Soma-workflow: a unified and simple interface to parallel computing resources

Soizic Laguitton[1], Denis Rivière[1,2], Thomas Vincent[1], Clara Fischer[1], Dominique Geffroy[2], Nicolas Souedet[3], Isabelle Denghien[2], and Yann Cointepas[1,2]

[1] CEA, I2BM, Neurospin, Gif-sur-Yvette, France
[2] IFR 49, Gif-sur-Yvette, France
[3] CEA, I2BM, MIRCEN, Fontenay-aux-Roses, France
http://www.brainvisa.info/soma/soma-workflow

Abstract. Parallel computing resources are now available everywhere, from a simple multiple core laptop to sophisticated clusters and grids. Soma-workflow originated in the observation that many computational processes in neuroimaging are highly parallel by nature and could take advantage of the available resources with few adaptations. These processes can be described by workflows which are sequences of parallel and serial sub-tasks. Soma-workflow is a software which makes easier the execution, control and monitoring of such workflow on various computing resources. It provides the possibility to submit a whole workflow without dealing with sub-task submission and dependencies. It also offers a homogeneous interface to the different resources. Soma-workflow was created for the purpose of being plugged to external software or being used directly by non expert users.

Keywords: Parallel computing, Distributed computing, Workflows, Coarse-grained parallelism, Neuroimaging

1 Introduction

Parallel computing resources are now highly available for researchers, from simple multiple core laptops, to sophisticated clusters and grids. Many steps in the research process can benefit from these resources. Neuroimaging research is a good example, it involves a lot of chained computational processes applied on high dimensional data. In this research field, the use of parallel resources can enable the validation of methods on larger datasets, the acceleration of some algorithms or the use of non-regression tests. Coarse-grained parallelism is well adapted in most cases: it involves many long processes (from a minute to several days or weeks) which do not need to communicate together. Extensive work to introduce fine grain parallelism in algorithms is indeed worthless when the algorithm is intended to be applied on large sets of data. These coarse-grained parallelized processes, also referred to as embarrassingly parallel, can be described by workflows which are sequences of parallel and serial sub-tasks.

Setting up the execution of workflows on parallel computing resources can be time consuming and strenuous for non-expert users. Furthermore, most of the time this effort has to be done anew for each computing resource. Indeed, resources can be of different kind (cluster vs multiple core machine for example), or managed by different distributed resource management systems (DRMS) which have various interfaces. Mutual research software such as BrainVISA [1] also have to face the problem of multiple interfaces in order to make use of the available parallel resources. The gap between researchers or mutual research software, and the parallel computing resources can be efficiently bridged by software with unified interface and practical features. The key features are the workflow management system and the monitoring tool. The former provides the possibility to submit a whole workflow without dealing with sub-task submission while respecting their dependencies. The monitoring tool displays the state of the workflow at any time during its execution.

Several levels of workflow description exist. We call "pipeline" a high level description: the sub-tasks are associated to functions which are possibly implemented by a choice of methods, the inputs and outputs are typed and have semantic descriptions. BrainVISA [1], Loni [2] or Nipype [3] are pipeline software in neuroimaging, but many pipeline software exist in a broad range of scientific domains [4–7]. Among other features, pipeline software provide solutions for pipeline creation, sharing and perenniality. Opposed to pipelines, low level workflows are "executable" workflows: their sub-tasks are executable jobs defined by program command lines.

A software dedicated to the execution of low level workflows on parallel resources is of interest. It can be plugged to pipeline software or can be used directly by end users who work independently from pipeline software, even if they already have settled their chain of computational processes. To our knowledge, software offering the workflow execution feature on parallel resource are either bound to a DRMS like DAGMan with Condor [8], or part of pipeline software [2, 5, 4, 7]. In the latter case, the conversion from pipeline to low level workflow is often done transparently for the user. The lack of an explicit or simple format for the executable workflow excludes the possibility to execute workflows from either end user or other pipeline software. Moreover, many software are dedicated to grids [6, 4, 7] and interface with grid middle-ware software such as Globus [9] or gLite [10]. The possibility to use them on a simple multiple core machine or on a cluster with common DRMS such as LSF, PBS, Condor and Grid Engine, is often missing or incomplete.

Various choices in the workflow description are possible: direct acyclic graph (DAG), Petri Net-based description [7] or specific iteration language [11] for example. The description of workflows as direct acyclic graphs (DAG) has the advantage of being simple and easily handled by users [8, 6, 2, 4]. The dynamic feature is sometimes added to these simple workflow description using special dynamic nodes [2].

