Spatial data and RasDaMan

Fernando Mendes Stefanini — ULB Student Id: 000470635
Evgeny Pozdeev — ULB Student Id: 000473391

Advanced Databases: INFO-H-415

Prof. Esteban Zimányi

December, 2018

Introduction

Motivation

Satellite imagery is often considered to be “Big Data”. For instance, NASA has more than 1100 active satellites in the space right now, one of them is the Landsat 8 mission which collects around 1TB of data every day, this data needs to be properly stored and preprocessed. The other example is Digital Globe: it is the largest satellite data providers, it collects more than 70 TB of raw data every single day. All this data needs to be stored and processed in a proper way.

Spatial DBMS is one of the tools to store and process georeferenced data. This research is motivated to investigate one of the widely used spatial array DBMS RasDaMan (Raster Data Manager) by implementing several common operations on geospatial data and benchmarking its execution time and compare it with PostGIS, the software that allows working with georeferenced data on PostgreSQL.

Brief intro to geospatial data

Geospatial data is the data with georeferencing. There are 2 main types of geospatial data: raster and vector.

The first one is represented by a multidimensional array, each pixel of which has geographical reference to the grid and value or tuple of values (e.g. the intensity of the measured solar radiation). Satellites that scan the Earth or any other planet or telescopes that captures the sky retrieve the data in raster format. Examples of raster geospatial data formats are GeoTIFF and NetCDF.

The second type, vector data, is a set of objects (or topologies), defined by geographical coordinates and other parameters (like time, id, name, etc.). This is the kind of data we use when driving with a GPS or ordering an Uber. Example of vector geospatial data are GeoJSON and WKT.

This research focuses on storing and processing raster geospatial data.

Rasdaman

General overview

RasDaMan ("raster data manager") is one of the leading Spatial Array DBMS. It allows storing and querying massive multi-dimensional arrays, such as sensor, image, simulation and statistics data appearing in domains like earth, space, and life science. Existing databases

implemented with RasDaMan exceed hundreds of Terabytes and are going towards Petabyte. This worldwide leading array analytics engine claims to distinguish itself by its flexibility, performance, and scalability.

The first prototype of RasDaMan was created in the 1990s at the TU Munich on top of the O2 object-oriented DBMS under the direction of Peter Baumann. After that RasDaMan was developed at Jacobs University and also developed the a enterprise (paid) version.

Installation guide

Rasdaman provides an instrument for installing. This script requires package unzip, which can be installed with the command:

```
sudo apt-get install unzip
```

In order to get the installation script next command should be launched:

```
wget http://download.rasdaman.org/installer/install.sh
```

Then a config file should be created, the default version could be found on the official web page (http://download.rasdaman.org/installer/rasdaman-installer/profiles/installer/default.toml). To run the installation next command should be launched:

```
bash install.sh -j default.toml
```

Then the only thing to do is to follow the instructions from the installation run. The complete guide can be accessed under *http://www.rasdaman.org/wiki/Installer.*

RasQL

RasQL (RasDaMan Query Language) is the SQL-style query language that allows users to flexibly build their own data product in a "mix and match" style. The underlying engine boosts performance through strong optimizations, parallelization, and modern hardware. This innovation has effectively pioneered the field of Array Databases, and is the blueprint for several Big Data standards.

General query structure follows the pattern:

```
select resultList
from collName [ as collIterator ]
[ , collName [ as collIterator ] ] ...
[ where booleanExp ]
```

This syntax is really close to the SQL-like one indeed, but only with simple queries. Whenever there is a need to process information in a non-trivial way (as opposed of just outputting simple results) then more complicated, more functional-like syntax structures appear. For example, the following query will take a sales table and consolidates it from days to week per product:

```
select marray tab in [ 0:sdom(s)[0].hi/7, sdom(s)[1] ]
over day in [ 0:6 ]
values condense +
using s[ day[0] + tab7 ] , tab[1] ]
from salestable as s
```

Model of data representation

An array (also called multidimensional array, multidimensional data, MDD) is a set of cells where each of them is tied to a point in the multidimensional space defined by its coordinates.

A cell has a type: it can be a primitive value (e.g. Boolean, Integer) or a complex structure (e.g. a list of Integers, tuple). Coordinates of a cell are discretized which means that they can be represented only with a set of integers. The number of integers needed to represent cell's coordinates is called dimension or dimensionality. Each dimension also has its upper bound and lower bound: the highest and lowest value available, upper and lower bounds for all dimensions define a spatial domain.

