Earth Observation Scientific Workflows in a Distributed Computing Environment

TL van Zyl[1*], G McFerren[1], A Vahed[1]

[1]ICT4EO Research Group, Meraka Institute, CSIR, Pretoria, South Africa

* Corresponding Author - tvzyl@csir.co.za

Abstract

Geospatially Enabled Scientific Workflows offer a promising paradigm to facilitate researchers, in the earth observation domain, with many aspects of the scientific process. One such aspect is that of access to distributed earth observation data and computing resources. Earth observation research often utilises large datasets requiring extensive CPU and memory resources in their processing. These resource intensive processes can be chained; the sequence of processes (and their provenance) makes up a scientific workflow. Despite the exponential growth in capacity of desktop computing, resources available on such devices are often insufficient for the scientific workflow processing tasks at hand. By integrating distributed computing capabilities into a geospatially enabled scientific workflow environment, it is possible to provide researchers with a mechanism to overcome the limitations of the desktop computer. The majority of effort in regard to extending scientific workflows with distributed computing capabilities has focused on the web services approach as exemplified by the OGC's Web Processing Service and by GRID computing. The approach to leveraging distributed computing resources described in this paper uses instead remote objects via RPyC and the dynamic properties of the Python programming language. The Vistrails (http://www.vistrails.org) environment has been extended to allow for geospatial processing through the EO4Vistrails package (http://code.google.com/p/eo4vistrails/). In order to allow these geospatial processes to be seamlessly executed on distributed resources such as cloud computing nodes, the Vistrails environment has been extended with both multi-tasking capabilities and distributed processing capabilities.

1 Introduction

The last few decades has seen a legitimisation of *in-silico* science alongside more commonly known and accepted scientific methods. Certain noteworthy characteristics of in-silico scientific endeavour are important to identify. First, scientists have access to vast and increasing datasets, generated by telescopes, physics experiments, sensor networks and remote sensing instruments, for example. Second, modern computers have allowed scientists the latitude to develop complex, long-running and often complex models and simulations which often emit enormous datasets, such as is common in climate models. Thirdly, interconnected networks like the Internet have enhanced scientific collaboration opportunities, necessitating an increased focus on data and model provenance, and have also resulted in more network-centric or distributed deployments of data, processing, storage and metadata services and resources. Fourthly, specialised high-performance computing environments have emerged to support these trends. The large data requirements and the need to perform complex calculations on these data in distributed multi-core environments have made scientists turn to the concept of scientific workflows [Asanovic et al., 2006, Gil et al., 2007, Barker and Van Hemert, 2008, Meglicki, 2001, Ludascher et al., 2006].

Research indicates that scientists can be shielded to an extent from the complexities of these high perfomance environments through the mechanism of scientific workflows, which automate complex processes and provide integrated access to datasets often characterised by their large sizes and distributed locations [Gil et al., 2007, Gibson et al., 2007, Ludascher et al., 2006].

Current approaches to distributed scientific computing in the geospatial domain are almost exclusively Web Services based as found in the use of OGC Web Services [Open Geospatial Consortium, 2010], OpenDAP [OPeNDAP development team, 2011] and, less commonly, GRID computing [Foster et al., 2001, 2003]. A major limitation of these Web Services approaches is that deployments are static, can not easily be modified and require redeployment to enable minor changes. This may be acceptable if the workflows and algorithms to be run upon these web services are predefined. However, not entirely clear is how a scientist would deploy his or her own dynamically and continuously changing algorithms to these infrastructures whilst still exploiting the massive computing power and extensive data handling capabilities available at these resources [Deelman and Chervenak, 2008].

In a scientific workflow computing environment, scientists may introduce entirely new algorithms and continuously make minor alterations to refine and tweak them. In addition, the pipelines connecting algorithms are continuously changing and being altered as the scientist explores alternate mechanisms for proving or disproving their hypotheses. These algorithms and pipelines of processes work upon large data sets that should not be moved around, but rather remain upon the specialised hardware and software infrastructures that support them. In the case of geospatial-temporal processing, moving the data away from the infrastructures that support indexing and rapid data retrieval, to the infrastructure containing a process is not always as viable as moving the process to the data. This speaks to a need for supporting mobile code in a high performance geospatial-temporal processing environment [Deelman and Chervenak, 2008, Barker and Van Hemert, 2008].

Although GRID computing does promise much of the above, the approach can be complicated, requiring the deployment of complex software infrastructure and expert knowledge to use the GRID resources. We describe an approach in this paper that is lighter, intuitive and requires less specialised expertise on the part of the end user. It is true that the approach is somewhat less sophisticated with respect to aspects such as automated scheduling and load balancing; however we expect that this challenge will be overcome in time [Foster et al., 2001, 2003].