A solution to address the aforementioned needs is presented in this paper: Soma-workflow is a Python application dedicated to the execution of low level

workflows on a wide range of parallel computing resources. Soma-workflow was created for the purpose of being plugged to external pipeline software or being used directly by non expert users. Its application programming interface (API) enables the users to create workflows described by simple DAG. The workflow can be submitted, controlled and monitored using the same API or a graphical user interface (GUI).

The rest of this paper is organized as follows. Section 2 is dedicated to the presentation of soma-workflow. An overview of the software is provided with the presentation of its main features. The core of the application is then addressed in the description of workflows and their execution. Afterward, the light interactions between soma-workflow and parallel resources is explained and justified. Soma-workflow aims at being flexible: its different modes of utilization are then presented, as well as tools which handle file transfer and file path matching problems. The benefit of soma-workflow in concrete cases is demonstrated in the section 3, with the presentation of three use cases in neuroimaging.

2 Soma-workflow

2.1 Main features

Soma-workflow provides a set of features in order to make the access to computing resources easier for non expert users:

- Unified interface to submit, control and monitor jobs and workflows on various computing resources. Soma-workflow was successfully used on multiple core machines and clusters with the DRMS: Grid Engine, LSF, Torque and Condor. Other systems can be plugged easily, in particular the systems which implement DRMAA [12] (see section 2.3).
- Python API designed to be complete but simple and usable by non expert users.
- A GUI to submit, control and especially monitor the workflow execution. (Fig. 1)
- Several installation modes: single process or client-server mode.
- Transparent remote access to computing resources and disconnection at any time with the client-server mode.
- Tools to handle file transfers and/or path name matchings in case the user machine and the computing resource have separated file systems.
- Prevention of the saturation of the computing resource queues.

2.2 Workflow description and execution

The workflows are created using the Python API of soma-workflow. The workflows are described by direct acyclic graphs. The graph nodes correspond to the sub-tasks of workflows and the arrows to the execution dependencies between sub-tasks. Each sub-task, also called job, wraps a program command line call.

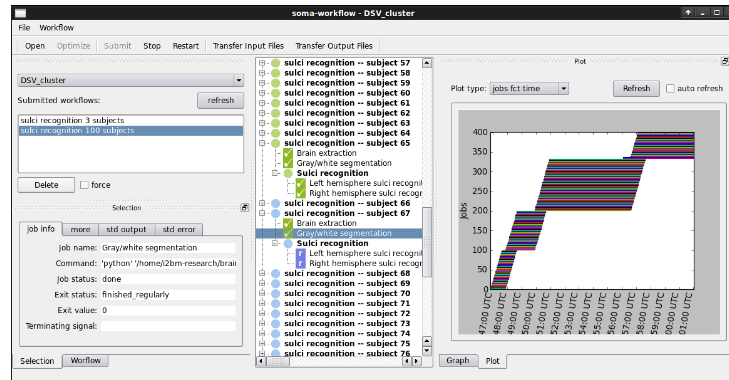

Fig. 1. Overview of Soma-workflow GUI. On the left: computing resource selection, list of submitted workflows, current selection and workflow information. In the middle: a view of the workflow in the form of a tree (green means ended jobs, blue means running jobs). On the right: execution plots, here the jobs function of the time.

Optional file paths for job standard input, output and error, and job working directory path can be specified.

Soma-workflow handles the execution of workflows on computing resources. Each job starts as soon as possible, considering its dependencies with other jobs. If a job fails the execution of the graph branch is stopped but independent branches keep on running. The status of workflows and jobs as well as other information like job standard output/error or exit value are displayed continuously in the GUI and can be recovered at any time using the API. The workflows can be stopped at any time: jobs are killed if they are running and removed from the computing resource queue if they are waiting. The workflows can be restarted: the jobs which failed or were killed are submitted again.

The execution of the workflows can also be data driven. Each job can be associated to file transfer objects which represent input and/or output files. The execution is data driven in the sense that the jobs bound to input files will not start before all their input files exist.

