Scalability of Web Systems - Project 1

Thor Olesen and Stig Killendahl

October 16, 2017

Contents

1 Image Data

How is the image data stored? How is metadata of images stored?

The Sentinel-2 satellite images are stores in the JPEG 2000 file format that is an image compression standard. Further, the images are organized in a Sentinel-2 tiling grid, where each tile covers a 100 km$\hat{2}$ division on earth (i.e. MGRS). Conceptually, such an area on earth is represented as a tile that is composed of a set of images (see 1. In this regard, a 'granule' is a tile that is part of a given grid square at a given point in time. By explanation, the Sentinel-2 satellite may take many pictures of a given earth division (i.e. square tile) at different points in time, where each image has an ID referring to such a granule.

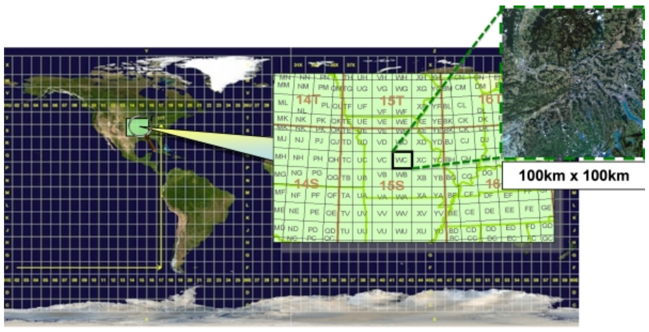

Figure 1: Tiling grid

In terms of the data structure, the granules have the following directory structure in the Sentinel-2 Cloud Storage bucket[1]:

Directory Structure : /tiles/[UTM_ZONE]/[LATITUDE_BAND]/[GRID_SQUARE]/[GRANULE_ID]/...

[UTM_ZONE] : number indicating the longitude zone

[LATITUDE_BAND] : letter in "C" through "X" without "I" and "O", indicating the latitude

[GRID_SQUARE] : two-letter code related to a particular 100 km square region (i.e. granule)

[GRANULE_ID] : ID of a granule, which contains tiles (sets of images) of a grid square at a given time

Granule example in German grid square: `gs://gcp-public-data-sentinel-2/tiles/33/U/UP/`

Inside the [GRANULE_ID] directory, the granule images and their metadata are organized based on the SAFE format[2]. The "GRANULE" subdirectory holds the granule data.

Granule subdirectories (image and meta data)

IMG_DATA : Contains JPEG 2000 image data files

QI_DATA : Contains quality control reports for granule

AUX_DATA : Contains weather forecast data for granule

For the assignment, any data of interest may be found in an index CSV file that lists the granules and their spatial extent as minimum and maximum longitudes and latitudes: `gs://gcp-public-data-sentinel-2/index.csv.gz`.

This data is queried using SQL in BigQuery: `https://bigquery.cloud.google.com/table/bigquery-public-data:cloud_storage_geo_index.sentinel_2_index`

[1]A Bucket represents a container that holds data in Google Cloud Storage
[2]Standard Archive Format for Europe is a standard format for achiving and conveying Earth observation data within the European Space Agency (ESA). Link: http://earth.esa.int/SAFE/ seen 16/10/17

2 Web Service

The web service is located in **"service.go"** and accepts GET requests with query parameters based on a given location. The "query.go" file contains the functionality used to SQL query granule data through the BigQuery API based on the following request format:

HTTP GET Request Format: `https://tvao-178408.appspot.com/images?lat=...&lng=...`

As a result, the id of all granules that match the provided location is returned.

3 Go Server Side Programming

How do Go language features support your server side programming?

By definition, server-side web programming is about making web applications that deliver formatted data in response to HTTP requests. In this regard, Go offers the built-in **net/http** package to build web applications. Specifically, this is used to create a web service that serves data in a JSON format through the HTTP protocol. Apart from this, the **encoding/json** package is used to encode and decode JSON data, the **errors** package for error handling, **regexp** for input validation with regular expressions and **log** package for logging system events.

The init function below is run before the execution of the application and begins with a call to **http.HandleFunc**, which tells the **http** package to handle all requests to the route ("/images") with the **appHandler**. NB: **http.ListenAndServe(":8080", nil)** may be used to start the http server on port 8080.

```go
func init() {
    http.HandleFunc("/", redirect)
    http.Handle("/images", appHandler(images))
}
```

The **appHandler** function below is of type **http.HandlerFunc** that takes an **http.ResponseWriter** and http.Request. Notice that composition is used in Go to make **appHandler** a composite of a handler and error, which enables more user-friendly and fine-grained error handling in the web service. In this regard, it must satisfy the **http.Handler** interface and implement a **ServeHTTP** method as shown below.

```go
type appHandler func(http.ResponseWriter, *http.Request) *appError
type appError struct {
    Error   error
    Message string
    Code    int
}
func (fn appHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")
    if err := fn(w, r); err != nil {
        http.Error(w, err.Message, err.Code)
    }
}
```

Finally, the **images** function below, parses and validates the query parameters, used to fetch the granules with BigQuery in **query.go**. The granule ids are converted into a JSON array with the **encoding/json** package and the result is written to the **http.ResponseWriter**. Any parsing, validation or query errors are handled.

```go
func images(w http.ResponseWriter, r *http.Request) *appError {
    ... // parse form and validate data
    links, err := getLinks(lat, lng, projectID, r) // error handling removed for brevity
    if err := json.NewEncoder(w).Encode(links); err != nil {
        return &appError{err, "Unable to map JSON to response", http.StatusInternalServerError}
    }
}
```

4 Bonus Exercise: Geolocation

See **"converter.go"** for the implementation of the backend system that deals with human-like locations, by using the Google Geocoding API to convert an address into a pair of latitude and longitude coordinates.

HTTP GET Request Format: `https://tvao-178408.appspot.com/images?address=...`