The work described in this article couples the scientific workflow value proposition to the use of remote objects and provision of mobile code support enabled by the dynamic nature of the Python programming language. This paper describes the broader architecture and design of the work, some lessons learnt and discussion of various strengths and weaknesses of this particular approach. These lessons and discussions include, for example, discovering limitations of inter-process communication and the implications of the use of the C and C++ programming languages that underpin the performance in many FOSS4G software tools. Figure 1 shows the general overview of this interaction.

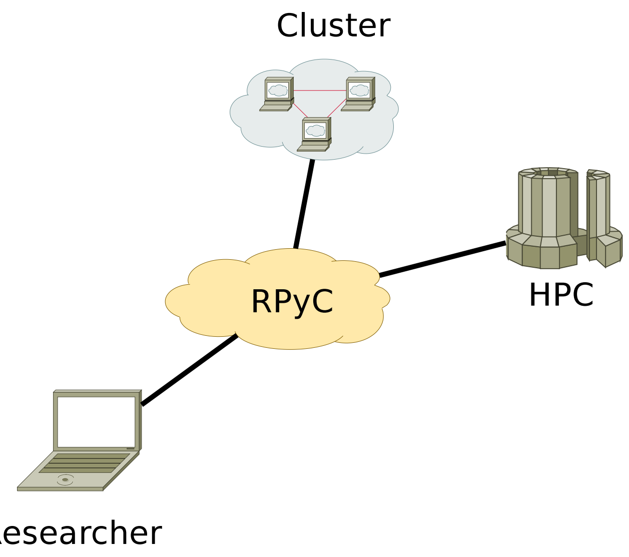


Figure 1: A researcher on his laptop gaining access to a compute cluster and a high performance computer via RPyC

The remainder of the article proceeds as follows: First we present some background on geospatially enabled scientific workflows and remote objects and mobile code in Section 2. Section 3 looks at the design decisions made in building the prototype of our system. Section 4 presents some results of using the prototype within the earth sciences domain together with lessons learnt. Section 5 discusses future work and research directions.

2 Background

In order to discuss the approach to distributed geospatial processing taken within this article, the concepts of geospatially enabled scientific workflows, remote objects and mobile code need to be explored. In the remainder of this section we expand on these ideas and provide the reader with some additional background.

2.1 Geospatially Enabled Scientific Workflows

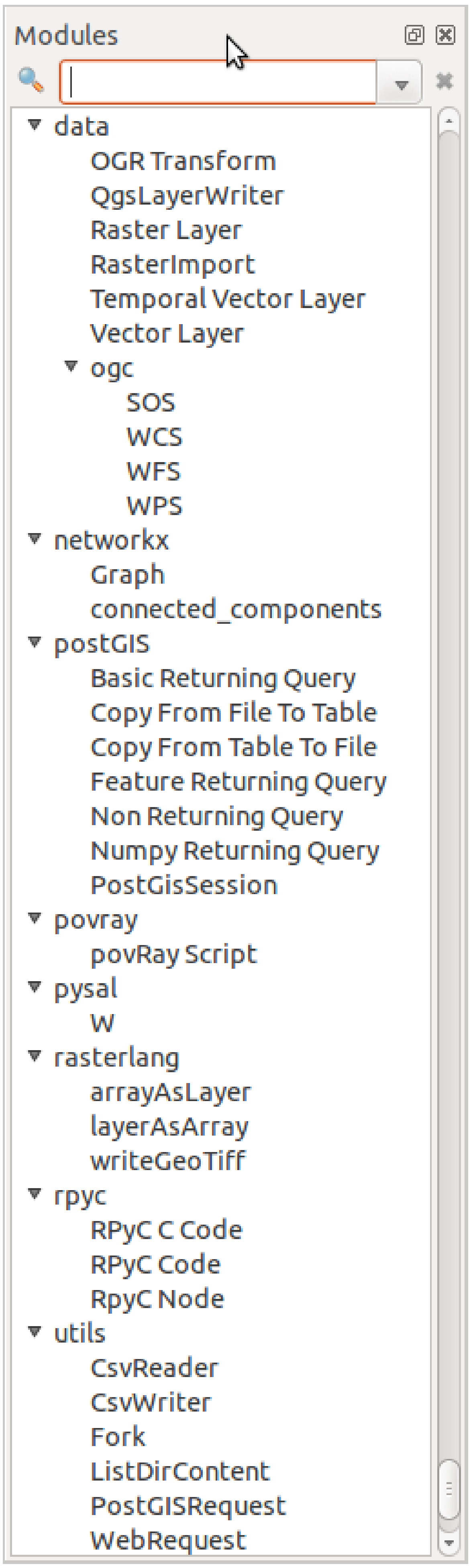
Given the iterative and fast-paced nature of *in-silico* science and often an imperative for researchers to share their results with the broader scientific community and to verify, interrogate and build upon the results of others in the community, an e-toolset is required that can support these tasks. This points to a need for a collaborative environment that allows for proper provenance, reproducibility, extensibility, knowledge sharing and automation of the scientific process. Scientific workflows provides an environment that allows these requirements to be met .

