Distributed Neuroimaging Analysis with Nipype and IPython

Satrajit Ghosh[1], Brian Granger[2], Fernando Pérez[3]

[1]MIT, Cambridge, MA [2]California Polytechnic State University, San Luis Obispo, CA [3]University of California, Berkeley, CA

Nipype: `http://nipy.sf.net/nipype` IPython: `http://ipython.scipy.org`

Introduction

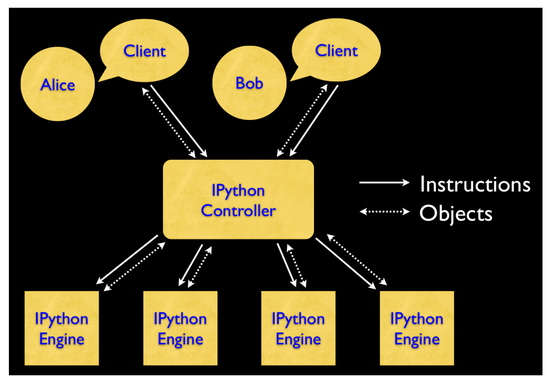
- **Data explosion.** We face an explosion in the size and complexity of neuroimaging datasets, whose analysis not only requires sophisticated algorithms but computational infrastructure to support their use with ease, efficiency, reliability and reproducibility.
- **Limited infrastructure.** While existing analysis packages provide excellent tools, they often constrain users to specific workflows and environments. Furthermore, very few of the available neuroimaging tools take advantage of the growing number of parallel hardware configurations (multicore, clusters, clouds and supercomputers).
- **Nipype+IPython.** We present distributed execution of neuroimaging analysis pipelines that the open source Nipype [2] project has developed by integrating with IPython's [3].

Nipype

- **Software explosion.** Current neuroimaging software offer users various ways to analyze their data. However, this has resulted in a heterogeneous collection of specialized applications without transparent interoperability or a uniform operating interface.
- **Uniform interface.** Nipype, an open-source, community-developed initiative under the umbrella of Nipy [1], solves these issues by providing a uniform interface to existing neuroimaging software and by facilitating interaction between these packages using workflows.
- **Workflow-based analysis.** Nipype provides an environment that not only encourages interactive exploration of algorithms, but simplifies the design and execution of neuroimaging analysis workflows.

IPython: Interactive Python

An **enhanced interactive shell** and **components for interactive parallel computing**.



- **Multiengine interface.** Direct control of code execution on any set of engines.
- **Task interface.** Automatically load-balanced task execution.
- **One cluster, many clients.** Multiple clients can interact simultaneously with one cluster.
- **Flexible.** Multicore, network, cloud, . . .

Features

- Parallel execution of complex workflows.
- No additional user code required for parallel execution.
- Embarassingly parallel execution supported for wrapped interfaces (currently supports SPM, FSL, FreeSurfer)
- Automatic load balancing across cluster.
- Multiple protocols for parallel communication.
- Open Source: BSD Licensed.

Indirect distribution

Import components

```
In [1]: from glob import glob
In [2]: import os
In [3]: from nipype.interfaces.spm import Normalize
In [4]: from nipype.pipeline.engine import (Workflow,
                                            Node)
```

Setup variables

```
In [5]: fl = glob(os.path.abspath('SAD*.nii'))
In [6]: normalize = Node(Normalize(template= \
                                   '/templates/T2.nii'),
   ...:                  name='normalize')
In [7]: normalize.iterables = ('source', fl)
In [8]: normworkflow = Workflow(name='normalizefa',
   ...:                         base_dir='./norm_workdir')
In [9]: normworkflow.add_nodes([normalize])
```

Run in parallel

```
In [10]: normworkflow.run()
```

Direct distribution

Import components

```
In [1]: from glob import glob
In [2]: import os
In [3]: from IPython.kernel import client
In [4]: from IPython.kernel.client import StringTask
In [5]: from nipype.interfaces.spm import Normalize
```

Setup variables

```
In [6]: taskstr = """import os
   ...: script_dir = 'script_dir/subj_%d'
   ...: os.makedirs(script_dir)
   ...: os.chdir(script_dir)
   ...: result = node.run(source='%s')"""
In [7]: node = Normalize(template='/templates/T2.nii')
In [8]: tc = client.TaskClient()
In [9]: fl = glob(os.path.abspath('SAD*.nii'))
```

