SoWS, Assignment 3 Report

By:

Arleth, Johan

Ulrik, Kenneth Ry

Note: Parts of the report may contain textual pieces from the Assignment 2 report.

Date:

November 29, 2017

1 Introduction

Assignment 1 in the course Scalabilty of Web Systems asked the students to build a web-server that can access the Sentinel 2 datasets available online, by allowing a web-browser user to supply search parameters in the shape of longitude and latitude values, and the web-server should use these parameters to find the closest Sentinel 2 image of the ground, unfiltered in terms of time and return a direct (download) link to that image.

The purpose of "Assignment 2" was then to extend upon Assignment 1, by (a) allowing the user to supply a range of latitude as well as longitude parameters, and retrieving all images within this rectangular space on earth, and (b) allow for various ways to rank by the average color in an image, or by the distance to a desired color, or by the distance to the euclidean line between two colors in the 3-dimensional color-space.

All of the color-related sorting-options have been excluded due to technical properties of the bridge between Go and C (which has a library required to perform reading of the ".jp2" files on the Sentinel 2 dataset) and deployment to the Google Cloud AppEngine.

Finally, Assignment 3, as is the primary topic of this report, then aims to support querying for any country in the world, looking said country up on "Geofabrik.de" to retrieve a so-called Planar straight-line graph for the country. This PSLG can then be converted - once again utilizing Sentinel 2 client libraries - into their RegionCover type, which is essentially a collection of planar rectangles as longitude and latitude points (two diagonal corners of the rectangle). Finally, one can extract various "CellUnions" stored in the RegionCover type, which then contain the actual rectangular longitude and latitude information.

The idea, then, is to give users an easy way to get an idea about the comparative size of any country in the world, or at least an idea of how well covered said country is by the Sentinel 2 dataset. What is to be done, is to return the amount of images stored for the queried country, optionally filtered by a time-range, for instance to grab only the most resent images stored.

2 Design

2.1 Looking up a country

We have made a new handler "/countries" which can take 3 different query parameters: "country", "fromtime" and "totime".

We use "country" to take the user's wish for which country to attempt to look up on Geofabrik. The other two parameters are used to determine which time period we want Sentinel 2 image data from. This enables the user to compare the total image data for a country with how much data exists for certain time-frames. They may even discover longer periods of time with no data recorded for an entire country at all - who knows?

By using the country polygon we can get from Geofrabrik, we can - after translating from PSLG to CellUnion cells from the Sentinel 2 client library - query up against BigQuery for images that are within the country polygon, or which contains one of the corner points of at least one Sentinel 2 area "Cell".

3 Implementation

3.1 Country lookup

3.1.1 Geofabrik

Since the Geofabrik data polygons of the PSLG format is grouped in regions of the world, we use a string array of these region identifiers and iterate these, and attempt to look up the queried country under each of these regions, one at a time. This is a rather simplistic approach, which of course potentially creates some overhead, if the user happens to query a lot of countries in the region that happens to be attempted to look up in last. An optimization would be to cache the relationships over time, as we learn the mappings of the Geofabrik data one by one. We decided against this to spare time.

Here is the code for the region-country lookup attempts. Note that we only return if there is no error when performing an HTTP GET on the string-interpolated address:

```
regions := [...]string{"europe", "north-america", "south-america",
   "asia", "central-america", "australia-ocenania", "africa",
   "antarctica"}
```

```
for _, region := range regions {
   resp, err :=
      client.Get(fmt.Sprintf("http://download.geofabrik.de/%s/%s.poly",
      region, country))
   if err == nil {
        bla := parseGeofabrikResponse(w, resp)

        bla2 := handlePolygon(w, bla)

    return countImages(ctx, w, bla2, timeArg1, timeArg2) // <-- TODO:
       Call JARL method instead of dummy
   }
}
```

Once we succeed in getting a result from Geofabrik, we must parse their results, which are given as pairs of latitude and longitude, one per line. We simply store this in a slice of arrays of size 2.

Using this set of latitude-longitude pairs, we then use the provided code to translate this polygon from Geofabrik to the Loop, then the Polygon and finally to a region-cover in the shape of a CellUnion. From this we read from the individual cells and add these to another sub-result of our logic, this time stored in a slice of arrays of size 4, to be able to store two latitude and two longitude values for each Cell, representing its rectangular space, which is then passed on to our BigQuery querying logic.