Scientific workflows and workflow environments are used as means for modelling and enacting scientific experiments. They share and leverage many features and techniques of mainstream business workflows and workflow environments, but also offer different or enhanced functionality. Conversely, scientific workflows promise to become an important area of research within workflow and process automation, possibly leading to the development of the next generation of problem-solving and decision-support environments. Scientific workflow environments have capabilities for handling data or transaction intense computation, more human interaction, and a large array of methods and tools to support scientific activities (e.g. visualisation, data transformation). Further, they focus strongly on workflow provenance, capturing rich details of *in-silico* experiments, and place an emphasis on re-execution and comparison [Barker and Van Hemert, 2008, Deelman and Gil, 2006, Deelman and Chervenak, 2008, Howe et al., 2008].

This paper uses the concept of *geospatially enabled scientific workflows* a concept that is tangential to that of *geospatial workflows* and expanded upon in previous work by the authors. We prescribe to the following definition of a geospatially enabled scientific workflow McFerren et al. [2010]:

> “In our conception of geospatially enabled scientific workflow environments, geospatial functionalities and data are intermingled with a variety of other sets of tools, functionalities and data, from, for example, the numerical modelling, computational intelligence, high performance computing and statistical domains.”

EO4Vistrails is an attempt at supporting this concept of geospatially enabled scientific workflows [EO4Vistrails development team, 2011]. The EO4Vistrails package shown in Figure 2 extends the capabilities of the Vistrails scientific workflow environment [Callahan et al., 2006] to support geospatial functionalities and datasources such as OGC Web Services [Open Geospatial Consortium, 2010]. In EO4Vistrails, the geospatial functionality supplied by the QGIS API [Open Source Geospatial Foundation Project, 2010], PostGIS [Refractions Research, 2010] and certain OGC services is being intertwined with a variety of other toolsets, functionalities and data, for example, the numerical analysis library NetworkX, computational intelligence libraries and statistical programs such as R and PySAL [R development team, 2008, PySAL development team, 2011]. EO4Vistrails connects these toolsets with distributed and high performance computing environments.



Modules

- data
  - OGR Transform
  - QgsLayerWriter
  - Raster Layer
  - RasterImport
  - Temporal Vector Layer
  - Vector Layer
  - ogc
    - SOS
    - WCS
    - WFS
    - WPS
- networkx
  - Graph
  - connected_components
- postGIS
  - Basic Returning Query
  - Copy From File To Table
  - Copy From Table To File
  - Feature Returning Query
  - Non Returning Query
  - Numpy Returning Query
  - PostGisSession
- povray
  - povRay Script
- pysal
  - W
- rasterlang
  - arrayAsLayer
  - layerAsArray
  - writeGeoTiff
- rpyc
  - RPyC C Code
  - RPyC Code
  - RpyC Node
- utils
  - CsvReader
  - CsvWriter
  - Fork
  - ListDirContent
  - PostGISRequest
  - WebRequest

Figure 2: Modules from the EO4Vistrails package

In Vistrails the basic unit of workflow composition is the a Vistrails module. A Vistrails module has a set of input ports and output ports. An input port can either take its value from the output port of some other Vistrails module or may be set to a constant value. When a Vistrails module is executed by the workflow engine the values from the input ports are read some processing is done using those inputs and any results are placed on the output ports for the next module. By connecting the input ports and output ports of Vistrails modules in this way a pipeline is formed and this is called a workflow [Callahan et al., 2006].

Developers specialise the basic Vistrails module to provide various additional modules with specific processing capabilities. All ports are typed and only input ports and output ports of the same specialised type may be connected.

2.2 Remote Methods and Mobile Code

As parallel computing expands, systems such as cloud computing and standards such as message parsing interface (MPI) encourage scientists to construct complex distributed solutions that span the networks [MPI development team, 2011, Zinn et al., 2010]. A valuable addition to geospatialy enabled scientific workflows aims to provide a simple concise notation that allows for easy parallelization and supports the composition of large numbers of parallel computations exploring various parameterisations.

Mobile code has found its most succes in the field of relational databases where structured query language (SQL) performs the function of a concise code snipit sent to a remote data source where it is executed, prcoessing occurres and possibly results resturned. In many cases scientific computing has very similar requirements to relational databases in that performing operations at the data is far more efficient than retrieving the entire data set and then performing those operations locally. In the same way the dynamic nature of python simplifies the use of mobile code and allows for some elegant and sophisticated solutions in which mobile code like SQL can be sent to a remote data source for execution.

