New implementation of OGC Web Processing Service in Python programming language. PyWPS-4 and issues we are facing with processing of large raster data using OGC WPS

Jáchym Čepický [a], Luís Moreira de Sousa [b]
[a] OpenGeoLabs s.r.o., Prague, Czech Republic – jachym.cepicky@opengeolabs.cz
[b] Swiss Federal Institute of Aquatic Science and Technology – EAWAG, Überlandstrasse 133 Postfach 611, 8600 Dübendorf Switzerland, luis.desousa@eawag.ch

Commission VII, SpS10 - FOSS4G: FOSS4G Session (coorganized with OSGeo)

KEY WORDS: OGC Web Processing Service, OGC WPS, PyWPS, Open Source Software, Free Software, FOSS4G, OSGeo, Python, Server

ABSTRACT:

The OGC® Web Processing Service (WPS) Interface Standard provides rules for standardizing inputs and outputs (requests and responses) for geospatial processing services, such as polygon overlay. The standard also defines how a client can request the execution of a process, and how the output from the process is handled. It defines an interface that facilitates publishing of geospatial processes and client discovery of processes and and binding to those processes into workflows. Data required by a WPS can be delivered across a network or they can be available at a server.
PyWPS was one of the first implementations of OGC WPS on the server side. It is written in the Python programming language and it tries to connect to all existing tools for geospatial data analysis, available on the Python platform. During the last two years, the PyWPS development team has written a new version (called PyWPS-4) completely from scratch.
The analysis of large raster datasets poses several technical issues in implementing the WPS standard. The data format has to be defined and validated on the server side and binary data have to be encoded using some numeric representation. Pulling raster data from remote servers introduces security risks, in addition, running several processes in parallel has to be possible, so that system resources are used efficiently while preserving security. Here we discuss these topics and illustrate some of the solutions adopted within the PyWPS implementation.

1. INTRODUCTION

PyWPS (PyWPS, 2016, Cepicky, Becci 2007) is a server side implementation of the OGC Web Processing Service standard (OGC WPS, 2007) in the Python programming language. PyWPS started in 2006 as a student project supported by German DBU foundation (DBU, 2016) and it has grown into a world wide used project providing for WPS implementation.

PyWPS itself does not provide any processing functionality. It should be understood as an empty envelope, offering WPS input and output interfaces. A typical user of PyWPS is a system administrator or scientist who needs/wants to expose his/her (geospatial) calculations (so called *processes*) to the world wide web (either with unlimited access or to a certain identified group of users).

The server can offer a theoretically unlimited number of processes. Each process should represent a unique operation on the input data and provide defined output.

2. OGC WEB PROCESSING SERVICE

The OGC Web Processing service (OGC WPS, 2007) belongs to the set of so called OGC Open Web Services, together with Web Mapping Service (WMS), Web Feature Service (WFS) and WCS (Web Coveradge Service). The standard provides 3 types of requests: *GetCapabilities*, *DescribeProcess and Execute.* The communication between server and client is based on an XML communication protocol. It is a *stateless* communication protocol – each request is unique and should have no relationship to other requests. The processes on the server should run as much as possible in separated environments.

2.1 *GetCapabilities* request/response

The *GetCapabilies* request provides *metadata* about the server deployment, server provider and contact person. It also gives a *list of processes available* on the server – their titles, parameters and abstracts. The client should be able to choose the appropriate process with this information.

2.2 *DescribeProcess* request/response

The *DescribeProcess* request (and *ProcessDescription* response) provides the necessary schema description of the desired process. Each process should describe itself (using XML encoding) not only by its title and abstract, but also by presenting a list of required inputs and resulting outputs. Input and output data can be of three different types: *LiteralValue*, *ComplexValue* and *BoundingBox*.

The *LiteralValue* type represents string data directly sent to the server. As for metadata, the client can specify units used and the atomic data type (integer, float, string, date, time, datetime, boolean and others).

ComplexValue parameters represent large (and possibly binary) datasets; usually these are geospatial raster or vector datasets. The server specifies an XML schema and Mimetypes, which it is able to accept – the client application should follow these rules. The vector data are usually encoded with GML (OGC GML, 2012). However, GeoJSON (GeoJSON, 2016) is growing highly popular; (compressed) ESRI Shapefile (ESRI Shapefile, 1998) is another option too. Raster data are usually provided using Base64 encoding.

The *BoundingBoxValue* is the definition of left-bottom, right-top corners of an area of interest, using the OGC Web Services Common specification (OGC WSC, 2007). The bounding box

can be specified using some special coordinate reference system.

A special case for the Complex and Literal input values is the so called *Reference* type of input: instead of providing the data directly as part of the *Execute* request, the client sends just an URL reference to the data and the server is responsible for automatically downloading it from the target server. This enables the usage of WFS or WCS services as inputs to WPS process.