A workflow can be made of thousands of jobs, and hundreds of jobs can be ready to execute at the same time. In the spirit of sharing a computing resource among a group of users, one user might want to avoid saturating the waiting queue with his jobs. Furthermore, a maximum limit of jobs per user in the queue can be configured on some computing resource. For these two reasons, a maximum number of jobs can be settled in soma-workflow. When the limit is reached, the jobs which cannot be added to the computing resource queue, wait in a second queue inside soma-workflow. As soon as a job leaves the resource queue, soma-workflow submits the first job in its own queue.

2.3 Interaction with parallel resources

Soma-workflow uses only very basic parallel resource functionalities:

- Submission of a job with the option of specifying standard input, output and error file paths, and working directory.
- Request of the status of jobs
- Request of the exit information of the jobs (exit value)
- Interruption and suppression of jobs

The restriction to these few essential functionalities makes it easier to use soma-workflow on a wide range of parallel resources. Interaction with a parallel resource interface can be simply added by implementing four python functions (kill_job, job_submission, get_job_status, get_job_exit_info). Jobs are identified with their own resource job identifiers.

If the resource is limited to one multiple core machine, soma-workflow offers a straightforward solution. Indeed, Soma-workflow comes with a basic stand-alone scheduler which has a waiting queue and runs jobs in parallel. The maximum number of jobs which can run in parallel is adjusted by the user.

In the case of distributed computing resources, namely a set of computers and users managed by a DRMS, a major objective for soma-workflow was to interact successfully with the API called DRMAA (Distributed Resource Management Application API) [12]. The DRMAA specification is a software standard developed in the Open Grid Forum for the submission and control of job to heterogeneous DRMS. The version 1 of DRMAA provides a unified interface for DRMS which is however difficult to use by end users. It has another important limitation: the job identifiers are not necessarily valid across the DRMAA sessions, in consequence once the DRMAA session is closed it is not possible to monitor or control the submitted jobs anymore through DRMAA. The architecture of soma-workflow was designed to handle these limitations. Using various implementations of the first version of DRMAA, soma-workflow was used with success on clusters with the following DRMS: Torque, LSF, Oracle Grid Engine and Condor. However, other implementations of DRMAA exist: the implementation of DRMAA for Grid Way could be used to extend soma-workflow to grid computing for example. The DRMAA working group is working on the second version of the specification. The improvements of this version suggest that it will be easily plugged into soma-workflow.

2.4 Regular or client-server application

For the purpose of being flexible, soma-workflow can be used as a simple single process application or as a client-server application. The simple single process mode is well adapted to work on a single multiple core machine and can be a good starting point to use soma-workflow. The client-server mode deserves more explanations. Compared to the single process application it offers two interesting additional features:

- Transparent access to remote computing resources. The communication with the resource is done securely within a ssh tunnel.
- Disconnection at any time. The client part of the application runs on the user machine and can be closed at anytime. A user can thus submit a workflow

to a remote resource from his laptop for example and then shut it down: the workflow will keep on running on the resource. The user can open soma-workflow at any time to monitor the progression of his workflow.

In the client-server mode, a server process has to run on the computing resource. However, it is worth pointing out that none of soma-workflow processes need any special rights to run. Indeed, soma-workflow respects the system of users and permissions on the resource. The remote access to the resource is done using the user own account on the resource. Therefore, all the actions done by a user through soma-workflow are done with the user identity on the resource.

2.5 Problem of separated file systems

The previous paragraph states that soma-workflow can be used as a client-server application to access remote clusters transparently. In this case, the user machine and the computing resource do not necessarily have a shared file system. To handle files and file paths in this case, soma-workflow provides two tools materialized by two types of objects defined in the soma-workflow API: the file transfer objects and the shared resource path objects. These objects can be used instead of any file path when creating workflows.

The file transfer objects enable the users to transfer files or directories to and from the computing resource file system. A file transfer object is a mapping between a file path which is valid on the user machine and a file path which is valid on the computing resource. The location of the files or directories on the computing resource as well as their life cycle are managed by soma-workflow. To create a file transfer object, the user only needs to specify the path of the file or directory on his machine. File transfer objects are thus nearly as simple to use as local paths.

The second type of objects, the shared resource path objects, are of interest when a copy of the user's data already exists on each file system. Shared resource path objects represent file paths which are valid on both file systems. An identifier is associated to the user's data, and the computing resource path of the data root directory is registered in soma-workflow. Soma-workflow "translates" the shared resource path object into a valid path when needed.