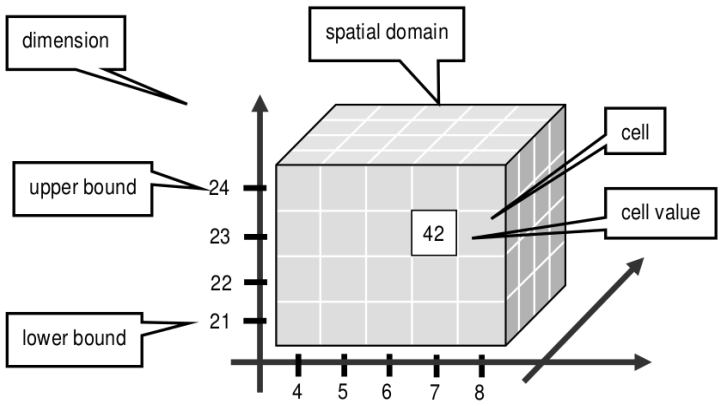

Image 1: RasDaMan data representation (source: http://doc.rasdaman.org/04_ql-guide.html#introduction)

Petascope

Since importing data into the database is one of the most challenging steps in geospatial data analysis (so many things to be aware of: format, georeferences including projection, CRS, etc.), RasDaMan provides an agile tool for importing data – Petascope. In order to import the data, the user should create the recipe file in JSON format, where he should specify all attributes and properties. There are 3 main types of recipes: image mosaic, regular time series, and irregular time series.

For importing files use wcst_import.sh script located at `/opt/rasdaman/bin/` (also might be in system vars and called just by `wcst_import.sh`). This command needs to have a recipe ingredient as a parameter: (source: http://rasdaman.org/wiki/WCSTImportGuide)

```
wcst_import.sh -j path/to/my_ingredient.json
```

Example of a recipe ingredient:

```
{
  "config": {
    "service_url": "http://localhost:8080/rasdaman/ows",
    "tmp_directory": "/tmp/rasdaman/wcst_import",
    "default_crs": "http://opengis.net/def/OGC/0/Index2D",
    "mock": false,
    "automated": false
  },
  "input": {
    "coverage_id": "mos_B4",
    "paths": [
      "/home/rasuser/data/mosaic/b4/*"
    ]
  },
  "recipe": {
    "name": "map_mosaic",
    "options": {
      "tiling": "ALIGNED [0:500, 0:500]"
    }
  }
}
```

Data

Landsat 8 (GeoTIFF)

Landsat programme is one of the most durable remote sensing projects: right now the programme in 40 years old. The first satellite was launched in 1972, the last one - Landsat 8 - was launched on the 11th of February, 2013 with the joint participation of NASA and USGS.

Landsat 8 retrieves images in 11 spectral bands (channels) with a spatial resolution of 30 meters per pixel for the whole Earth within 16 days interval. Landsat 8 data is publicly available.

Spectral Band	Wavelength (micrometers)	Resolution (square meters)
1 - Coastal / Aerosol	0.43 - 0.45	30
2 - Blue	0.45 - 0.51	30
3 - Green	0.53 - 0.59	30
4 - Red	0.64 - 0.67	30
5 - Near Infrared	0.85 - 0.88	30
6 - Short Wavelength Infrared	1.57 - 1.65	30
7 - Short Wavelength Infrared	2.11 - 2.29	30
8 - Panchromatic	0.50 - 0.68	15
9 - Cirrus	1.36 - 1.38	30
10 - Long Wavelength Infrared	10.60 - 11.19	30
11 - Long Wavelength Infrared	11.50 - 12.51	30

Table 1: Description of Landsat 8 spectral bands

A scene is a single satellite image defined by the geospatial reference and time it was taken. Every pixel of a scene represents the intensity of the measured solar radiation for a given area (30 meters in the case of Landsat 8) and the spectral range.

A single Landsat 8 scene is a collection of 11 images (one image per one spectral band/channel) and it has a resolution of 8191×8271 pixels, which is around 1 Gb in zip format. For a better understanding of a scene size, there is an example of Landsat 8 scene of North-West of Europe, which covers North-Eastern part of France and South-Western part of Belgium.

Image 2: example of a landsat 8 image range over a map

The red rectangle on the picture below represents the region of interest - Dataset #1 (smaller one), which was used in this paper for conducting experiments.