Based on the *DescribeProcess* response, the client application is usually able to construct a final *Execute* request and prepare for its outputs.

2.3 *Execute* request/response

The *Execute* request commands a particular process to be executed, sending the input data to the server (or their reference), along with the process identifier. The Server is then able to accept the process and send the response to the client.

There are two modes in which a process can be executed: *synchronous* and *asynchronous*. In the synchronous mode, the server accepts the requesting XML with the input data, passes the input data to the required process, and waits until all calculations are performed; in the end it sends the resulting *ProcessExecuted* XML response.

In the asynchronous mode, the server provides the client with the *ProcessAccepted* response, *immediately* after the request arrives and closes the connection to the client. The client must then check the *process status* on a specified URL, while the server is running the process in the background. When this is done, the final response is constructed and stored on the specified URL and the next time the client checks the process state it detects, that the execution finished successfully its calculations and can download the resulting outputs.

3. PYWPS IMPLEMENTATION OF OGC WPS

As already stated, PyWPS does not contain any processes in it's distribution. The user is prompted to add his/her own code that is deployed on the PyWPS server instance and the server then takes care of getting the data, evaluate them and call the process *execute* method (function).

The essential functions of PyWPS are:

1. Providing the WPS communication bridge.
2. Getting input data, send in the *execute* request.
3. Preparing a container for the process instance.
4. Calling the *execute* function of the process, communicating with the process during its calculation, providing progress reports and logging various process states.
5. Storing the data to the target location and deleting the created container when the process is finished.
6. Creating and communicating the resulting report to the client (either directly or via predefined way).

From the user's perspective, the focus is on the input and output definition and the *execute* function, that takes already validated inputs, performs the calculation and sets the outputs. The rest is done by PyWPS.

While no process are distributed with PyWPS, the development has focused on providing the best support for external tools, which can be used with geospatial operations. GRASS GIS (GRASS Development Team, 2015) has always been one of the most important external tools supported by PyWPS, as well as R (R Development Core Team, 2016). Beyond these, more common modules from Python, like JSON, SQLconnections, NumPy and other modules are encouraged to be used.

4. NEW VERSION OF PYWPS – PYWPS-4

PyWPS-4 is a completely new version of PyWPS. The PyWPS development team decided to rewrite the code from scratch based on new the knowledge we gained during the years of development.

4.1 MotivationPyWPS is now almost a decade old and the world of the Python programming language and its geospatial bindings has changed in the meantime. The main technical reasons are: the new Python version 3 , native Python bindings to existing projects (like GRASS GIS), new popular input/output formats (GeoJSON, KML, TopoJSON, …).

We have also decided to change the licence from GNU/GPL (Free Software Foundation, 2007) to MIT (MIT License, 2016). The reason for this is to have less restrictive rules for the processes. With the PyWPS-3 version, all code used for the process had to be released under terms of GNU/GPL. With PyWPS-4, the conditions are less viral.

4.2 Implementation

4.2.1 Fundaments

PyWPS-4 is implemented using the Flask microframework (Flask, 2016). We also apply test-driven development (TDD) and since beginning, are using OGC Cite tests (OGC Compliance Program, 2007) in order to make sure PyWPS will be certified by the OGC once a first stable version is released.

PyWPS-4 being is developed using Python version 3, with backwards support for Python 2.7.

To handle XML files, the lxml library is used (Lxml, 2016). This is a C-based XML parser, which provides great flexibility and speed regarding serialization and deserialization of XML files; on the other hand, we are now limited to the official Python interpreter and can not use some of the advantages of PyPy (PyPy, 2016) or Jython (Jython, 2016). We hope, this will be possible in the future.

For some data types, especially the bounding box type, OWSLib structures (Tom Kralidis, 2016) are used.

4.2.2 Validating data inputs

While previous version of PyWPS did not validate any types of inputs, it just took care about the input file size, PyWPS-4 can have up to 4-level (modes) based input data validation.

Validation of *ComplexData* (raster and vector files) is based on 4 steps:

1. *None validation* - the data are always considered valid.
2. *Simple validation* -uses just the mimeType for validation.

3. *Strict validation* - tries to open the file using the GDAL (GDAL, 2016) library and compare used driver to declared *mimeType* file type.
4. *Very strict validation* - validates input data against a given XML schema.

For *LiteralData* (strings, numbers, list of allowed values etc.) only two modes are available:

1. *None validation* - only the data type is checked (input string is converted to integer and so on).
2. *Simple validation* - a stack of conditions is applied to make sure the input data are really what they declare to be.

Users of PyWPS can set custom validating functions, to be applied on each allowed input data formats. PyWPS is distributed with detailed validating functions for ESRI Shapefile, GeoJSON, GML and GeoTIFF formats, as well as most common literal data types.

This makes PyWPS-4 saferwhen dealing with various input data from unknown sources.

