WhizPS: An Architecture for Well-conditioned, Scalable Geoprocessing Services Based on the WPS Standard

Marius Laska, Stefan Herle
and Jörg Blankenbach

Geodetic Institute and Chair for Computing
in Civil Engineering & Geo Information Systems,
RWTH Aachen University
52074 Aachen, Germany
Email: marius.laska@gia.rwth-aachen.de

Eric Fichter
and Jérôme Frisch

Institute of Energy Efficiency
and Sustainable Building,
RWTH Aachen University
52074 Aachen, Germany
Email: fichter@e3d.rwth-aachen.de

***Abstract*—Spatial simulations and models are often expert tools which solve a specific spatial problem or model a spatial process. Exposing theses analysis capabilities as a web service is a huge benefit to users of web-based Geographic Information Systems (GISs). The Web Processing Service (WPS) standard was developed to realize these services. In the Geothermal Information System for Potential Studies in Subsurface Soil Layers (GeTIS) project, several complex analysis tools should be exposed as a WPS service and, simultaneously, follow the concept of well-conditioned, scalable services. In this paper, we describe our implemented backend, which can be used to bind expert tools and facade them with the WPS interface. The architecture rests on different communication mechanisms such as Remote Procedure Calls (RPCs) and message queuing as well as geospatial services such as Web Map Service (WMS).**

***Index Terms*—Geoprocessing; Web service; Web Processing Service; Scalability; Geothermal Simulation**

I. INTRODUCTION

The Web Processing Service (WPS) interface was introduced in 2007 by the Open Geospatial Consortium (OGC) for accessing geospatial processing capabilities by HTTP methods. Unlike other important and already established services in the geospatial world, such as Web Map Service (WMS) or Web Feature Service (WFS), the new standard should not just give access to data but enabled processes and models to be requested. But still, the WPS is not heavily used basically because of required advanced knowledge to develop an application as a compliant service and some drawbacks of the underlying protocols.

However, in the Geothermal Information System for Potential Studies in Subsurface Soil Layers (GeTIS) project the WPS interface is a central component of the architecture. The goal of the project is the development of a web-based information system that provides all required data for the regulatory approval and planning of geothermal systems [1]. Currently, these information has to be requested explicitly by the approval authority from different governmental agencies (e.g., geological service, environmental agency, cadastral agency), which is not trivial and inhibits a faster diffusion of geothermal systems. In order to have a single access point, different data sources have to be integrated by standardized service interfaces in such an information system. Apart from that also simulation processes (e.g., for simulating the extent of the temperature plume) are connected to the web portal. However, these simulation tools are developed as expert desktop applications and are not supposed to be deployed as web services. For exposing them as standardized web services, a sophisticated architecture relying on the WPS standard was designed. Based on the following requirements, we implemented a distributed WPS architecture:

- The effort of the service provider for making software accessible and ensure interoperability at the same time should be minimized.
- The system should be loosely coupled, which means that service providers can freely choose on which operating system the task is executed. Furthermore, service providers have the capabilities to run their software in a distributed environment or on their local server. This allows for minimal deployment effort as well as scalable execution of resource dependent tasks.
- The architecture should be built based on open source software and reuse existing and well-known solutions.

This paper describes the distributed architecture developed in the GeTIS project. It starts in Section II with a research about the state-of-the-art of distributed services and especially other WPS-based solutions. After introducing the basic concepts of the WPS interface in Section III, we describe our architectural approach (see Section IV) with the conceptual design and our implementation. Then in Section V, the GeTIS Online Simulation (GOS) as a use case and its integration in our distributed WPS architecture is presented. Finally, we summarize the insights of the implemented system and discuss further developments.

II. State of the art

A WPS interface is used in the geospatial community to provide geospatial analysis algorithms as a service to users. Often, these processes, such as simulations or predictions are very time consuming, which makes it important to run these on appropriate hardware to speed up processing time. Additionally, the implemented algorithms are often expert tools, which are developed by scientist for running on a single machine and are not supposed to be provided as a service. The WPS facades the expert software to facilitate execution and to provide interoperability. The important factor in merging expert tools with a service interface is to implement the WPS with respect to requirements of well-conditioned services.