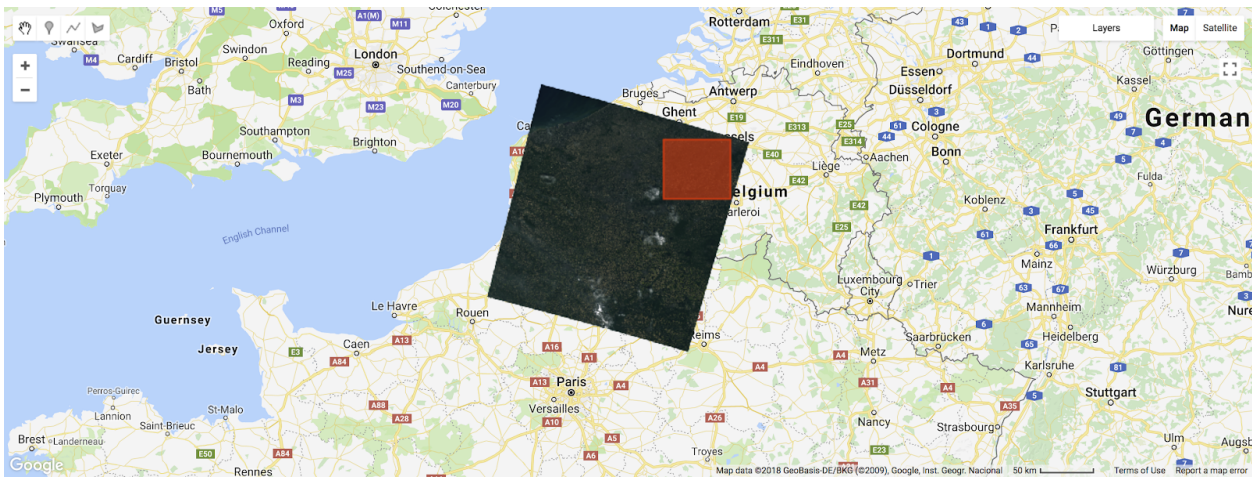

Image 3: Example of different granularities for image size in landsat 8.

This picture below shows the region of interest with greater scale. The bounding box for the region is (3.5244, 50.4388, 4.3708, 50.9133), every image in this dataset is around 13 MB, the whole dataset (9 images and 3 bands) is 361 MB zipped.

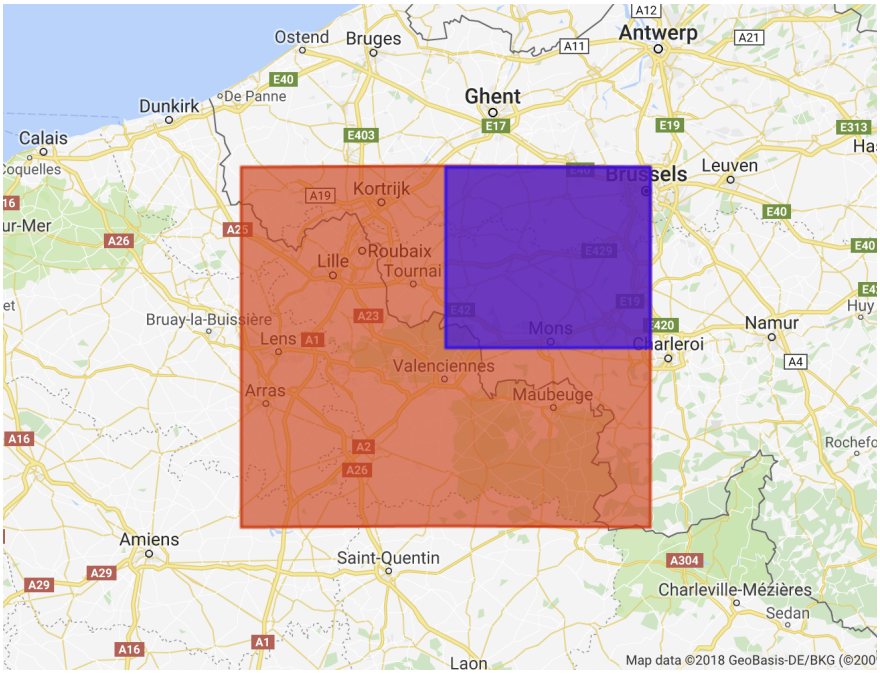

Image 4: visualization of big and small datasets used in experiments

The Dataset #2 (the bigger one) used in this paper is 4 times bigger than the first one, it matches with Dataset #1 on the top right corner (blue square is the previous region of interest), the lower left corner is twice further. The bounding box for Dataset #2 is (2.678, 49.9643, 4.3708, 50.9133), every image in this dataset is around 50 MB, the whole dataset (9 images and 3 bands) is 1,4 GB zipped.

