RESTFUL PROXY IMPLEMENTATION FOR BALANCING WPS SERVICES

SILVIU PANICA*

Abstract. Load balancers are widely used to distribute the load from the frontend across multiple backends. These will help a web application or service to scale, manually or automatically, as the workload increases or decreases. In this technical report we present the applicability of such a proxy service in case of legacy services that are not by design scalable. OGC WPS[1] implementations suffer from this limitation and their compatibility with the modern processing infrastructures is limited in terms of scalability or high availability. Therefore the adaption of a proxy service will have great impact over the performance. Our solution proposes to deploy a special WPS proxy service as a frontend to easily distribute the load over the WPS resources hosted by the execution environment.

Key words. load balancer, REST web service, WPS, proxy service, distributed WPS.

1. Introduction. This technical report will present an overview of the WPS[1] standard, commonly used in geospatial domain. This overview will outline the drawbacks of the current WPS implementations and in the end we will propose a solution to overcome the limitations and to improve the overall performance of the implementations.

WPS is a standard used in geospatial applications. Its' main usability refers to a common way of expressing the geospatial processing services interaction. In general, a geospatial processing involves several components: an application able to manipulate the satellite images in order to extract geographic information, a set of coordinates that defines the area of interest where the application logic should be applied to and a list of parameters for tuning the application execution. Based on the area of interest a set of satellite images are extracted from a catalog and fed as input for the geospatial application. In the end, the application along with the inputs (the parameters) are scheduled for execution on a processing platform. WPS standard defines this entire workflow. Later on this paper we will present a short survey over the existent implementations.

Load balancers are used to distribute the load (the consumers) across multiple backend resources (the producers). Also these endpoints act as a common entry for the consumers. They will only have to communicate with only one endpoint in order to reach the distributed backends. This service is also known as a proxy. So the proposed solution combines two components, a proxy and a load balancer, to obtain a frontend service that can be used to enhance web services performance, tipically RESTful[9].

WPS is a widely adopted standard but most of the implementations were not up to date with the new processing paradigms. In this report, we focus the discussion on distributed processing with auto-scaling features. This is a major drawback because WPS implementations are not able to use the modern processing infrastructures at its' full potential.

In Table 1.1 we present a list of available WPS implementations. WPS.int and WPS.Net are no longer supported. The most widely deployed implementation is Zoo-Project WPS [2]. All the analyzed WPS implementations doesn't have native support for distributed processing environments. For example, Zoo-Project WPS

*Institute e-Austria Timisoara, Romania and West University of Timisoara, Romania (silviu.panica@e-uvt.ro).

[1]WPS - Web Processing Service, a standard defined by OGC for geospatial related applications

offers a platform able to expose specific geospatial services as WPS services for easily integration with different other geospatial platforms.

Name	WPS version	Distributed	Reference	Language
deegree	1.0.0	No	[3]	Java
WPSint	0.4.0	No	[4]	Java
pyWPS	1.0.0	No	[5]	Python
Zoo Project WPS	1.0.0, 2.0.0	No	[6]	C, Java, Python ...
52 north	1.0.0	No	[7]	Java
WPS.Net	1.0.0	No	[8]	.Net

TABLE 1.1
A list of WPS implementations

Although Zoo-Project WPS has support for a variety of development languages (like Java, C, C++, Python etc.), it is not specialized on running those specific geospatial applications in a distributed environment. A geospatial application is packed (along with its requirements) as a WPS service and executed locally as a normal system application. Although we can spawn as many instances as you need for you simulation, there is no proxy service offered to load the access across the multiple instances.

In the same way, degree, is a comprehensive geospatial solution but when it comes to application execution it supports only simple Java described applications that cam be executed locally without the possibility to link to an external distributed execution system.

In order to overcome this problem we propose to use a frontend proxy that is able schedule, track and execute WPS requests on a pool of WPS backends. Regardless of the WPS implementation we aimed to build a solution that can be applied in any context. The proposed solution is presented in the next section and it called WPS Proxy.

2. The proposed solution. In order to define the WPS proxy we have to analyse what a WPS service implies. WPS standard defines a set of operations that can be called using HTTP requests in order to define the inputs and the outputs that a geospation application needs to execute over a given set of satellite images. The operations described in WPS standard are briefly described in the following paragraphs and outlined in figure 2.1.