Another technolgy is that of remote objects supported by remote method invocation (RMI) that allows developers to instantiate an instance of a object on a remote compute node and iteract with it as if it were a local instance. The research involved the evaluation of a number of remote objects and parallel process execution frameworks including Pyro [Pyro development team, 2011], Python multiprocessing [Python development team, 2011], Parallel Python [Parallel Python development team, 2011] and ipython. Eventually RPyC was decided upon due to its flexibility, stability and support for both mobile code and remote objects [RPyC Team, 2011].

RPyC provides us with both remote objects capabilities via remote method calls, remote procedure calls and the ability for mobile code that can be execute on a compute node. Here we use the term compute node to refer to a computing resource along with any peripheral network and data handling capabilities that allows us remote access to a fully fledged python interpreter via RPyC. It is thus possible for a number of compute nodes to be running on a single physical machine. In the future the RPyC requirement will be weakened as it is possible to deploy a

RPyC compute node to any python capable physical machine that has SSH running [RPyC Team, 2011].

3 Design

In order to allow for the execution of Vistrails modules on remote compute nodes a decorator is applied to a standard Vistrails module. The decorator is contractual agreement by the developer indicating the module is now capable of executing on remote compute nodes i.e. it is thread and multi-processor safe. In addition the decorator acts as a mixin that upon execution of the Vistrails module by the workflow engine replaces the module in Vistrails on the users desktop with a proxy and moves all execution code to the remote compute nodes. This proxy Vistrails module is wired to the Vistrails module on a remote compute node. In other words: remote objects occures. The result is that the user of the Vistrails system is unable to differentiate between a locally executing Vistrails module and a remotely executing Vistrails module as shown in Figure 3. In fact if there are no remote compute node capabilties available the module will run on the local machine as if it were a normal Vistrails Module.

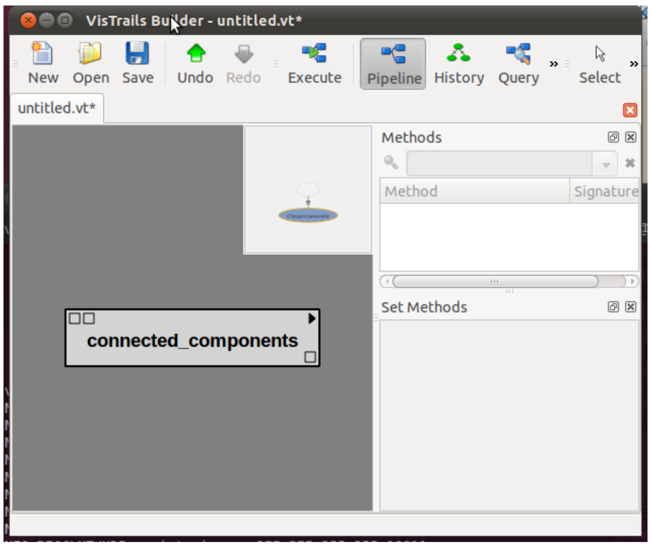
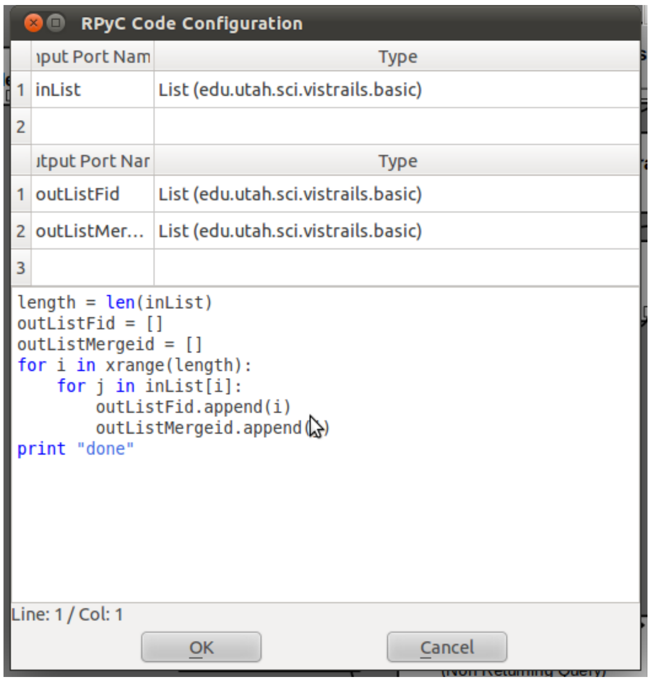


Figure 3: A Vistrails module capable of remote compute node execution looks and behaves just like any other Vistrails module

In the instance where the user wishes to execute a code snippet such as a new algorithm in his or her workflow, the system uses mobile code to transfer the given code to the remote compute node as shown in Figure 4. All relevant variables from the current local context are also transfered to the remote context. The remote compute node is now able to execute the mobile code remotely. Upon completion of execution on the remote compute node relevant results are copied back into the local context. It should be noted that the use of remote objects in our conntext involves mobile code as well since the objects need not have existed on the remote compute node prior to execution but are transferred as needed.