A *well-conditioned service* is defined as a simple pipeline, which depth is determined by the path through the network and the processing stages of the service [2]. Thus, increasing load also increases the delivered throughput proportionally until the pipeline is full and the throughput saturates. In other words, the service is not allowed to overcommit its resources, otherwise all clients would suffer. The key property of well-conditioned services is *graceful degradation*, which implies that the service maintains a high throughput without a dramatic increase in response time. In concurrent server designs, multiple requests can be accepted and processed at once. The *threaded server design* uses a dispatcher to distribute each incoming request to a separate thread. Each thread processes its request and returns a result to the client. Challenges with long-lasting processes may occur in this setup. With many clients connecting simultaneously, many threads may be active at the same time and context switching may consume large memory and CPU resources. Limiting the number of concurrent clients e.g. by thread pools can tackle this problem. Another approach is the *event-driven server design*. It does not follow the thread-per-connection model, but uses an event loop in a single thread to consume events from an event queue. This main thread processes incoming events and drives the execution of many finite state machines (FSM) with so-called event handlers. Each FSM represents a single request but the complexity is the event scheduler which controls the execution of each FSM [2]. Both architectural models can be used to build highly scalable servers [3], however, the highest scalability and load adaptability can only be accomplished with a distributed approach, since it can easily expand the resource pool.

In the literature, some approaches can be found for distributing WPS request between multiple processing units. In [4] a WPS mediation is implemented to process geospatial data on different computing backends. The job submission software Ganga [5] is used to provide distributed computing in a grid or on a cluster. Performance and scalability were improved successfully. Similarly, other approaches use different distributed computing infrastructure, such as Unicore [6] [7] or forward the WPS request to a Hadoop Cluster [8]. In these solutions, the processes are invoked by sending the input data and the executable of the application to the grid utilizing the job submission tool. The implementations improve calculation performance and service availability enormously. In [9], a spatial computing node based on WPS is designed. In their approach, the utilized spatial data libraries, such as Geotools or GDAL are deployed in distributed machines to process spatial data concurrently and effectively. The single instance uses an appropriate middleware to communicate. Data processing velocity was improved for common spatial processing tasks. In [10] a RabbitMQ queue is used to communicate with a high performance cluster to calculate flooded tiles. The request is computed on the cluster while the result handling is done by the WPS server. They conclude that exploiting supercomputing infrastructures provides scalability and performant processing. Other approaches use middleware software to distribute the requests directly to multiple machines. The WPS remote community module of the GeoServer [11] allows to run requests on one or more remote machines by exposing processes with the WPS protocol. For realizing this, some RPC methods, such as run, progress, complete or kill are implemented utilizing the Extensible Messaging and Presence Protocol (XMPP) protocol for remote commands. Additionally, a remote balancer is included to distribute incoming request based on occupancy of the servers.

III. Web Processing Service (WPS)

Standardized geo web services are used in modern Spatial Data Infrastructures (SDIs) to ensure interoperability. With the OGC WPS version 1.0, a standard for accessing and initiating geospatial processing was introduced in 2007 [12]. The main characteristics of the standard are the introduced rules to define in- and outputs of deployed processes and the different request methods of the service.

The Hypertext Transport Protocol (HTTP)-based WPS follows the request/response messaging pattern and has three core operations: The *GetCapabilities* operation can be used by requesting clients to receive meta-information about the server and its services. The response includes descriptions and the identifier about each process. Detailed information about in- and outputs of each process can be requested by the *DescribeProcess* operation given the process identifier. Finally, the *Execute* operation invokes the process. The user submits the request by posting necessary input parameters to the server. The inputs and outputs can have different data types, such as literals or references to other geo web services.

Processes can be executed in *synchronous* or *asynchronous* mode. In synchronous mode, the server parses the requesting XML, executes the process with respect to the inputs, waits until all calculations are performed and returns the resulting *ProcessExecuted* XML response to the client. In asynchronous mode, after receiving and accepting the request, the server sends a *ProcessAccepted* response immediately to the client. The response contains an URL with can be requested to check the process status while the process runs in the background. When the process finishes successfully, the server creates a final response and stores this document at the specified URL.

The client fetches the result when it requests the given URL the next time.