3 Three Neuroimaging use cases

The three neuroimaging research applications presented in this section show the benefits of soma-workflow at three different steps in the research process. In the first case the parallelism exists at the level of a single data treatment as the algorithm splits the data into chunks, processes them and merges the results afterward. In the second case soma-workflow is used for non-regression tests for the purpose of reducing the sensitivity of an algorithm to variability. Finally, an extensive cross-validation process of cortical sulci identification models is performed.

3.1 Joint detection-estimation of within-subject fMRI data

At the subject level, this functional neuroimaging application aims at jointly detecting which parts of the brain are involved in a given stimulus while also estimating the underlying dynamics of activation by recovering the so-called Hemodynamic Response Function (HRF). To address these two tasks, a Bayesian joint detection-estimation (JDE) was developed in [13, 14] which takes into account the spatio-temporal structure of the observed Blood Oxygen Level Dependent (BOLD) signal. Indeed, adaptive spatial mixture models have been introduced to model spatial correlation of fMRI data at the voxel level. The temporal aspect mainly concerns the HRF modelling which is known to vary accross the whole brain and thus cannot be considered constant. However, the HRF estimation cannot be robustly performed at the voxel level due the low signal-to-noise ratio of fMRI data. The JDE framework is then based on a parcel-based modelling of the BOLD signal: the shape of the HRF is considered constant within the extent of a small brain region, while its amplitude may vary accross voxels. In this setting, the JDE approach relies on a prior functional parcellation which typically divides the input fMRI data into several hundreds of parcels. Since there are as many *independent* models as parcels, the analysis can be split up into parcel-wise *parallel* analyses (activation contiguity across parcels is not guaranteed a priori, but is observable in results). As the parcel size is not fixed, some big parcels may arise from the parcellation process and may slow down the overall parallel processing. To overcome this, the maximum parcel size was controlled by splitting too big parcels (larger than 1000 voxels) according to a balanced partionning which also guarantees the spatial connexity and thus properly satisfies the above-mentioned assumptions on the HRF. In practice, the parallel implementation with soma-workflow required a straightforward and light development task (script of 150 lines in one afternoon), which mainly consisted in linking several commands and defining their input/outputs files. A hierarchical workflow was constructed to handle all the steps of the JDE procedure, namely: data splitting into parcel data sets, parcel-wise parallel detection-estimation processes and merging of parcel results. Initially, a whole brain analysis lasted 10 hours on a single CPU and boiled down to 15 mins with soma-workflow on a 192 cores cluster. Consequently, a group study comprising 20 subjects and several acquisitions was performed within one day.

3.2 Pipeline optimization and large dataset processing

The improvement of an algorithm robustness, namely the reduction of its sensitivity to variability, can take advantage of a unified interface to make use of various parallel computing resources. In medical imaging, a lot of sources of variability can be encountered, among others acquisition modality, scanner manufacturer, acquisition parameters and scanner environment. A special consideration for the robustness of algorithms is thus required when an image analysis technique is intended to be used on images coming from many acquisition facilities. This work involves an ever growing database containing images covering as much

variability as possible. During the design of the image processing algorithm, every improvement made for a specific subset of the database must be applied on the whole data set to assess that no regression is introduced by the modification. In this process, the developer needs to use two kinds of computing resources. The first is a fast responding computing resource with little parallelization (such as a multiple core workstation) for running the algorithm on small subsets of data. The second is a large parallel resource (such as a cluster or a grid) for testing the algorithm on the entire database.

With soma-workflow, the parallelization code is independent of the computing resource. It allows the user to choose the most appropriate resource each time an algorithm (or series of algorithms) has to be run. Moreover, using soma-workflow in association with a pipeline software such as BrainVISA [1] makes it even easier to process large datasets. The creation of pipeline iterations including the parallelization and the data management is handled by BrainVISA and converted to a low level workflow, the execution is then performed by soma-workflow.

Soma-workflow has been used with BrainVISA to improve the robustness of the Morphologist pipeline of BrainVISA. This pipeline extracts the main brain structures (hemispheres, gray/white matter, cortical surface, cortical folds, etc.) from a T1-weighted MR image. The goal is to produce good quality segmentations regardless of acquisition parameters and image quality. Step by step the algorithm was tested on a total of about a thousand of T1 MR images. We made up a high variability sample database containing 80 T1 MR images picked up from various databases. Each time a new set of images produces a bad segmentation, some steps of the algorithm are improved. The whole pipeline is then tested anew on the sample database and additional tests on larger datasets can be carried out. During the pipeline improvement, preliminary tests can be done on a simple workstation. Indeed, the changes often concerns only isolated steps of the pipeline and the computational process for one subject takes about 15 to 20 minutes for the entire pipeline. However, tests on larger databases require the use of larger computing resources: the whole pipeline applied to the single sample database (80 images) takes about 23 hours on a single processor. This time is reduced to about an hour using soma-workflow on a 192 core cluster depending on the cluster occupation.