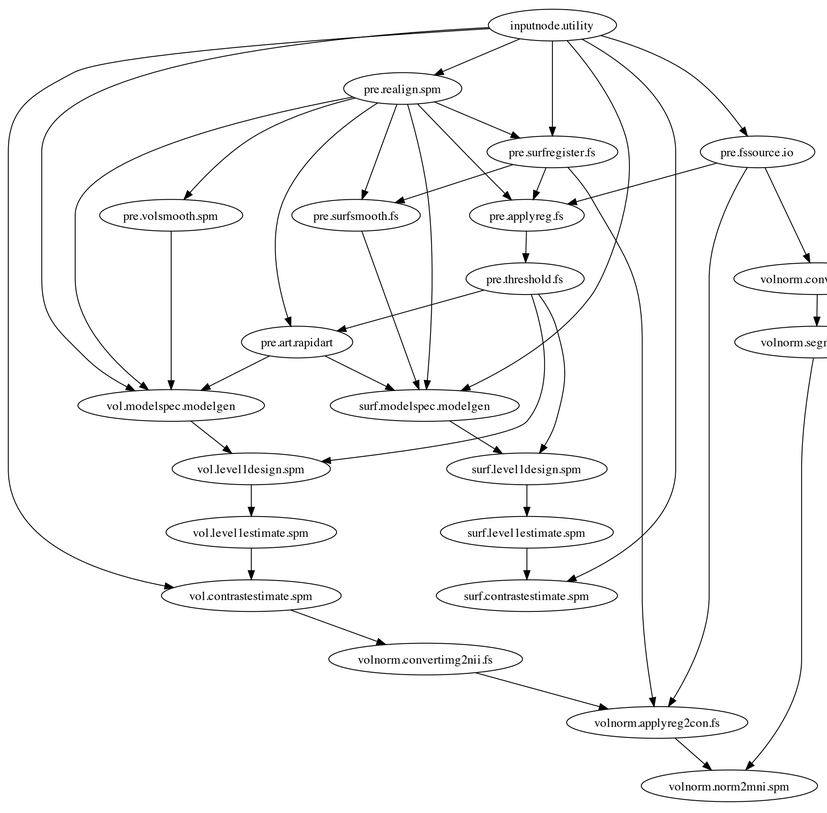
Run in parallel

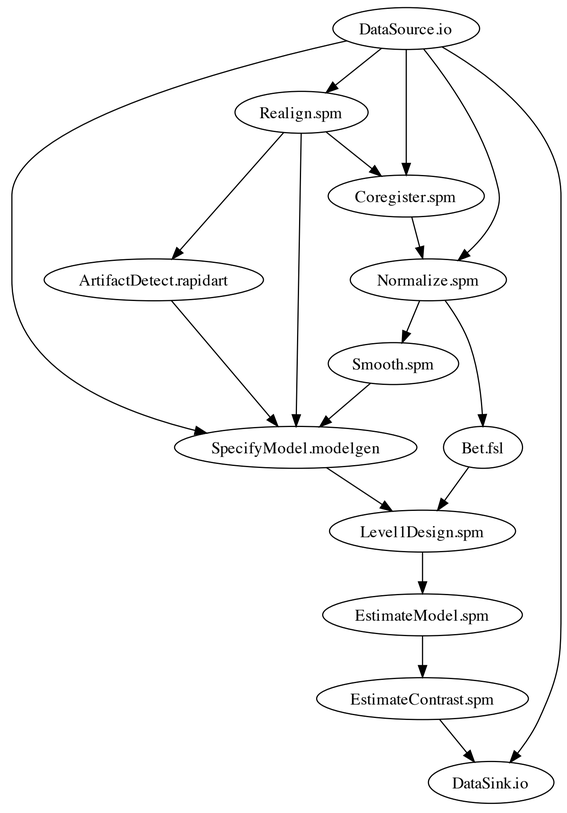
```
In [10]: ids = [tc.run(StringTask(taskstr%(i,f),
   ....:         push=dict(node=node),pull=['result']))
   ....:         for i, f in enumerate(fl)]
```

Future plans

- **IPython DAG support.** Moving the DAG scheduler to IPython itself for efficient load balancing
- **Dataserver-centric parallelization.** Replace Nipype controller with data-server (e.g., XNAT) -controlled, file-system independent execution.
- **Intelligent routing.** Processes will be routed based on their attributes (e.g., OS specificity, software availability) or user requirements (e.g., time, funding).

Complex workflows





Smoothing comparison workflow
(4 core single machine)

Serial	1 Subject	41min
Parallel	1 Subject	25min
Parallel	**2 Subjects**	**41min**

SPM workflow
(40 core cluster)

Serial	1 Subject	32min
Parallel	1 Subject	30min
Parallel	**69 Subjects**	**1 hr 40 min**

Conclusion

- **Integration.** Integrated IPython's parallel and distributed computing capabilities into Nipype's pipeline execution machinery.
- **Codeless parallelization.** Nipype interfaces can be parallelized without explicit parallel programming.
- **Multi-package support.** Current support for many SPM, FSL and FreeSurfer functions. Available on different distributed computing architectures with a common file-system and have these pacakges installed.

References

[1] M. Brett. Nipy: an open library and development framework for fmri data analysis. *Neuroimage*, 47(S1), 2009. http://nipy.org.

[2] S. Ghosh, S. Whitfield-Gabrieli, and A. Nieto Castanon. A python-based software package for pipelined, batch analysis of fmri data. *Neuroimage*, 47(S1), 2009. http://www.nitrc.org/projects/nipype/.

[3] F. Perez and B. E. Granger. IPython: A system for interactive scientific computing. volume 9 of *Computing in Science and Engineering*, pages 21–29, 2007. http://ipython.scipy.org.

Funding

This project was supported by NIH grants NIBIB R03 EB008673 (PI: Ghosh, Whitfield-Gabrieli), NIMH R01 MH081909 (PI: D'Esposito).