Since WPS version 1.0 has some drawbacks, the version 2.0 was released in 2014. The introduced features cover for instance improvements in the process descriptions or a *Dismiss* operation to cancel a process.

IV. Architectural approach

A. Conceptual approach of the architecture

Research projects in the GIS domain, such as the GeTIS project often involve the development of expert tools and processes. While rapid prototyping displays a major demand, the deployment of theses processes and accessibility remains a challenge. Implemented processes should be discoverable and exposed as a service via web technologies. This requires that they are accessible using a uniform interface and that their inputs and outputs are clearly formulated. The WPS specification aims at solving this demand. However, existing implementations, such as the PyWPS [13] server lack the ability to bind services and initiate processes remotely. It requires central development and deployment of the processes, which has two major drawbacks: First, development of processes is slowed down since the exposure as a service usually requires adaption and deployment on a web server and, thus, cannot be directly offered on the developer's machine. Second, horizontal scaling is not possible if the WPS server simultaneously represents the single computing back-end. Processing large-scale data or long-lasting processes would influence the responsiveness of the server, which contradicts the well-conditioned service paradigm.

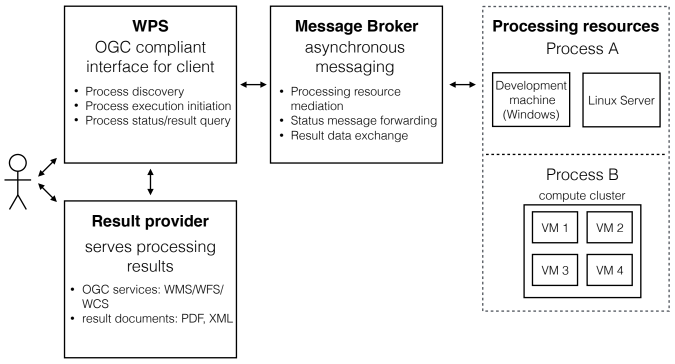
WPS
OGC compliant interface for client
• Process discovery
• Process execution initiation
• Process status/result query
Message Broker
asynchronous messaging
• Processing resource mediation
• Status message forwarding
• Result data exchange
Processing resources
Process A
Development machine (Windows)
Linux Server
Process B
compute cluster
VM 1
VM 2
VM 3
VM 4
Result provider
serves processing results
• OGC services: WMS/WFS/WCS
• result documents: PDF, XML

Fig. 1. High level view of main components of the distributed WPS architecture.

An architecture to meet the requirements of exposing expert tools and to handle the described drawbacks in WPS servers is illustrated in Figure 1. Providing an OGC compliant WPS interface for the client is reasonable to allow for process discovery and initiation in a standardized way. Thereby, the possibly multi-layered back-end and expert processes are masked by the WPS interface. Furthermore, instead of directly processing the requested task on the WPS web server, mediating the request to available computing resources constitutes a much more sophisticated way to process the request. Enabling physical separation between the WPS interface and the processing resources requires some orchestration via asynchronous message exchange. A message broker mediates the task to one of the available processing resources and forwards status messages as well as the final result data to the WPS interface. In order to make both, the status messages and the results, accessible for the client, a result provider is required. Depending on the result data format, it should offer the client direct access to documents, such as PDF or XML files, or provide a OGC compliant service for discovering and accessing spatially referenced data.

B. Applied tools

A PyWPS server (version 4.0.0) builds the foundation for constructing the WPS compliant interface in our architecture. PyWPS is an implementation of the WPS standard from the OGC written in Python. It enables integration, publishing and execution of Python processes via the WPS standard [13]. PyWPS can be deployed with integrated Flask [14] web server, or with an Apache web server. Currently, only the WPS specification 1.0.0 is supported but adaption of the new version 2.0.0 is planned. Geospatial processes can be implemented by extending the *Process* class, which contains a handler and a list of input and output according to the WPS specification. Whenever the PyWPS server receives a new WPS Execute request, it creates a new thread and executes the defined handling method. This limits the ability to distribute processing load over multiple processing machines, since all processes run on the same machine where the PyWPS server is deployed. In order to employ remote processing resources, they have to be addressed via patterns like RPC, which requires asynchronous messaging. In our architecture, a RabbitMQ server is chosen for tackling this issue.

