Distributed Neuroimaging Analysis with Nipype and IPython

Abstract No:

816 MT-PM

Authors:

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

Institutions:

[1]MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MA, [2]Cal Poly San Luis Obispo, San Luis Obispo, CA, [3]University of California, Berkeley, Berkeley, CA

Introduction:

We present the support for distributed execution of neuroimaging analysis pipelines that the open source Nipype (http://nipy.sf.net/nipype [1]) project has developed by integrating with IPython's (http://ipython.scipy.org [2]) parallel computing facilities. Today we face an explosion in the size and complexity of neuroimaging datasets, whose analysis requires not only sophisticated algorithms but computational infrastructure to support their use with ease, efficiency, reliability and reproducibility. 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) or integration with neuroimaging databases. By using IPython's high-level facilities for distributed computing, Nipype can optimally execute any parallelizable analysis pipeline.

Methods:

Python has become the leading open-source language for high-level scientific computing, and the Nipy project (http://nipy.sf.net; [3]) unites a number of efforts to develop modern open source tools for the analysis of neuroscientific data. Nipype is one of the nipy components, focused on providing: 1) a uniform interface for accessing different neuroimaging software, including existing packages like FSL or SPM as well as custom code; 2) a graph-based (using the Networkx library [4]) pipeline interface for the efficient specification of neuroimaging analysis workflows; and 3) a plugin-like environment to easily embed new algorithms and functionality into workflows.

High-level languages such as Python or Matlab are ideal for exploratory data analysis, where a scientist must repeatedly change code and ideas as the problem is better understood. For this workflow to be sufficiently fluid, an interactive experimentation environment is necessary. IPython was developed to provide such support for the Python language, and today it is widely used across disciplines. Since several key abstractions necessary for interactive work are in fact very similar to those needed to distribute a computation, IPython provides a high-level set of tools for controlling a distributed set of computational engines.

IPython provides the facilities for starting and controlling these engines, sending data and code to them, retrieving results from them, and more. By building on top of these basic functions, IPython provides different interfaces to express parallel computations; by default one interface allows direct synchronous or asynchronous execution of code on any set of engines, and a different one allows for the specification of independent tasks which get scheduled as resources become available. This basic task mechanism provides a simple first in-first out (FIFO) scheduling of tasks with automatic load balancing and fault recovery provisions.

Results:

Since Nipype represents an analysis pipeline as a Directed Acyclic Graph (DAG), it has enough information to compute the proper order of execution of nodes and to infer the dependencies between them. It can thus find the parts of the pipeline which can be executed in parallel, and using IPython, send them to the computational engines. Since IPython handles the integration with a distributed computing environment (job control systems, starting and stopping engines, security and more), Nipype can focus on the aspects of the problem specific to pipeline execution.

The current version of the system manages the pipeline DAG manually and uses IPython to send requests to the individual engines. As processing statistics are collected, this can be used to automatically route node execution to the most appropriate computational node.