For downloading this data we used Google Earth Engine: it's a cloud-based service by Google made for Geospatial data analysis, which also provides a wide range of data ready to use. We pre-selected images by the region (coordinates of Brussels) and the level of cloud coverage: if there is an overcast, the image itself is hard to analyze visually, so as the results of queries.

Below you can find a table with images description, which includes date and time when the scene was acquired and cloud coverage in percent of scene area.

Scene ID	Date acquired	Scene center time	Cloud coverage
LC81990252013200LGN00	2013-07-19	10:42:23.8687816Z	0.08
LC81990252013248LGN00	2013-09-05	10:42:27.8966224Z	0.15
LC81990252013344LGN00	2013-12-10	10:42:01.6379840Z	0.12
LC81990252014107LGN00	2014-04-17	10:40:21.3221697Z	6.18
LC81990252014139LGN00	2014-05-19	10:39:56.9791017Z	1.16
LC81990252015110LGN00	2015-04-20	10:39:48.2593700Z	4.19
LC81990252015254LGN00	2015-09-11	10:40:19.9055603Z	2.99
LC81990252015270LGN00	2015-09-27	10:40:26.2439051Z	0.24
LC81990252016129LGN00	2016-05-08	10:40:00.6103180Z	0.78

Table 2: image description of dataset used in experiments

GeoTIFF format

Georeferenced TIFF (GeoTIFF) is a public domain metadata format and de facto standard is used as an interchange format for georeferenced raster imagery, it is in wide use in NASA Earth science data systems (source: https://earthdata.nasa.gov/user-resources/standards-and-references/geotiff). It is based on TIFF file format, but also includes all necessary information to work with geospatial data (e.g. map projection, coordinate systems, ellipsoids).

Experiments

While creating shortlist of experiments for testing RasDaMan, we tried to find the most popular use cases of using a database management system for geospatial data processing.

Simple operations

Aggregation (max, min, avg)

Aggregating time series of satellite images is one of the widest spread operations: for example calculation average NDVI (normalized difference vegetation index) for a sown area of the earth's surface over a period of time gives an understanding of how the harvest is ripe.

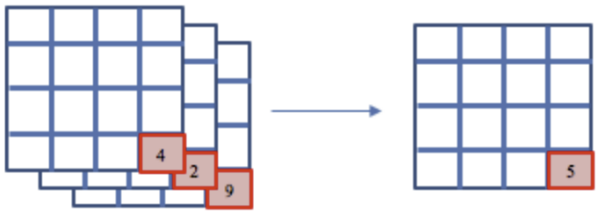

Image 5: example of aggregation using average value.

The peculiarity of the aggregation operation is that the function may vary depending on the purpose of the application: from trivial min, max to more complex standard deviation or weighted average. The tricky part of implementing the aggregation operation is considering geographical coordinates: since the orbit of any satellite is not the same every period, images may vary in terms of bounds. There are 2 common ways to tackle this issue:

1) Saving all information (pic. X, a): the geographical spanning is used to calculate the bounding box for the time series, all scenes are increased to this shape so that the areas where the image does not coincide with the enclosing form are filled with NULL values;

2) Reducing outlying parts (pic. X, b): the geographical spanning is used to calculate the maximum intersection box for the time series is calculated, then the region of the calculated shape is cut out from all the scenes.

Image 6: display of possible dimensions for output of an aggregation operation.

The code snippet below shows how to perform maximum aggregation operation with 9 scenes within a timeseries cube in RasDaMan and Postgis

RasDaMan	PostGIS
`select condense max over x in [0:8] using r[x[0],*:*,*:*] from bel_cube_b3 as r`	`Select ST_NumBands(ST_Union(rast, 'max')) from b3;`

Filter (2D, 3D)

Using a mask for filtering the scenes to get rid of invalid information is also a very popular operation. For example, Landsat 8 data is often provided along with BQA (Band Quality Assurance) - the band, where each pixel represents the encoding for the surface. With BQA any user can understand if there are any kinds of clouds, snow, water for a given pixel/area, and it can be used to filter the original scene from this kind of invalid data.