RabbitMQ is the most widely deployed open source message broker. It is lightweight and easy to deploy on premise or in distributed cloud settings and supports multiple messaging protocols. Producers send messages to *exchanges* from which they are forwarded to queues that consumers bind to the specific exchange. This allows for realizing patterns like a worker queue, where multiple consumers listen for messages on the same queue. Furthermore, publish/subscribe patterns can be implemented by using an exchange with *fan-out* characteristics, such that each consumer that wants to receive the messages can bind its own queue to that exchange. In the proposed architecture a RPC like pattern is implemented. Each PyWPS process has its own worker queue. Processing resources for that PyWPS process listen for tasks on that queue. Upon entering the worker queue, an incoming task is fairly dispatched to one of the available resources. In order to allow communication between the PyWPS process and the processing resource, the PyWPS process creates a temporary response queue that the processing resource utilizes for exchanging status message and result data.

In order to use the described RPC pattern, a threaded python implementation using the Pika BlockingConnection [15] has been realized, which will be referred to in the following as PyRPCproducer and PyRPCconsumer. Each PyWPS process has its dedicated PyRPCprocuder. Furthermore, each processing resource operates its own PyRPCconsumer, which can be configured to listen to a specific worker queue and starts a specifiable script. The PyRPCconsumer listens for the console output and forwards messages that contain a specific logging keyword back via the temporary response queue. After having successfully executed the script, all data of a specifiable output folder is encoded and sent back using the temporary response queue. In order to configure a new processing resource, solely the PyRPCconsumer has to be installed and configured, which minimizes the deployment overhead.

C. Workflow

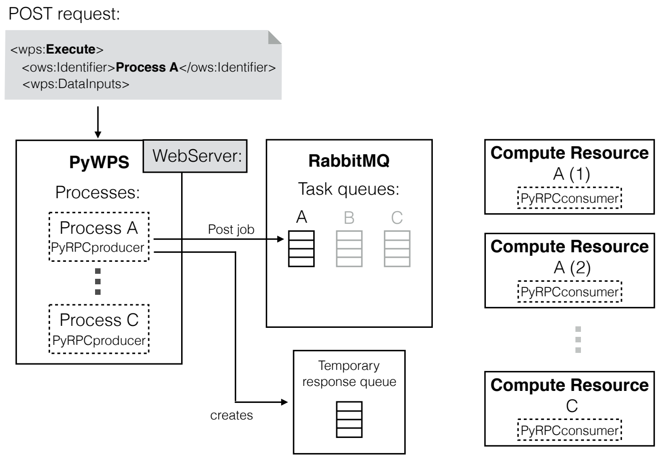

Fig. 2. Workflow after client requests new processing task at the PyWPS server with focus on setup required for task mediation and communication.

The PyWPS server offers multiple processes (according to WPS formulation), which can be discovered and described according to the WPS specifications using the *GetCapabilities* and *DescribeProcess* operations. A process can be started using the *Execute* operation, while specifying the process identifier as well as supplying the process with the specified input data. This request is illustrated by Figure 2. Each of the processes deployed in the PyWPS server uses a dedicated instance of the PyRPCproducer implementation, which was introduced before. The process encodes the input data in JSON format and sends it to the corresponding task queue. In Figure 2, an execute operation for process A is sent to the PyWPS server, such that its PyRPCproducer posts the job to task queue A. Simultaneously, it instantiates a temporary response queue, which will be used by the assigned working machine of the service provider to transmit status messages and the final results of the process. The service provider of process A might be offering multiple physical machines for handling incoming tasks in order to handle increasing processing load. This facilitates horizontal scaling.