GetCapabilities. This operation is called without any arguments and it will return a list of geospatial services enabled on the WPS endpoint. Each service is known as a *Process* and it has an unique name.

DescribeProcess. This operation will return a description of a given *Process*, where all the inputs, outputs and other service specific informations are described. This operation request must specify as an input the name of the *Process*.

Execute. This operation allows the execution of a specific *Process*. The execution part can be triggered in two ways: synchronous and asynchronous. Synchronous execution means that the client must stay connected to the WPS service until the execution ends in order to receive the output while the asynchronous execution mode will start the execution and return a process identifier. In both modes, at the end of

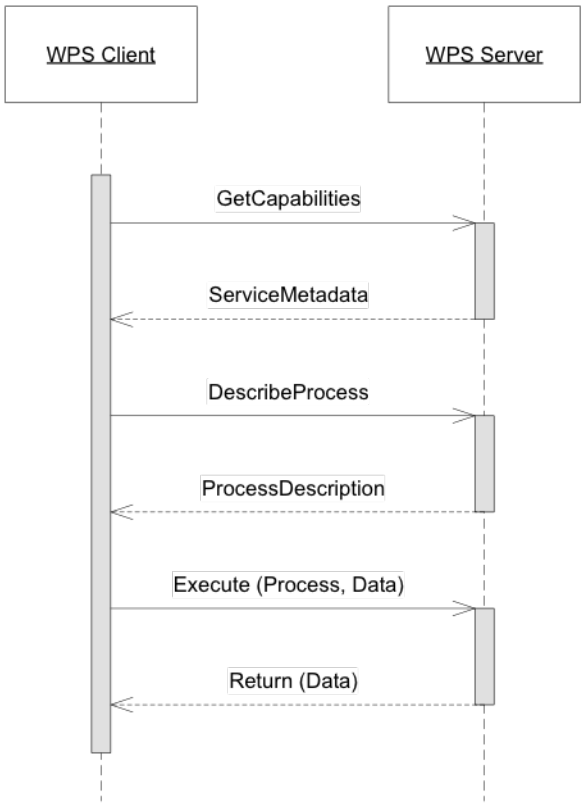

FIG. 2.1. *Common sequence of WPS operations. (source: docs.opengeospatial.org)*

the execution the client can retrieve the output data. This operation request must specify an input the description of the process wanted for execution.

GetStatus. This operation will return the status of the execution of a given process identifier. The request of this operation must supply the process identifier returned by the *Execute* operation called in asynchronous mode.

GetResult. This operation will return the output data generated by the execution of the process. The request of this operation must contain the process identifier.

All the operations requests and responses are described using XML documents, based on the WPS standard specifications. The WPS Proxy must be able to parse and mangle the requests and responses in some specific cases and to balance the clients load over the available WPS resources.

The WPS Proxy architecture is depicted in Figure 2.2 and each component is explained in the following paragraphs.

WPS Pool. This service represents a group of WPS native resources which can be any of the supported WPS implementations. Moreover each resources must have the geospatial applications already configured like in a normal geospatial computational scenarios. The difference from the basic scenario is that now we have more than one instance o available, hosted on different resources (cloud providers[2], Docker

[2]Cloud providers - infrastructure as a service providers either by using hosted services (Amazon, Rackspace, Flexiant etc.) or hosted cloud stacks (OpenStack, OpenNebula, HP Eucalyptus etc.)