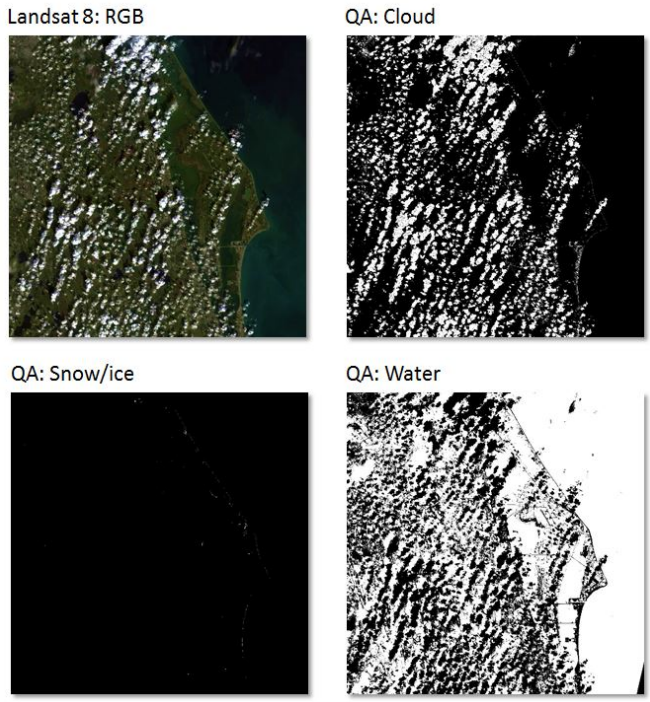

Image 7: an example of applying filtering operation based on BQA (Band Quality Assurance).

The code snippet below shows how to filter an array using itself as a mask and threshold=100 in RasDaMan and Postgis

RasDaMan	PostGIS
`select r*(r > 100) from bel_cube_b3 as r`	`Select ST_MapAlgebra(a.rast, 'case when [rast.val] > 100 then [rast.val] else null end') from b3;`

Complex operations

Multiresolution pyramid

Multiresolution pyramid is one of the approaches to represent graphical raster information, typically used for storing and serving data enhancement. In geo applications multiresolution pyramid is used for displaying a map of a certain resolution, for example, when a user looks at the map with large scale factor, the server sends the map with low resolution (upper levels of the multiresolution pyramid), and when the user increases scale factor, the server sends the map with high resolution (lower levels of the multiresolution pyramid). This

approach server sends only that pieces of information that needs for a moment, and there is no situation when the client receives information that he doesn't need.

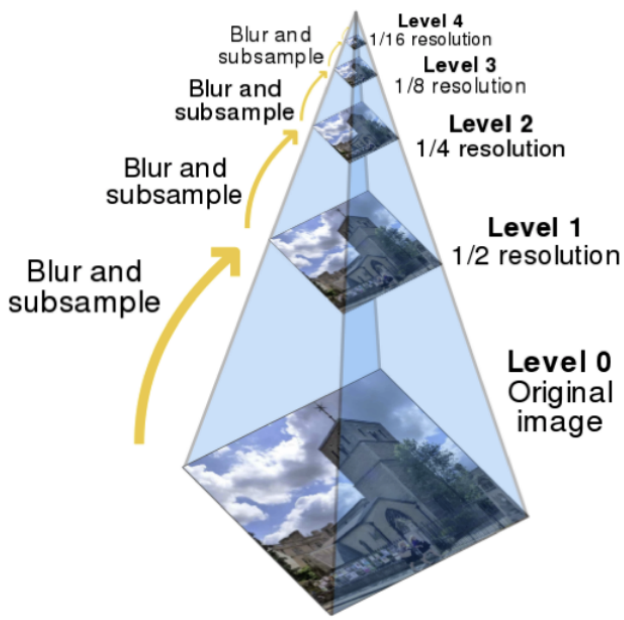

Image 8: 5-level multiresolution pyramid.

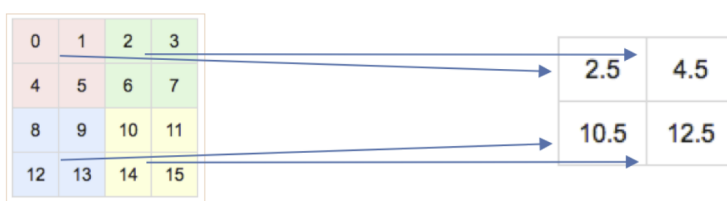

Image 9: computing of the next level raster uses resampling.