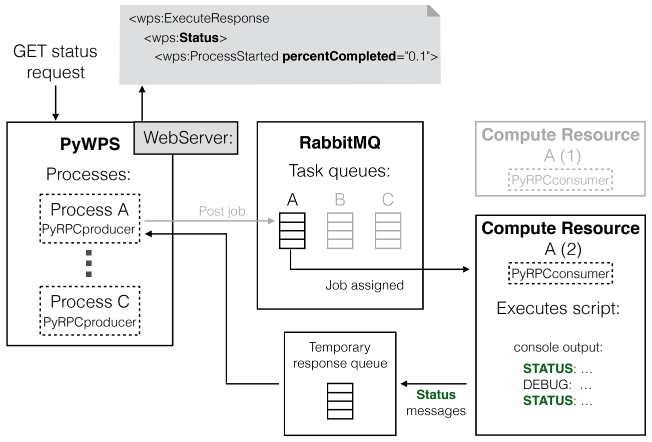

Fig. 3. Workflow after remote processing machine is assigned with emphasis on communication between PyWPS, the computing resource and the requesting client.

Each working machine of a process provider is subscribed to one or multiple task queues, depending on which processes it should handle. Given that computing resources are available, processing requests that arrive at the RabbitMQ broker are immediately forwarded to one of the subscribed clients. This process is illustrated by Figure 3. Worker resource A (2) is assigned to handle the task and starts the processing software. The process' RPC client listens for the console output and forwards messages, which contain specific keywords, such that status updates can be communicated back to the PyWPS server. The messages containing the keyword (e.g., *STATUS*, green in Figure 3), are forwarded via the temporary response queue.

After successful completion of the script, all result documents that are present in a specific folder (*output folder*, green in Figure 4) are encoded and returned via the same temporary response queue.

During the whole processing of a request, the initiator of the *Execute* operation is able to access the status messages of the processes by requesting a resource at PyWPS server. All status messages as well as the results and input files of the processes are served by an integrated Flask web server. PyWPS can also run as WSGI application on an Apache HTTP server. Finally, the results of the process are served back to the process' initiator, either as direct data, or as references to the location where the data can be requested (illustrated by Figure 4).

D. Result handling by PyWPS extension

The implemented PyWPS extension offers various result handling solutions, which are conform to the WPS standard. They are basically divided into direct responses containing the result in plain text, such as an XML document, or responses that contain a reference to a resource combined with its format specification. Typically, geo processes deliver spatially referenced data as result, such as GeoTIFFs (raster data)

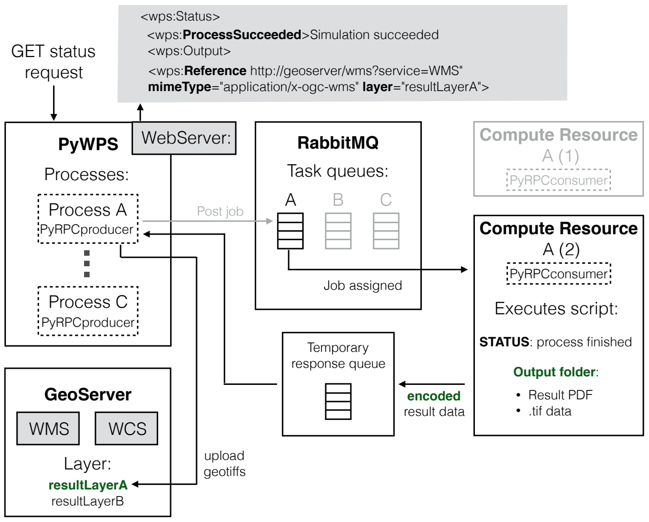

Fig. 4. Workflow of result data provisioning with emphasis on handling spatially referenced data.

or shapefiles (vector data format of ESRI). For convenience reasons, it is beneficial to offer access to these results via a dedicated service. The OGC defines several standards for services and encodings. The WMS is a so-called portrayal service, which delivers georeferenced data as styled images for visualization purposes. Raw georeferenced raster data is accessible by the Web Coverage Service (WCS), while vector data can be provided by the WFS standard. Several implementations of these services exist, among other the GeoServer [16], which is a widely used implementation of an open source server for sharing geospatial data. It provides OGC compliant implementation of the mentioned standards WMS, WFS and WCS. Spatial data, such as GeoTIFFs can be uploaded and function as data source for any of the services.

In our proposed solution, the PyWPS process uploads the spatially referenced result data of the remote process to a GeoServer instance via its REST interface [17]. When a client requests the status of a finished process, the output section of the result XML file links to the WMS *GetCapabilities* request, but also includes the name of the corresponding layer. This enables the client to specifically discover information on how to access the WMS layer that has been generated as result of the process.