4.2.3 Containerising of processes

A WPS server should be able to run multiple concurrent processes in parallel. They are not allowed to share any resources or data with each other. The WPS standard itself provides the possibility to reject incoming *Execute* requests, if all available resources are consumed by previously running processes.

PyWPS creates a temporary directory for every *Execute* request where process calculations take place. When the calculation is finished, resulting the data are moved to the publishing directory and the temporary directory (container) is deleted.

This procedure is to be modified in following versions of PyWPS, so that a safer approach is employed (probably using Docker, vagrant or other more insulated environment).

4.2.4 Asynchronous execution of processes

PyWPS can run up to a determined number of processes in parallel and save another number (both are configurable) of processes in a queue, to later start their execution once new slots are available.
For processes running in the background, the Python *multiprocessing* module is used – this makes it possible to use PyWPS on the Windows operating system too. In previous versions, where the *os.fork()* method was used, this was not possible..
To facilitate the management of concurrent processes , process metadata are stored into a local database. This database is used for logging , saving waiting *Execute* requests in the queue and invoking them later on.
This database will also enable the implementation of pausing, releasing and deleting running process These features will allow PyWPS to comply with WPS version 2.0.0.

4.3 Future work

Future work will be focused ion the implementation of WPS 2.0.0 features, support for non-English languages, better process containerising, an administrative web interface, and the implementation of external services for output data publishing.

We are also focusing on non-technical aspects of project development, such as setting up a documentation environment (user and developed oriented). A Project Steering Committe was also formed around PyWPS – a decision making body required to pass the incubation process of the Open Source Geospatial Foundation (OSGeo). This also makes the project more mature and sustainable for the future.

5. CONCLUSION

The paper introduced a new version of PyWPS – PyWPS-4 – and technical details regarding its development. PyWPS-4 should be more secure, thanks to input data validation, more robust, thanks to the usage of asynchronous process paralellisation. More flexible, thanks to the third party libraries used. Yet, it should remain easy to set up, as it was in previous versions of PyWPS. Thanks to a new MIT license, it should be easy to plug-in to third party systems, with no major legal restrictions.

Future work will bring more security thanks to advanced containerising of running processes, a simple web interface and support for the WPS 2.0.0 standard. Some features of WPS 2.0.0 are getting to PyWPS-4 soon, thanks to current work.

REFERENCES

Cepicky, Becchi, 2007. OSGeo Journal, may 2007. Geospatial Processing via Internet on Remote Servers – PyWPS
PyWPS, 2016. PyWPS project web page, http://pywps.org, (2016-04-16)

OGC WPS, 2007. OpenGIS® Web Processing Service, Open Geospatial Consortium Inc., version 1.0.0, OGC 05-007r7, 2007-06-08, http://opengeospatial.org/standards/wps, 2016-05-16

DBU 2016, Deutsche Bundesstiftung Umwelt, http://dbu.de, 2016-05-16

OGC GML, 2012, OGC® Geography Markup Language (GML), Open Geospatial Consortium Inc., version 3.3.0, OGC 10-129r1, 2012-02-07, http://opengeospatial.org/standards/gml, 2016-05-16

GeoJSON, 2016, GeoJSON data format (web page), http://geojson.org/, 2016-05-16

ESRI Shapefile, 1998, ESRI Shapefile Technical Description, An ESRI White Paper, ESRI, https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf, 2016-05-16

OGC WPS, 2007, OGC Web Services Common Specification, Open Geospatial Consortium Inc., OGC 06-121r3, Version: 1.1.0 with Corrigendum 1, 2007-02-09, http://www.opengeospatial.org/standards/common, 2016-05-16

GRASS Development Team, 2015. Geographic Resources Analysis Support System (GRASS) Software, Version 7.0. Open Source Geospatial Foundation. http://grass.osgeo.org, 2016-05-17

R Development Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2008, http://r-project.org, 2016-04-17

Free Software Foundation, Inc. 2007, GNU GENERAL PUBLIC LICENSE, Version 3, 29 June 2007, http://www.gnu.org/licenses/gpl-3.0.en.html, 2016-04-17.

MIT License, 2016, https://opensource.org/licenses/MIT, 2016-04-17

Flask microframework, 2016, http://flask.pocoo.org, 2016-04-17

OGC Compliance Program, 2007, Open Geospatial Consoritum, http://cite.opengeospatial.org/, 2016-04-17)

Lxml, 2016, lxml - XML and HTML with Python, http://lxml.de, 2016-04-17

PyPy 2016, http://pypy.org/, 2016-04-17

Jython 2016, http://jython.org, 2016-04-17

Tom Kralidis 2016, OWSLib, release 0.11.0, https://geopython.github.io/OWSLib/, 2016-04-17

GDAL 2016, GDAL - Geospatial Data Abstraction Library, http://gdal.org, 2016-04-17