	ıput Port Nam	Type
1	inList	List (edu.utah.sci.vistrails.basic)
2		

	ıtput Port Nar	Type
1	outListFid	List (edu.utah.sci.vistrails.basic)
2	outListMer...	List (edu.utah.sci.vistrails.basic)
3		

```
length = len(inList)
outListFid = []
outListMergeid = []
for i in xrange(length):
    for j in inList[i]:
        outListFid.append(i)
        outListMergeid.append()
print "done"
```

Line: 1 / Col: 1

Figure 4: Example of a code snipit to be executed on a remote compute node

The result is that the entire process of remote code execution is being abstracted away from the user. This allows the user to focus mainly on the scientific workflow construction and evaluation and less on the technology that enables it. Long running, complex modules can be deployed dynamically to distributed high performance computing infrastructure where they have access to large amounts of RAM, CPU cycles and high performance data access just by selecting the appropriate destination from a drop down as shown in Figure 5.

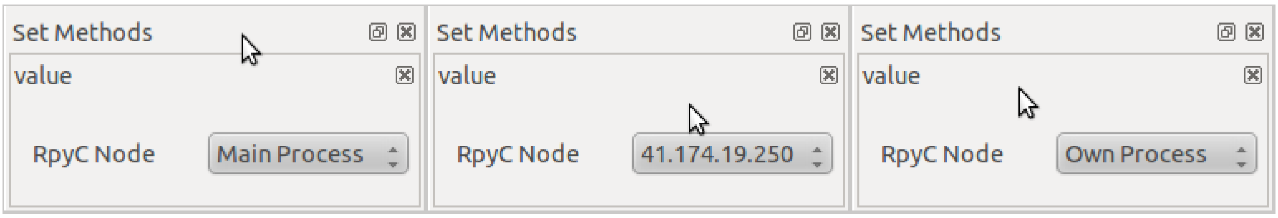


Figure 5: Selection of the destination remote compute node from a drop down selection box

Where possible, the appropriate library dependencies and Vistrails dependencies are also remoted to the executing remote compute node. Effectively, the compute nodes need have very little knowledge of the workflows that will be executed on them. This creates a level of independence between the user and the infrastructure being used. In our research, we seamlessly integrated remote execution of PostGIS queries, Numpy remoting, NetworkX and PySAL on distributed computing resources to perform spatio-temporal processing of multi-million row datasets. {significantly reducing processing time and allowing careful re-inspection of the process steps to check for correctness}

4 Results and Lessons Learnt

4.1 Results

The system was primarily tested in the wildfire research domain using amongst many the workflow shown in Figure 6. 500 metre burned area observations from the MODIS instrument for a four year period, across four scenes, derived from two algorithms ($>$ 200 individual scenes) needed to be roughly clustered (via spatial and temporal adjacency) into presumed individual fire events for the given time period. These events then needed to be associated with weather data at a coarser resolution, and the resultant data exported to a collection of CSV files for use in a downstream statistical analysis. The approach taken to this problem involved the development of a graph data structure from a set of spatial and temporal queries and the use of an algorithm to extract all the subgraphs, where each subgraph would be one fire event. PostGIS, PostGIS WKT Raster and NetworkX were the primary tools used in the exercise, alongside the functionality provided by Vistrails. Figure 7 shows the provenace of this workflow.