Non-spatial data, such as a resulting PDF file are served by the internal web server of the PyWPS instance and are also linked in the outputs section of the status response.

Additionally, for long-lasting processes it might be inappropriate to constantly poll status updates from the web server. Therefore, we implemented a notification mechanism that sends out an email to the initiator of the process as soon as the results of the process are available. It requires a specified email address in WPS execute request. Other notifications mechanisms are conceivable, which could include sending an SMS or a message via any other messaging protocol like XMPP [18].

V. Use Case - Online Geothermal Process

For demonstrating our architecture in a real usage scenario, an example process is described below that was developed within the scope of the GeTIS project.

The GeTIS Online Simulation (GOS) [19] is a transient three-dimensional subsurface simulation that allows to plan geothermal borehole heat exchangers. Using a finite volume approach, it considers heat conduction in the rock as well as convectional heat transport caused by ground water flow. For this purpose, spatial, physical and geological properties of the subsurface are needed. The GOS requires these input data encoded in an XML file. Therefore, the WPS server facades the corresponding process with a single input field for the XML input file and transfers it via the messaging broker to the computing machine on which the GOS runs. The structured and parsed information is allocated to the setup functions building the three-dimensional simulation grid. While solving the mathematical equations for the simulated time span of 50 years, the remaining processing time is communicated to the WPS. The results of the GOS are processed textually and graphically to inform the user. Information about simulation and subsurface conditions as well as plots for time series and spatial field data, e.g., to visualize volume fluxes and performances, are copied into a single PDF. Horizontal section views showing temperature data are processed to GeoTiff files. All result documents are transferred back to the WPS server via the messaging broker. The georeferenced files are uploaded to the GeoServer using the REST interface such that a WMS service can be offered to demonstrate the influence of the heat extraction caused by the borehole heat exchangers on neighboring properties. Among the request parameters of the WMS specification is a so-called elevation dimension, which enables the presentation of geospatial information at different elevations. In the context of the GOS, the elevation dimension is used to allow the requester to browse through the different subsurface levels. The resulting PDF file is stored on a web server and can be accessed by a request specific URL.

Like the GOS simulation, the GeTIS project involves two other processes, which can be invoked by the WPS interface and computed in our architecture. This includes an analytical subsurface model and a building simulator (see [19]).

VI. Conclusion

We proposed the implementation of an architecture to expose geospatial processes as web services. Since a wide range of developed spatial simulations and models can be classified as expert tools, we formulated specific requirements. The architecture facilitates deploying implemented tools with minimal effort, it is loosely coupled with the actual expert tools allowing for hardware independent deployment and, furthermore, it is based on existing open source components. In detail, we implemented an OGC compliant WPS interface based on the PyWPS server with respect to the stated requirements. The RabbitMQ middleware is used to distribute WPS requests based on occupancy to available processing resources. For this, we developed a PyRPCconsumer script that serves as the sole interface for software providers to connect their expert tools such that they can be invoked with standardized WPS request. The WPS implementation is able to handle spatial as well as non-spatial result data by generating standardized geospatial portrayal and data services (e.g., WMS) provided by a GeoServer or by serving documents through a simple web server. The applicability of the proposed architecture has been demonstrated by utilizing it in the GeTIS project to expose several complex analysis tools. For demonstration purposes, the GOS has been described, which simulates the subsurface allowing to plan borehole heat exchangers.

Currently, each WPS request facades a single expert tool, however in the future the architecture could be easily extended so that requests can be split up into sub-tasks, which could be run concurrently on multiple machines to decrease processing time. Furthermore, it is not possible to cancel running processes at the moment, since the underlying PyWPS implementation is based on the WPS 1.0 specification, which lacks of the *Dismiss* operation that has been first introduced in version 2.0. However, our architecture does already support the abortion of processes by sending broadcast messages to all computing resources. If the request is already accepted and invoked, it is cancelled directly. Otherwise, the en-queued and pending request is ignored by the computing machines. As soon as the PyWPS implementation has been upgraded to support WPS 2.0, the abortion of running processes can be effortlessly integrated.