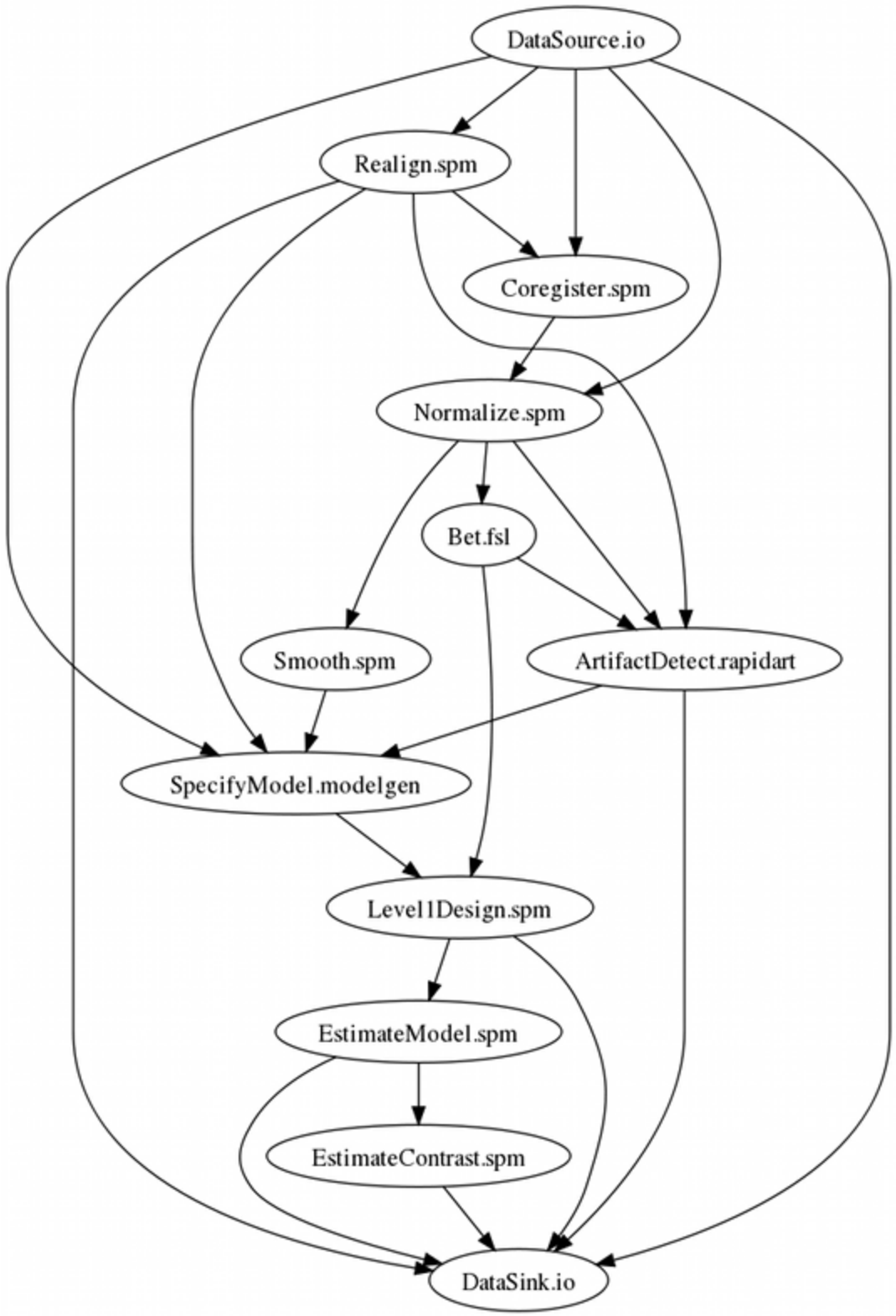
The combination of Nipype and IPython provides convenient, transparent parallel and distributed execution of any neuroimaging pipeline. Since IPython can spawn multiple engines on multicore computers or any other parallel hardware (including the cloud), users of Nipype can immediately execute their pipelines taking advantage of available parallel resources. For debugging purposes, Nipype is capable of executing any pipeline in serial mode, so that users can easily isolate any problems in their code independent of parallelization issues.

To demonstrate the efficiency of this architecture, we ran a first-level SPM pipeline and combined contrast images with a one sample t-test at the second level. The DAG for this workflow (see figure) is created automatically by nipype to account for two subjects and two different smoothing kernels. Nipype extracts the available parallelism from this DAG, and when the pipeline is executed on a 4-core machine, a 1.7x speedup relative to the serial version is obtained. This speedup is less than the naively expected 4x due to the fraction of tasks in the DAG that must be executed serially (Amdahl's law).

Conclusions:

We have developed a practical integration of IPython's parallel and distributed computing capabilities into Nipype's pipeline execution machinery. This provides an easy to use system for parallelizing analysis tasks with minimal effort and without the requirement for explicit parallel programming. Currently, one can execute SPM, FSL and a variety of FreeSurfer functions in parallel using this system on variety of different distributed computing architectures as long as they share a common file-system and have all underlying tools installed. Future improvements will focus on moving the DAG scheduler to IPython itself for efficient load balancing, file-system independent execution, selective execution of nodes on compatible systems (e.g., run FSL and FreeSurfer nodes only on unix systems in a mixed operating system cluster), interfacing with cloud services and support of more neuroimaging analysis packages.

Supported by NIH grants - NIMH 5R01MH081909-02, NIBIB 1R03EB008673-01



References:

Ghosh, S. (2009), 'A Python-based software package for pipelined, batch analysis of fMRI data', *NeuroImage*, vol. 47, no. S1.

Perez, F. (2007), 'IPython: A System for Interactive Scientific Computing', *Computing in Science and*

Engineering, vol. 9, no. 3, pp. 21-29.
Brett, M. (2009), 'NIPY: an open library and development framework for FMRI data analysis', *Neuroimage*, vol. 47, no. S1.
Hagberg, A. (2008), 'Exploring network structure, dynamics, and function using NetworkX', *Proceedings of the 7th Python in Science Conference* .

Categories

- Multi-modal Integration (Imaging Techniques and Contrast Mechanism)
- Bold fMRI (Modeling and Analysis)
- Exploratory Methods, Artifact Removal (Modeling and Analysis)
- Anatomical Studies (Neuroanatomy)