There are several ways to compute the pyramid, all of them vary on the number of layers and kernel used for compressing the image (Average, Weighted average, Gaussian kernel, Laplace kernel).

Considered the case of this research, we build 5 level pyramid: we build it from the bottom to the top. The resolution of scale on the given level ***i*** is ***(N/2^i)x(M/2^i).*** For example, the very bottom image has resolution NxM, the second has resolution (N/2)x(M/2), then (N/4)x(M/4), etc. When calculating the pixel value on the level ***i*** we find 4 corresponding pixels on the level ***i-1*** and calculate its average.

The code snippet below shows how to calculate level 2 (scale=0.5) of Multiresolutional pyramid in RasDaMan.

RasDaMan
`select scale(r, 0.5) from bel_cube_b3 as r`

In Postgis Multiresolutional pyramids are calculated during data import time, and hence it's not shown here.

Sobel filter

Edge detection is really essential for dealing with computer vision tasks, and one of the most popular algorithms for that is Sobel filter. This operation requires kernel application for each segment of an original image and storing the result in a separate array. Each pixel in the result array represents the module of the gradient vector. For gradient calculation, it is required to apply 2 kernels: one for horizontal edge detection and another for vertical edge detection.

To calculate the value of the resulting gradient, it is needed to take the square root of the sum of squares of the components according to the parallelogram rule. All values of the result array will be non-negative, which is important when composing an image from the resulting matrix. An important feature of the Sobel filter is the ability to calculate the direction of the face with high accuracy - having the values of the vertical and horizontal vector, you can use the simple formula to get the cosine of the angle between gradient and the normal.

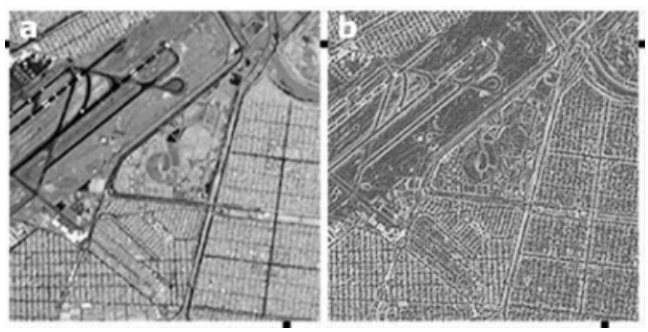

Image 10: example of applying Sobel filter: (a) - original raster, (b) - edges of original image

The code snippet below shows how to calculate the vertical component of Sobel filter in RasDaMan and Postgis

RasDaMan	PostGIS
`select (marray x in sdom(r) values condense + over xy_step in [-1:+1,-1:+1] using (r[x[0]+xy_step[0],x[1]+xy_step[1]] * sobmask_vert[xy_step[0],xy_step[1]]))/9 from bel_cube_b3 as r`	`SELECT ST_MapAlgebra( ST_MapAlgebraFctNgb(rast, 1, null, 3,3, 'sobel_h(float[][], text, text[])'::regprocedure, 'NULL', NULL), ST_MapAlgebraFctNgb(rast, 1,`

	null, 3,3, 'sobel_v(float[][], text, text[])'::regprocedure, 'NULL', NULL), 'sqrt([rast1]*[rast1] + [rast2]*[rast2])') FROM b3 where rid = 1;"

The functions sobel_v and sobel_h for Postgis can be checked in the appendix.

Tiling

Tiling is the operation of dividing large bitmap images into smaller ones for efficient storage and use. Together with the multiresolution pyramid, the tiling is used in almost any rendering engine.