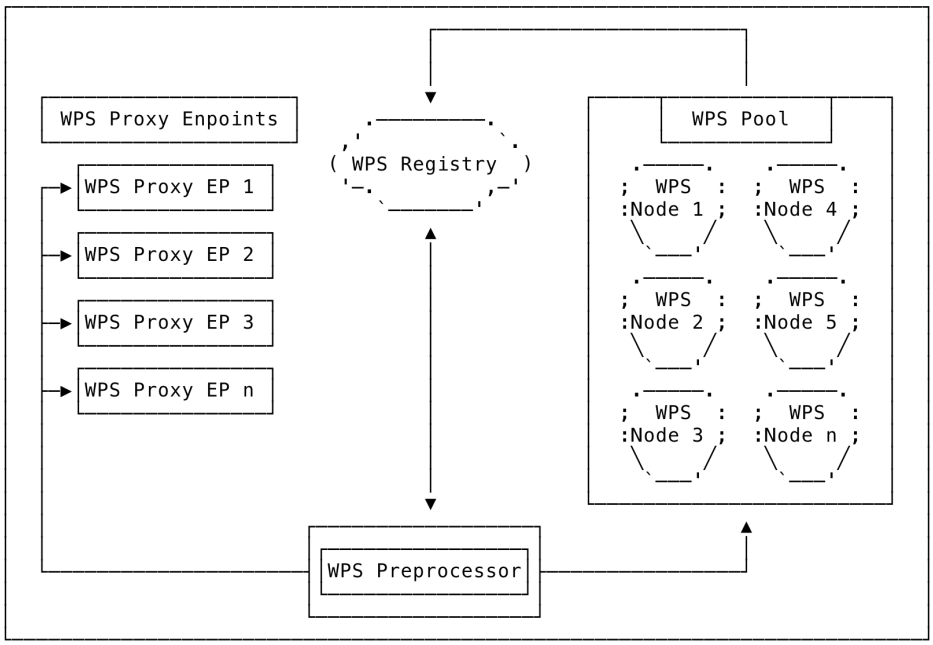

FIG. 2.2. *WPS Proxy Architecture*

conainters[3], etc.) but with only one entry point that is able to balance the WPS requests over these resources. The current version of the WPS Proxy supports only round-robin resource balancing. Also WPS Pool is responsible of mainting a list of available WPS Nodes withing the WPS Pool. This list is stored in the WPS Registry but also available on request using a HTTP call. The workflow of this component is described in Figure 2.3.

WPS Registry. This registry will retain information regarding the WPS processes and the WPS native resources available. It uses a distributed data store to support the instant activation of multiple *WPS Proxy Endpoint* instances in case of heavy workload.

WPS Proxy Endpoint. This component is the entry point for the WPS clients. Each received WPS request is evaluated, mangled and dispatched to a WPS Node for execution. In Figure 2.3 the entire workflow is presented toghether with all the interactions between the components.

WPS Preprocessor. This subcomponent, part of WPS Proxy Endpoint, is in charge with calls managament. The original WPS requests must be dispatched to one of the available WPS resources in the WPS Pool. In case of WPS operations *Execute*, *GetStatus* and *GetResult* the request must contain an operation identifier to query for. When a WPS *Execute* operation is requested, the WPS Node will return an operation identifier. WPS Proxy must map this operation identifier with the designated WPS Node identifier. To achieve this functionality WPS Proxy will respond back to the client with a custom generated operation identifier (proxy operation identifier) that is mapped into the WPS Registry with the original WPS Node iden-

[3]Docker containers - a layer of abstractization and automation of operating system level virtualization [10]

tifier and operation identifier. When a *GetStatus* and *GetResult* request is received, the WPS Proxy will take the request and replace the proxy operation identifier with the original identifier and will dispatch the request to the corresponding WPS Node, where the process is registered. In this way the WPS Preprocessor will ensure that each WPS call is registered correctly to a native WPS Node and the mapping is retain in the registry for further usage. Moreover the WPS Prepocessor will also save into the registry a document with the specifications of the WPS process and the current status of the operation. The workflow of this component is depicted in Figure 2.3.