The code to read the two pairs of latitude-longitude from the retrieved Cells is a simple question of data-access, as the following snippet illustrates:

```
for i := 0; i < len(cover); i++ {
   c = s2.CellFromCellID(cover[i])
   // Store result-set with the 4 values representing the two ranges
      for lat and long
   res = append(res, [4]float64{c.RectBound().Lo().Lat.Degrees(),
      c.RectBound().Hi().Lat.Degrees(),
      c.RectBound().Lo().Lng.Degrees(),
      c.RectBound().Hi().Lng.Degrees()})

   totalArea += c.RectBound().Area()
```

```
}
```

3.1.2 BigQuery

For Assignment 3, we have reused the client setup for BigQuery that we created in the previous Assignment 2. The only necessary change is to build a different query. Instead of looking for an image which contain one point, we build a huge conditional query.

First we add conditions for all images which are contained within the squares created by the Geofabrik country polygon, namely the rectangles given by the so-called "CellUnions".

We then add additional conditions for any images where the corners of a CellUnion square is within that image's area.

An illustration of the two clause-descriptions just mentioned can be seen in figure 1.

And for a quick, optional read, here is the related code to build up the query:

```
for _, rect := range rectangles {
   if !first {
        queryString += " or ( "
   } else {
        queryString += " ( "
        first = false
   }
   lat1 := rect[0]
   lat2 := rect[1]
   long1 := rect[2]
   long2 := rect[3]

   queryString += fmt.Sprintf(" north_lat < %g and west_lon > %g and
      south_lat > %g and east_lon < %g )", lat2, long1, lat1, long2)
   queryString += fmt.Sprintf(" or ((north_lat > %g and south_lat < %g)
      and (west_lon < %g and east_lon > %g) )", lat2, lat2, long1,
      long1)
}

if time1 != "" && time2 != "" {
   queryString += " and (generation_time > " + time1 + " and
      generation_time < " + time2 + " )"
```

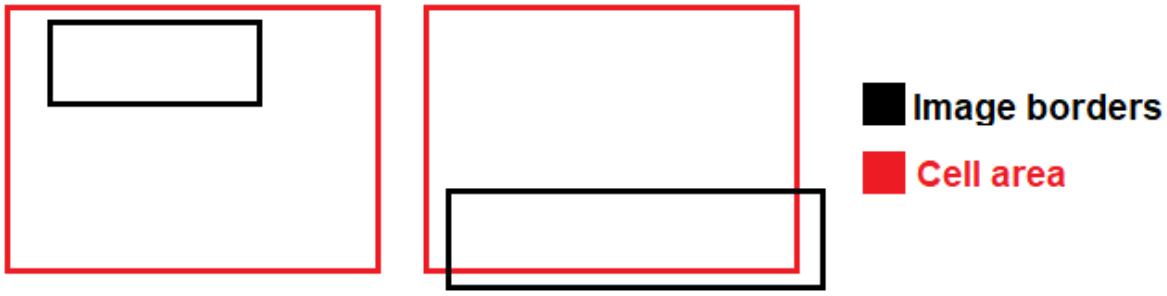

Figure 1: An illustration of the desired SQL WHERE clauses to be created for every pair of latitude-longitude corners that defines the area of a country.

```
}
```

Finally, we add the optional time constraints if the fitting query-parameters are given, which is taken as a start time and an end time, in which the images can have been generated. The format of these is a string following the simple format: "yyyymmdd". For instance, "20160317" represents the 17th of March, 2016.

The final result is the count of how many rows this query finds, where each row represents one so-called "base URL", which contains loads of meta-data and in the end represents a single image found. In other words, with a single SQL query up against BigQuery, we are able to reach our final result.

4 Benchmarking

4.1 Method

We have used an ASUS X550JX[1] connected via LAN on the ITU internet for benchmarking our solution.

What will be showcased in this benchmarking section is only the addition that Assignment 3 has asked, as per the description in the introduction of this report. To find details of the other supported functionality in the delivered source code, please refer to the relevant Assignment 1 and Assignment 2 reports for analyses of these pieces of functionality.

Our testing will involve querying various countries of various sizes, in an attempt to see developments in the amount of images returned as well as the amount of time that the web-service

[1] `https://www.komplett.dk/product/853766#`

handler takes to handle our request.

4.2 Expectation and predictions

Assuming that the Sentinel 2 dataset contains data at a fairly equal rate across all countries, and that the amount of images stored somewhat correlates to the area of the country, which we would indeed expect, then it would seem to be most obvious to expect a linear development in the amount of images retrieved, in relation to the size of the queried countries.

Also, in terms of execution time, we would in fact NOT expect any significant differences in the time that it takes to fetch the amount of images stored. This is because - as described previously - we use the BigQuery client library, which essentially allows us to use a *single SQL request* to get the count of images for an amount of areas queried for by expanding the WHERE clause of the SQL statement that we use for BigQuery.

