @doc Encode latitude and longitude into a geonum
-spec encode(float(), float(), pos_integer()) -> binary().
encode(_Latitude, _Longitude, _Precision) ->
    exit(geonum_nif_not_loaded).
```

The example wraps the NIF in Erlang and exports its functions:

```
-module(geonum).

-export([
    encode/3,
]).
```

Documentation: http://www.erlang.org/doc/tutorial/nif.html

Backend – GeoNum NIF - C

The C includes the Erlang NIF header, exports the functions with ERL_NIF_INIT, and prepares the return values to be understood by Erlang.

```
#include "erl_nif_compat.h"
…

/**
 * Erlang Wrapper for geonum_encode
 */
ERL_NIF_TERM
erl_geonum_encode(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    … function code here …
    return make_ok(env, enif_make_int64(env, bin_geonum[0]));
}

…

static ErlNifFunc nif_functions[] = {
    {"encode", 3, erl_geonum_encode}
};

ERL_NIF_INIT(geonum, nif_functions, &on_load, &on_reload, &on_upgrade, NULL);
```

Documentation: http://www.erlang.org/doc/tutorial/nif.html

Back End – Redis

Redis (**RE**mote **DI**ctionary **S**ervice)

- Stores and serves foundational data structures.
- Provides a sort of shared memory for many systems.
- Very fast.
- Limited horizontal scaling capabilities at present.
- Presently single-threaded.
- See: http://redis.io/topics/benchmarks
- Configurable to store only in RAM.
- BSD License
- http://redis.io/
- Command reference: http://redis.io/commands

Optimal Redis Use Case

- Small data.
  - Full data set must fit in physical RAM.
- Frequent reads and writes.
- Transient.

Back End – Redis Keys and Redo

- Redis key structure
  - **geonum:**{numeric hash}
    - SET of user IDs within the specified region.
  - **geonum_user:**{facebook id}
    - STRING containing user information.
      - Name
      - Picture
      - Profile link
      - Full geolocation information
      - For convenience, this data is stored as an Erlang term representing pre-parsed JSON.
  - **geonum_expire**
    - ZSET (sorted set)
      - User IDs
      - Scored by future expiration time
- Redo
  - Very, very simple implementation of Redis Erlang client.
  - One api function, takes a raw Redis command: redo:cmd
  - https://github.com/JacobVorreuter/redo
  - MIT License (apparently).

Backend - Mochiweb

- Erlang library for building lightweight http servers.
- https://github.com/mochi/mochiweb

URL Routing

```
loop(Req, DocRoot, AppParams) ->
    "/" ++ Path = Req:get(path),
    case Req:get(method) of
        Method when Method =:= 'GET'; Method =:= 'HEAD' ->
            Params = Req:parse_qs(),
            case Path of
                "" ->
                    Req:serve_file("html/hello.html", DocRoot)
            end;

        'POST' ->
            Params = Req:parse_post(),
            case Path of
                ”url" ->
                    code;
                _ ->
                    Req:not_found()
            end;
        _ ->
            Req:respond({501, [], []})
    end.
```

Backend - Rebar

- Widely accepted Erlang build tool.
- Apache license.
- GitHub: basho/rebar (https://github.com/rebar/rebar/wiki/rebar)
- Call rebar when running make.
- Makefile

```
REBAR=rebar

DEPS_EBIN = `find . -name ebin -type d`

all:
	@$(REBAR) get-deps
	for a in deps/*; do cd $$a; make; cd -; done
	@$(REBAR) compile
	@$(REBAR) doc
clean:
	for a in deps/*; do cd $$a; make clean; cd -; done
	@$(REBAR) clean
```

- Rebar.config

```
{deps, [
	{geonum,  "0.1.0", {git, "git://github.com/erlocator/geonum.git"}}
]}.
%%{lib_dirs,["deps"]}.
%%{erl_opts,[]}.
```

Backend - eDoc

- eDoc is the standard for documenting Erlang applications.
- http://www.erlang.org/doc/apps/edoc/chapter.html
- Generate documentation: rebar doc
- Overview: doc/overview.edoc
  - An overview of the application.
  - Top page of generated documentation.
- File headers:

```
%% @author Boris Okner <boris.okner@gmail.com>
%% @author Josh Murphy <jmurphy@lostbitz.com>
%% @author Christian Gribneau <christian@gribneau.net>
%% @copyright 2013
%% @doc Web server for geofilter.
```

- Function descriptions:

```
%% @doc Stop the geofilter webserver.
stop() ->
    mochiweb_http:stop(?MODULE).
```

API

- GET
  - geo/neighbors geonum
    Returns the users in a 3x3 grid of regions around the specified region.
  - geo/bbox geonum
    Returns the extremities of the 3x3 grid of regions around the specified region, and the extremities of the specified region itself.
- POST
  - geo/set UserId, Lat, Lon, +++
    Creates or updates data for the specified user, returns the geonum hash.
  - geo/delete UserID
    Removes the data for the specified user. Returns empty HTTP 200.
  - geo/generate geonum, n
    Generate n test users around region geonum. Returns empty HTTP 200.
  - geo/flushall
    Clear all keys in Redis. Returns empty HTTP 200.

Conclusion

Questions?