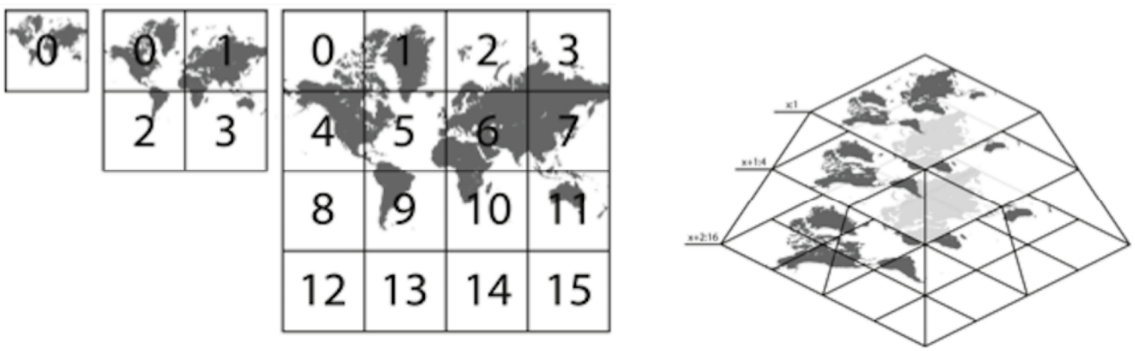

Image 11: tiled map of the Earth with 3 different granularity (on the left). Together with multiresolution pyramid tiling provides a very common approach to serve geospatial data (on the right).

The tiling operation is determined by the input image and the size of the tile - the output images. The amount (*n_images*) of images produced by applying a tiling operation with the T parameter to an NxM size scene is defined by:

$$n\_images = (\left\lceil \frac{N}{T+1} \right\rceil) \times (\left\lceil \frac{M}{T+1} \right\rceil)$$

An all the resulting images will have dimensions TxT, and the resolution of the original image (m / pixels) remains the same. Segments of tiles that go beyond the boundaries of the original scene are filled with *null* values.

The code snippet below shows how to get a tile from an array with spatial domain [0:100,0:100] in RasDaMan and Postgis

RasDaMan	PostGIS
select r[0:100,0:100] from bel_cube_b3 as r	Select ST_Tile(rast, 2, 2, False) from b3;

Infrastructure description

For experiments conduction we used Compute Engine of Google Cloud Platform. Experiments were conducted on 2 separate virtual machines with the same specification: n1-standard-4 (4 vCPUs, 15 GB memory).

Benchmarking results

Data Upload

The following table shows the result of the benchmark over RasDaMan and Postgis when uploading 2D (a single raster) and 3D (a raster time-series) images over the two systems for both sizes of datasets. All the columns are in seconds, and each operation was ran three time and then averaged.

	System under test	run #	1(sec)	2(sec)	3(sec)	avg time (sec)
Dataset #1	RasDaMan	Single image	1.612	1.115	1.404	1.377
		Image time series	11.830	12.157	11.515	11.834
	PostGIS	Single image	1.387	1.378	1.377	1.381
		Image time series	13.386	14.960	16.934	15.093
Dataset #2	RasDaMan	Single image	2.074	1.901	1.864	1.946
		Image time series	46.724	46.410	45.278	46.137
	PostGIS	Single image	4.093	3.885	3.986	3.988
		Image time series	35.835	36.000	45.369	39.068

Table 3: results of data upload benchmark, in seconds.

Operations

Below we have the results when running each of the described operations in RasDaMan and Postgis, each being ran 4 times for each dataset and then averaged. All the results are in seconds. Operations that exceeded the set limit of 30 minutes are marked as so.

		Dataset #1					Dataset #2				
	operation	0(sec)	1(sec)	2(sec)	3(sec)	avg(sec)	0(sec)	1(sec)	2(sec)	3(sec)	avg(sec)
RasDaMan	aggregation min	17.128	17.422	17.244	17.178	17.243	197.265	198.403	209.347	194.069	199.771
	aggregation max	17.125	17.450	17.219	17.124	17.230	200.895	197.165	202.498	201.669	200.557
	aggregation avg	19.118	19.113	19.483	18.933	19.162	226.132	229.577	228.519	224.578	227.202
	filter 2D	0.273	0.327	0.304s	0.313	0.304	1.370	1.392	1.343	1.362	1.367

