Towards an infrastructure for interactive Earth Observation data analysis and processing

A. Burger and P. Soille

Contacts: Armin.Burger@jrc.ec.europa.eu
Pierre.Soille@jrc.ec.europa.eu
The Joint Research Centre (JRC)

- JRC is the science service of the European Commission
- JRC provides independent scientific support to EU policy making
- Wide usage of Earth Observation [EO] data as basis for research and policy support
“Earth Observation & Social Sensing
Big Data Pilot Project”

• The EU Copernicus Programme with the Sentinel fleet of satellites acts as a game changer by bringing EO in the Big Data era:
  • expected 10TB/day of free and open data
  • Requires new approaches for data management and processing

• Pilot project launched in January 2015

• Major goal: set up a central infrastructure for storing and processing of Earth Observation and Social Sensing data at JRC
Proposal for a “JRC Earth Observation Data Processing Platform” (JEO-DPP)

• Main focus on satellite image data
• Shall support existing processing workflows and environments (C/C++, Python, Matlab, Java)
• Provide different processing levels:
  • Low-level batch processing
  • High-level interactive processing
• Project timeline:
  • Prototype development: end 2015 – mid 2017
  • Scaling-up in 2017/18:
    JRC Data Centre vs a public cloud solution
**Low-Level Processing**

Translate the prototype programs to an operational tool for large-scale processing - sending jobs through the resource manager.

**Mid-Level Processing**

Create prototype programs for algorithm tests on limited sample data in a dedicated Docker container.

**High-Level Processing**

Interactive programming via a Web interface based on Python with custom API and processing modules.

---

**JEO-DPP processing components**

- **IP[y]: IPython**
- **GDAL**
- **slurm workload manager**

**Processing nodes**

High-speed interconnection with dedicated 10 Gb switch

**Storage nodes**

[Image of a diagram with JEO-DPP processing components and descriptions for low, mid, and high-level processing]
Low-level batch processing

- Running large-scale data processing tasks in a cluster environment
- **Docker containers** for flexible management of processing environments
  - Custom builds for different requirements
  - Facilitates upgrades of processing environment (libraries, tools)
- Run through a workload manager
  - Using SLURM scheduler
  - Usage of MESOS as backend to be evaluated
  
  Advantage: better integration with Docker environment
High-level interactive processing

- Web user interface to server-based data processing
- Based on **IPython Notebook** (Jupyter)
- Development of a data analysis and processing API
  - *Python as core, with C/C++ modules*
  - *Incorporate modules developed by various projects*
- Community building
  - *Sharing expertise and analysis algorithms*
  - *Share and extend existing Notebooks*
IPython/Jupyter interactive analysis
Interactive processing - visualisation

- Visualisation of intermediate results
- Interaction with pre-defined processing and visualization Web services
Jupyter with GDAL-Python API and rendering in desktop client

In [4]: # Create the Shapefile
driver = ogr.GetDriverByName(DriverName)
   if os.path.exists(output):
      driver.DeleteDataSource(output)
   ds = driver.CreateDataSource(output)
   layer = ds.CreateLayer(output, geom_type=ogr.wkbPolygon)

In [5]: # Set up spatial reference systems
   latlong = osr.SpatialReference()
   ortho = osr.SpatialReference()
   latlong.ImportFromProj4('+proj=latlong')

Out[5]: 0

In [6]: # For each grid point, reproject to ortho centered on itself,
   # buffer by x kilometers, reproject back to latlong,
   # and output the latlong ellipse to shapefile
   startx = 188 + resx / 2
   starty = 90 + resy
   for x in np.arange(startx, 188, resx):
      for y in np.arange(starty, 90, resy):
         f = ogr.Feature(feature_def=layer.GetLayerDefn())
         wkt = 'POINT(%f %f)' % (x, y)
         p = ogr.CreateGeometryFromWkt(wkt)
         p.AssignSpatialReference(latlong)
         proj = '+proj=ortho +lon_0=%f +lat_0=%f' % (x, y)
         ortho.ImportFromProj4(proj)
         p.TransformTo(ortho)
         b = p.Buffer(rad)  # radius
         b.AssignSpatialReference(ortho)
         b.TransformTo(latlong)
         f.SetGeometryDirectly(b)
         layer.CreateFeature(f)
         f.Destroy()
   ds.Destroy()
Interactive Processing – set-up and interfacing

- User Notebooks running in Docker containers
  - Separating user environments
  - Allows for detailed resource allocation
  - Web interface for managing user Notebooks
  - Reverse proxy set-up for accessing Docker instances

- Interaction with workload manager
  - Launching batch processing from Notebooks
    => running large-scale processing from your browser
Web services for everyone?

- Public or restricted?
- Data view services via standard protocols
- Web processing services
  - *Embeddable in Web applications or called from processing scripts*
  - *For example for*
    - Image sub-setting: areas of interest, band combination,
    - Image compositing
    - Cloud-free image mosaicking
    - Atmospheric correction
    - ...
Using public Cloud solutions

• Shall be evaluated for the scaling-up phase
  • *JRC Data Centre vs Public Cloud*
  • *Possible scenarios with mixed environments*
    *JRC <-> Public Cloud*

• Interfacing between local and public cloud infrastructure
  • *Docker containers for portability of processing environment*
  • *Seamless distributed processing*
  • *Issue is the availability of input data*
  • *Distributed file system for data sharing*

• Location of processing transparent for users
Thank you for your attention!