Acknowledgment

The work and implementations described in this paper are conducted in the GeTIS project which is funded by the German Federal Ministry for Economic Affairs and Energy (BMWi).

References

[1] S. Weck-Ponten, R. Becker, S. Herle, J. Blankenbach, J. Frisch, and C. van Treeck, “Automatisierte Datenaggregation zur Einbindung einer dynamischen Gebäudesimulation in ein Geoinformationssystem,” in *Tagungsband der 7. Deutsch-Österreichischen IBPSA-Konferenz BauSIM 2018*, Karlsruhe, 2018, pp. 516–523.

[2] M. Welsh, D. Culler, and E. Brewer, “Seda: An architecture for well-conditioned, scalable internet services,” in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, ser. SOSP ’01. New York, NY, USA: ACM, 2001, pp. 230–243. [Online]. Available: http://doi.acm.org/10.1145/502034.502057

[3] D. Pariag, T. Brecht, A. Harji, P. Buhr, A. Shukla, and D. R. Cheriton, “Comparing the performance of web server architectures,” in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys ’07. New York, NY, USA: ACM, 2007, pp. 231–243. [Online]. Available: http://doi.acm.org/10.1145/1272996.1273021

[4] G. Giuliani, S. Nativi, A. Lehmann, and N. Ray, “WPS mediation: An approach to process geospatial data on different computing backends,” *Computers and Geosciences*, vol. 47, pp. 20–33, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.cageo.2011.10.009

[5] “Ganga,” URL: https://ganga.readthedocs.io/en/latest/ [accessed: 2018-12-13].

[6] “Unicore,” URL: http://www.somewebpage.org/ [accessed: 2018-12-13].

[7] B. Baranski, “Grid computing enabled web processing service,” in *GI-Days, Münster*, 2008, pp. 1–12.

[8] Z. Chen, N. Chen, C. Yang, and L. Di, “Cloud computing enabled web processing service for earth observation data processing,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 6, pp. 1637–1649, Dec 2012.

[9] L. Liu, G. Li, and J. Xie, “Design & implementation of distributed spatial computing node based on WPS,” in *IOP Conference Series: Earth and Environmental Science*, ser. IOP Conference Series: Earth and Environmental Science, vol. 17, 2014, pp. 1–8.

[10] A. Tellez-Arenas, R. Quique, F. Boulahya, G. Le Cozannet, F. Paris, S. Le Roy, F. Dupros, and F. Robida, *Scalable Interactive Platform for Geographic Evaluation of Sea-Level Rise Impact Combining High-Performance Computing and WebGIS Client.* Cham: Springer International Publishing, 2018, pp. 163–175. [Online]. Available: https://doi.org/10.1007/978-3-319-74669-2_12

[11] Open Source Geospatial Foundation, “GeoServer WPS Remote community module,” 2018. [Online]. Available: http://docs.geoserver.org/stable/en/user/community/remote-wps/index.html

[12] P. Schut, “OpenGIS Web Processing Service 1.0.0 [OGC 05-007r7],” Open Geospatial Consortium, Tech. Rep., 2007.

[13] PyWPS Development Team, “Python Web Processing Service (PyWPS),” 2009, URL: http://pywps.org [accessed: 2018-12-13].

[14] “Flask,” URL: http://flask.pocoo.org/ [accessed: 2018-12-13].

[15] “Pika Blocking Connection,” URL: https://pika.readthedocs.io/en/0.10.0/modules/adapters/blocking.html [accessed: 2018-12-13].

[16] “GeoServer,” URL: http://geoserver.org/ [accessed: 2018-12-13].

[17] “GeoServer REST Interface.” [Online]. Available: https://docs.geoserver.org/stable/en/user/rest/index.html

[18] “Extensible Messaging and Presence Protocol (XMPP),” URL: https://xmpp.org/ [accessed: 2018-12-13].

[19] E. Fichter, S. Weck, R. Becker, J. Derksen, S. Düber, J. Frisch, D. Koppmann, R. Löhring, J. Blankenbach, C. van Treeck, and M. Ziegler, “Geothermal Information System for Potential Studies in Subsurface Soil Layers,” in *Proceedings of Building Simulation: 15th Conference of IBPSA*, San Francisco, CA, USA, 2017, pp. 662–671.