4.3 Results

Our benchmarking results are shown in figure 2.

What immediately jumps to eye is the seemingly un-existing relationship between amount of images and execution time. This is even though we were on a stable LAN connection.

While we can't say much for the reasoning for the varying execution-time (and in particular the odd data-point for Portugal, which in particular undermines the theory that the returned count and the execution-time should be related), the depicted data does support our hypothesis, that the execution time will continue to be rather constant across varying amounts of images fetched.

If there is actually a relationship from time spent to something regarding the SQL-query overhead, it would probably regard the amount of data that is touched upon by the query and its WHERE clauses. Since the where clauses are much the same, but with different values, we believe that the fact, that every query (albeit a different country) queries across the same scope of data, could be the primary reasoning factor that the handling time is somewhat the same throughout the tested countries.

In terms of the data itself, it is rather curious that so many images are returned for Portugal. We have to presume that for one reason or another, there is simply a lot more coverage of Portugal over time in the Sentinel 2 dataset than relatively for the other countries.

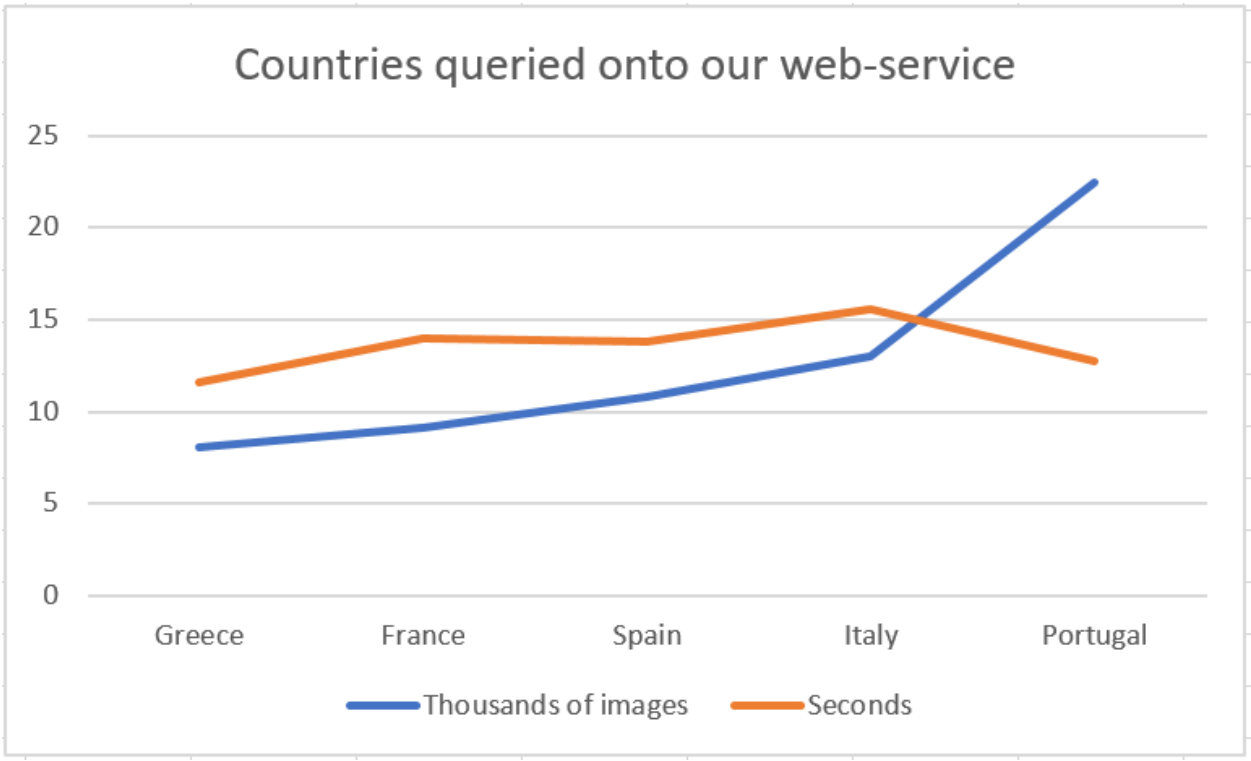

Figure 2: Our benchmarking results visualized. The X-axis is the queried countries, and the first series shows the amount of thousands of images outputted by the service for the queried country, when no time-constraints are supplied. Finally, the second series showcases the amount of seconds the query for the country took.