Scalability Project 3

Nicoline Scheel - `nisch@itu.dk`
Dennis Thinh Tan Nguyen - `dttn@itu.dk`

November 22, 2017

1 Implementation

The following sections describe the implementation of the web service, which allows users to query for meaningful areas, e.g. a country.

1.1 Parsing of input from user

Submit a report describing your design and implementation of both the service as well as an evaluation of how your service scales with the size of a country. You should use snippets from your code to illustrate your design decisions.

The user posts the data of the poly file in the body of the request to the url https://nisch-179108.appspot.com/imagesByPSLG. This means that the service is not limited to the poly files as defined by GeoFabrik, but you can also input other poly files.

The input is parsed by iterating over each line in the body, splitting each line by whitespace and parsing each latitude and longitude as float. After parsing latitude and longitude to float, a point is created and appended to a list of points. At the end, this is returned to a method which creates the region cover and operation approximation.

1.2 Finding the region cover

The parsed points is returned to a function getImageCover, which uses S2's RegionCover type to create the approximation of resulting polygon. This code is based on the provided code in the assignment description. A polygon is created from the points, which the RegionCoverer takes as input to create a region cover with maximum 100 cells.

The cover is iterated over in the function IterateCover. Within the loop that iterates over the cover, the cell is retrieved by its id and the bounds of the cell are retrieved. These bounds are then used in a goroutine to query BigQuery for the given granules for those bounds. That is, a goroutine is started for every cell, which queries BigQuery. For every granule within the bounds of a given cell, a count of 13 (which is the standard number of images per granule) is atomically added to an atomic count which is propagated to all the goroutines. To ensure that all goroutines finishes before the total count is returned, sync.WaitGroup is used to await the routines. Once all covers has been parsed, the total result is returned and encoded to the output writer.

2 Scalability of the Web Service

The fact that we always estimate the number of images to be 13 for each granule is a trade-off we had to make in order to achieve scalability. Querying Google Storage for every granule creates a large overhead both in terms of latency and memory. In terms of memory, we have to keep the granules for the given cells in memory (this will potentially be a lot of memory, depending on how many goroutines are running in parallel), while we query Google Storage for the contents of the bucket. Furthermore, it would potentially be an extreme number of times you have to query the Google Storage, causing large overhead. Querying BigQuery is not as large a bottleneck, because we potentially only query BigQuery for each cell, which is maximally 100 times. However, there is a much larger number of granules for each cell, which is why the requests to Google Storage happens much more frequently.

You can think of the number of images returned by the service as an approximation of the number of images. If Google decided to include more or less images per granule, this approximation could potentially be very wrong, but in order to achieve scalability, this is a choice we have made.

Furthermore, our implementation is designed to be concurrent by using goroutines and communication across goroutines through channels. While it is unlikely that you have 100 processing cores at your disposal, every cell in the region cover can potentially be handled in parallel, because querying BigQuery and fetching the number of images for the granules contained in the cell is designed to be concurrent. Therefore, if the service was handling a particular country size slowly, it is always possible to scale the hardware horizontally or vertically to cater for this.