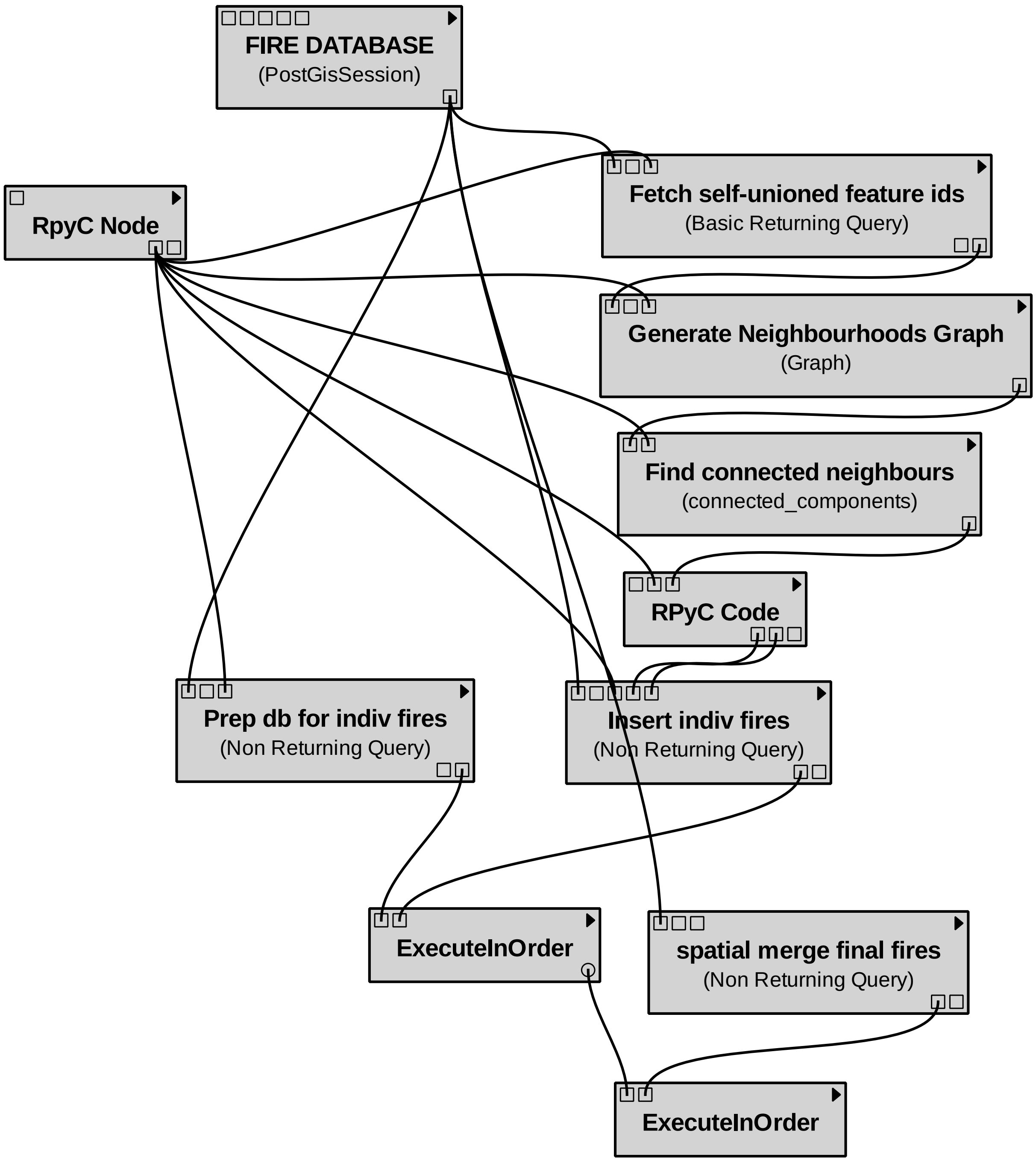


Figure 6: One of the workflows used in the fire prototype

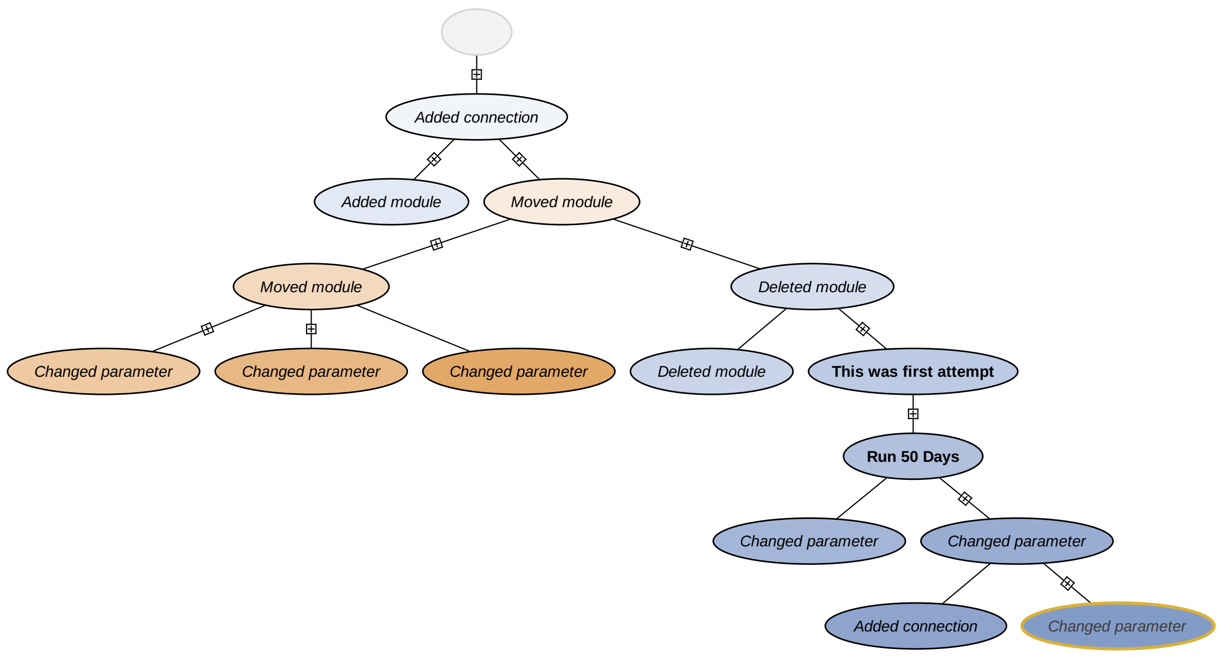


Figure 7: Provenance for prototype test workflow

Given the large spatial and temporal extent of the data and the number of spatial tests that needed to be executed, the RAM requirements and the parallel nature of the problem necessitated the use of some high performance computing capabilities. The problem was initially solved on the desktop in EO4Vistrails using a small subset of the data. Then the relevant modules where tasked to execute on remote compute nodes and the workflow was re-parameterised and re-executed, without altering or specialising the logic of the process. Results were returned acceptably after a moderately long running time.

Other problems that have been tested using remote compute nodes include various raster operations including NDVI calculations. The overall result is that the architecture and design of the system supported the requirments as layed out in Section 1. In the following section we describe some of our lessons learnt.

4.2 Lessons Learnt

The overall result was mixed, with some success and in other places much difficulty. Here we outline and discuss some of the lessons learnt:

- Modules should be stateless and atomic. This is especially important when modules need to be placed on remote nodes and when parallelisation is required. This style of componentisation is common in scientific workflow environments and encouraged by the community at large. However, this has a significant impact on performance, as the module needs to complete its operations on all the data before it can pass control to the next Vistrails module. A more efficient mechanism may be for the current module to perform some processing and write back intermediate results for the next Vistrails module in the processing chain to work on concurrently, though this possible solution remains an open research question. This limitation can be overcome to some extent by allowing for multiple concurrent executions of the same workflow each tackling a different piece of the same problem.

- Passing of URIs is preferable to passing of data in many cases. A module should read the data from the URI (which may be local). After processing, the module should write the results back to a file. This procedure has significant benefit since the storing of intermediate results allows for a form of caching such that workflows can be resumed from the last unchanged module. Additionally, this provides for persistence of intermediate results. Passing URIs around is far cheaper than moving data and allows for a level of flexibility where, for example, if a module is run from different locations, there is no effect on the code.

- C and C++ code underlying Python libraries is not easily remoted in the same way that Python code is. In these scenarios the libraries must be present on the remote compute node.