	filter 3D	30.196	30.846	32.608	30.689	31.085	929.509	763.408	912.028	821.039	856.496
	m-res. pyramid x3	0.202	0.222	0.223	0.223	0.218	0.801	0.863	0.842	0.838	0.836
	m-res. pyramid x5	0.324	0.222	0.253	0.225	0.256	1.120	1.109	1.106	1.120	1.114
	m-res. pyramid x7	0.413	0.389	0.417	0.295	0.379	1.381	1.380	1.325	1.418	1.376
	sobel filter	286.930	286.037	286.315	286.680	286.491	> 30 min	> 30 min	> 30 min	> 30 min	> 30 min
	tiling 2x2	0.344	0.326	0.364	0.303	0.334	1.437	1.609	1.604	1.535	1.546
	tiling 4x4	0.533	0.504	0.574	0.546	0.539	1.857	2.046	1.950	1.986	1.960
	tiling 8x8	1.475	1.518	1.469	1.426	1.472	3.612	3.325	3.336	3.852	3.531
	aggregation min	9.318	9.314	9.315	9.343	9.323	37.374	38.177	38.142	38.374	38.017
	aggregation max	9.336	9.283	9.249	9.268	9.284	37.855	37.598	37.215	37.932	37.650
	aggregation avg	9.396	9.115	9.115	9.126	9.188	37.013	39.316	37.490	37.175	37.749
	filter 2D	9.404	9.429	9.414	9.415	9.416	37.127	41.178	38.231	37.277	38.453
	filter 3D	81.110	80.484	80.376	80.494	80.616	323.203	322.971	323.508	323.321	323.251
	m-res. pyramid	-	-	-	-	-	-	-	-	-	-
	sobel filter	660.660	658.645	662.202	659.053	660.140	> 30 min	> 30 min	> 30 min	> 30 min	> 30 min
	tiling 2x2	9.923	9.893	9.721	9.853	9.848	39.137	40.331	40.186	39.243	39.724
	tiling 4x4	3.940	4.084	3.907	3.865	3.949	15.176	15.157	15.070	15.125	15.132
PostGIS	tiling 8x8	2.359	2.347	2.390	2.408	2.376	10.337	9.106	8.994	8.960	9.349

Table 4: results of operations benchmark, results in seconds unless the operation took more than 30 minutes. In that case it's marked as so.

Conclusion

The goal of this project was to show that RasDaMan is a viable alternative for doing spatial data processing. We performed some common GIS operations on it and also on Postgis in order to have a better understanding of how it performs compared to another platform. In our results it's clear to see that for most operations RasDaMan actually outperforms Postgis. The only set of operations that Postgis does better is for aggregations.

Given that, it can be said that depending on the purpose of your application considering RasDaMan as your DBMS could be worth it. A drawback is that Postgis bases its syntax on SQL queries which are industry standard and well known. On the other hand while for simple queries RasDaMan resembles SQL, more complex operations have a really different semantic that would add the overhead of actually training your users to be able to write it.

Also, although not fully covered in this report, RasDaMan installation referenced here has a higher complexity for most general users as it's a standalone installation. Differently from Postgis again on the fact that installing Postgres and adding Postgis as an extension is a task that more developers should be familiar with.

Appendix

Function declarations for Sobel filter in postgis

```
CREATE OR REPLACE FUNCTION sobel_h(matrix float[][], nodatamode text, variadic args
text[])
   RETURNS float AS
   $$
   DECLARE
       _matrix float[][];
       x_v numeric;
       y_v numeric;
       res float;
   BEGIN
        _matrix := matrix;
       res := 0;
       FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
           FOR y in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
           IF x = 1 THEN
               x_v := 1;
           ELSIF x = 2 THEN
               x_v := 2;
           ELSIF x = 3 THEN
               x_v := 1;
           END IF;
           IF y = 1 THEN
               y_v := 1;
           ELSIF y = 2 THEN
               y_v := 0;
           ELSIF y = 3 THEN
               y_v := -1;
           END IF;
               res := res + (_matrix[x][y] * (x_v * y_v));
           END LOOP;
       END LOOP;
       RETURN res::numeric;
   END;
```

```
   $$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;


CREATE OR REPLACE FUNCTION sobel_v(matrix float[][], nodatamode text, variadic args
text[])
   RETURNS float AS
   $$
   DECLARE
       _matrix float[][];
       x_v numeric;
       y_v numeric;
       res float;
   BEGIN
       _matrix := matrix;
       res := 0;
       FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
           FOR y in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
           IF x = 1 THEN
               x_v := 1;
           ELSIF x = 2 THEN
               x_v := 0;
           ELSIF x = 3 THEN
               x_v := -1;
           END IF;
           IF y = 1 THEN
               y_v := 1;
           ELSIF y = 2 THEN
               y_v := 2;
           ELSIF y = 3 THEN
               y_v := 1;
           END IF;
               res := res + (_matrix[x][y] * (x_v * y_v));
           END LOOP;
       END LOOP;
       RETURN res::numeric;
   END;
   $$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;
```