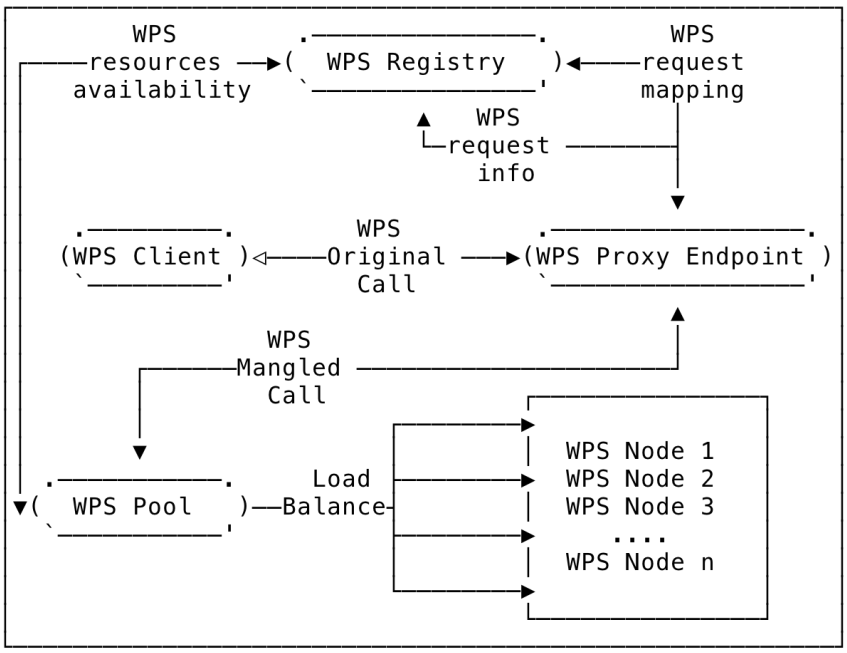

FIG. 2.3. *WPS Proxy Workflow*

The initial tests were conducted on resources booked from VI-SEEM Project through and OpenCall resource request schema. The resources were offered in form of OpenStack[4] virtual machines (VMs). For this testbed scenario we used WPS Zoo-Project implementation and several geospatial applications that are supported by default by Zoo-Project and a set of standard satellite images usefull for the geospatial simulations. This implementation was packed as an OpenStack VM template. This template was used to start a pool of 4 WPS Nodes (hardware template: 4GB RAM, 50GB hard drive and 2 CPUs for each VM). On a different VM we installed the WPS Proxy (together with all the requirements) used as an unique entry point for the WPS resource.

3. Conclusions. The tests of this initial implementation of a distributed WPS, by using the proposed WPS Proxy, revealed a major drawback: the lack of a monitoring system that can be used to automate the resource balancing and possible a dynamic management of the WPS Pool resources (scalability). The WPS Pool is currently static as new resources can be added manually. By using a monitoring system

[4]OpenStack - software stack that offers insfrascture as a service, https://www.openstack.org/

we receive load information over the resources and we can optimize the scheduling phase were instead of using round-robin we can use a more optimal algorithm to better make use of the available resources. More over this monitoring data can be used to detect anomalies in the platform and to take actions to recover the nodes from this possible problems.

4. Future work. We want to extend the current WPS Proxy by enhancing it with three new components. *A monitoring system* able to collect usage data that can be passed to a *anomaly detection system*. This anomaly detector will be in charge with actions to prevent and recover for anomalies (like WPS node failure or WPS node over used because of larger geospatial applications that consume too many resources). Moreover, based on this workload information we want to also introduce a *resource broker* in charge with WPS Pool dynamic management by adding or removing resources based on the workload. By this we ensure that resource consumption is optimal and the system is able to automatically scale to adapt to workload needs.

Acknowledgments. This work is partially supported by the DICE+ romanian funded research project (grant number PN-III-P3-3.6-H2020-2016-0007) and VI-SEEM (https://vi-seem.eu) research project funded through H2020 research and innovation programme (grant number 675121).

REFERENCES

[1] Open Geospatial Consortium - Web Processing Service standard, http://www.opengeospatial.org/standards/wps, as visited in 12.2017.
[2] Gerald Fenoy, Nicolas Bozon, Venkatesh Raghavan, *ZOO-Project: the open WPS platform*, in Applied Geomatics, March 2013, Volume 5, Issue 1, pp 19-24
[3] Degree Geospatial Data Management, http://www.deegree.org/, as visited on 12.2017.
[4] WPSint WPS implementation, http://wpsint.tigris.org/, as visited in 12.2017.
[5] pyWPS - Python WPS Geospatial, http://pywps.org/ , as visited in 12.2017.
[6] Zoo Project - Open WPS Platform, http://www.zoo-project.org/, as visited in 12.2017.
[7] 52 North Web Processing Service, http://52north.org/communities/geoprocessing/wps/index.html, as visited in 12.2017.
[8] WPS.Net - WPS implementation in C#, https://code.google.com/archive/p/wps-net/, as visited in 12.2017.
[9] Fielding, Roy Thomas, *Chapter 5: Representational State Transfer (REST)*. Architectural Styles and the Design of Network-based Software Architectures (Ph.D.), 2000. University of California, Irvine.
[10] Vivek Ratan, *Docker: A Favourite in the DevOps World*, Open Source Forum, February 8, 2017.