- C and C++ data structures such as Numpy arrays and the GDAL and OGR data translators can not be used via Python remote objects. These issues can be overcome by following the recommendation to write all intermediate results to disk or some shared high performance data space.

- Visualisation requires special precautions especially in cases where the interaction with the visualisation is dynamic such as found in a GIS. In cases of less interactive visualisation, the images or videos may be generated and transfered to the users desktop from the remote nodes.

- Vistrails modules accessing data from a Vistrails module on another compute node can be prohibitively slow, even when the compute nodes reside on the same machine. This is due to interprocess communication. This problem can be overcome on a single physical device by placing all data structures in shared memory. However, when working across physical devices, the correct solution is the intermediate writing of data to a fast shared data store such as a storage area network (SAN) or Network-attached storage (NAS). In the case where two process will run sequentially on the same data, the two Vistrails modules can be run in the same compute node one after each other. Certain naturally parallel processes are not bound by the same problem.

- Although the RPyC approach taken tries to protect the end user from the underlying technology and the details of where processes are running as much as possible, this is not always easy when Vistrails modules make use of local resources such as temporary files. However this detail in most cases becomes a challenge for the Vistrails module developers as opposed to the end user.

- Allowing users to execute arbitrary mobile code on private infrastructure may seem like a major security issue. Providing read only access to a data source may be one solution, which would not hinder the scientific process much, since most scientific processes transform a copy of the source data rather than the source data itself.

5 Conclusion

In conclusion the use of remote objects and mobile code coupled with geospatially enabled scientific workflows provides a viable alternative to web service based approaches. In fact when combined with the Web Services based approach, better results may be achieved. We expect an increased up take of scientific workflows and access to distributed commodity computing resources will see mobile code becoming more mainstream. The architecture and design outlined here is a candidate solution to enabling these resources in the geospatial scientific computing environment.

References

K. Asanovic, R. Bodik, B.C. Catanzaro, J.J. Gebis, P. Husbands, K. Keutzer, D.A. Patterson, W.L. Plishker, J. Shalf, S.W. Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

Adam Barker and J. Van Hemert. Scientific workflow: A survey and research directions. *Parallel Processing and Applied Mathematics*, pages 746–753, 2008. URL `http://www.springerlink.com/index/52053063K0708658.pdf`.

S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, S.P. Callahan, and H.T. Vo. Vistrails: Visualization meets data management. In *ACM SIGMOD*, Chicago, Illinois, USA, 2006.

E. Deelman and A. Chervenak. Data management challenges of data-intensive scientific workflows. In *Proc. 8th IEEE Int. Symp. Cluster Computing and the Grid CCGRID '08*, pages 687–692, 2008. doi: 10.1109/CCGRID.2008.24.

E. Deelman and Y. Gil. Managing large-scale scientific workflows in distributed environments: Experiences and challenges. In *Proc. Second IEEE Int. Conf. e-Science and Grid Computing e-Science '06*, 2006. doi: 10.1109/E-SCIENCE.2006.261077.

EO4Vistrails development team. Eo4vistrails. Accessed 31-07-2011, 2011. URL `http://code.google.com/p/eo4vistrails/`.

I. Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid. *Grid computing: making the global infrastructure a reality*, 2001. URL `http://books.google.com/books?hl=en&amp;lr=&amp;id=b4LWXLRBRLsC&amp;oi=fnd&amp;pg=PA17`

I. Foster, Carl Kesselman, J.M. Nick, and Steven Tuecke. The physiology of the grid. *Grid computing: making the global infrastructure a reality*, pages 217–250, 2003. URL `http://books.google.com/books?hl=en&amp;lr=&amp;id=b4LWXLRBRLsC&amp;oi=fnd&amp;pg=PA21`

A. Gibson, M. Gamble, K. Wolstencroft, T. Oinn, and C. Goble. The data playground: An intuitive workflow specification environment. In *Proc. IEEE Int e-Science and Grid Computing Conf*, pages 59–68, 2007. doi: 10.1109/E-SCIENCE.2007.72.

Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40 (12):24–32, 2007. doi: 10.1109/MC.2007.421.

B. Howe, P. Lawson, R. Bellinger, E. Anderson, E. Santos, J. Freire, C. Scheidegger, A. Baptista, and C. Silva. End-to-end escience: Integrating workflow, query, visualization, and provenance at an ocean observatory. In *Proc. IEEE Fourth Int. Conf. eScience eScience '08*, pages 127–134, 2008. doi: .2008.67.

B. Ludascher, S. Bowers, T. Mcphillips, and N. Podhorszki. Scientific workflows: More e-science mileage from cyberinfrastructure. In *Proc. Second IEEE Int. Conf. e-Science and Grid Computing e-Science '06*, 2006. doi: 10.1109/E-SCIENCE.2006.261078.

G. McFerren, T. van Zyl, and A. Vahed. Foss geospatial libraries in scientific workflow environments: Experiences and directions. In *Selected academic posters of FOSS4G 2010*, 2010. Accesed 14-04-2011.

Zdzislaw Meglicki. Advanced Scientific Computing. 2001.

MPI development team. Message passing interface. Accesed 31-07-2011, 2011. URL `http://www.mcs.anl.gov/research/projects/mpi/`.

Open Geospatial Consortium. OGC standards and specifications. Accessed 23-10-2010, 2010. URL `http://www.opengeospatial.org/standards`.

Open Source Geospatial Foundation Project. Quantum gis geographic information system. Accessed 23-10-2010, 2010. URL `http://qgis.osgeo.org`.

OPeNDAP development team. OPeNDAP: Open-source Project for a Network Data Access Protocol. Accessed 15-07-2011, 2011. URL `http://www.opendap.org/`.

Parallel Python development team. Parallel python. Accessed 31-07-2011, 2011. URL `http://www.parallelpython.com/`.

Pyro development team. Pyro. Accessed 31-07-2011, 2011. URL `http://irmen.home.xs4all.nl/pyro/`.

PySAL development team. Python spatial analysis library. Accessed 15-07-2011, 2011. URL `http://pysal.org/`.

Python development team. Python. Accessed 31-07-2011, 2011. URL `http://docs.python.org/`.

R development team. R: A language and environment for statistical computing. *R Foundation for Statistical Computing Vienna Austria ISBN*, 3(10), 2008.

Refractions Research. PostGIS: Support for geographic objects to the PostgreSQL object-relational database. Accessed 23-10-2010, 2010. URL `http://postgis.refractions.net`.

RPyC Team. RPyC. Accessed 31-07-2011, 2011. URL `http://rpyc.wikidot.com/`.

D. Zinn, Q. Hart, B. Ludascher, and Y. Simmhan. Streaming satellite data to cloud workflows for on-demand computing of environmental data products. In *Proc. 5th Workshop Workflows in Support of Large-Scale Science (WORKS)*, pages 1–8, 2010. doi: 10.1109/WORKS.2010.5671841.