3.3 Extensive validation of cortical sulci identification model

The validation and comparison of methods are a complex but crucial steps in the research process. The increased availability of parallel resources makes it easier. The demonstration is done here with the presentation of an extensive cross-validation of the older cortical sulci identification models of BrainVISA [15].

The sulci identification model uses a congregation of about 260 artificial neural networks for a brain hemisphere. Each neural network is responsible for a local aspect of the recognition. Compared to the implementation in [15], we used SVM-based models [16]. A Markovian relaxation is responsible for the global coherence of the identification. Each neural model is trained using a supervised

learning scheme, based on a learning database of 62 manually identified brains [17]. The generalization capacity of the model had previously been estimated on a single, very limited, dataset. But a complete assessment involves a leave-one-out cross-validation of the models on the learning database (Fig. 2). This requires the training and test of 62 different complete models. The complete validation represented about 5500 hours of computing (more than 7 months), and used about 70000 individual jobs. It ran in about 3 days using approximately 100 cores of a 192-cores cluster.

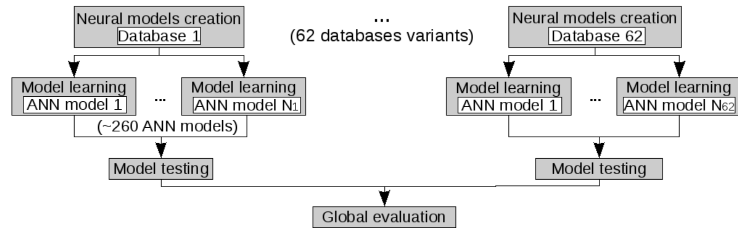

Fig. 2. Workflow graph of the sulcal model validation application. Gray boxes are the jobs, white boxes are the main data used or produced by the jobs. 62 different databases are used, each of them being composed of the whole database except one subject, and being used to train a recognition model.

The validation provided reliable generalization recognition rates, which brings a precise idea of the performance of the model, and allows comparisons with different models such as the newer methods by Perrot [17] built from the same learning database.

This work was done using BrainVISA and soma-workflow. It is interesting to note the dynamic nature of the workflow (Fig. 2). Indeed, the number of neural models may slightly vary from one learning base to another, thus the number of jobs is not known before the first steps of the workflow are executed. The execution of such a workflow was handled simply using the soma-workflow Python API which enables us to wait for jobs or workflows and submit new jobs or workflows when needed. In client-server mode, the client disconnection during the workflow execution is however forbidden using this scheme, since the execution is driven by the client application. To overcome this limitation, the possibility to create dynamic workflows will be added to soma-workflow in the future.

4 Conclusion

Soma-workflow is an application which handles the execution of low level workflows on multiple core machines and clusters. Its advanced features as well as its unique position, between computing resources and user or high level workflow software, makes it useful in a broad range of situations. The three neuroimaging use cases presented here demonstrate its relevance in the research process.

Acknowledgments. This work was funded by the ITEA2 HiPiP project. Developments relative to the implementation of JDE with soma-workflow were supported by the CL1-38093-007 Servier contract.

References

1. Cointepas, Y.: The BrainVISA project: a shared software development infrastructure for biomedical imaging research. In: Proceedings 16th HBM (2010)
2. Dinov, I., Van Horn, J.D., Lozev, K.M., Magsipoc, R., Petrosyan, P., Liu, Z., MacKenzie-Graham, A., Eggert, P., Parker, D.S. and Toga A.W.: Efficient, Distributed and Interactive Neuroimaging Data Analysis using the LONI Pipeline. In: Front Neuroinformatics. vol. 3, no. 22, pp. 1–10 (2009)
3. Nipype, `http://nipy.sourceforge.net/nipype/`
4. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. In: Nucleic Acids Research. vol. 34, iss. Web Server issue, pp. 729–732 (2006)
5. The Kepler project, `https://kepler-project.org/`
6. Sakellariou, R., Zhao, H., Deelman, E.: Mapping Workflows on Grid Resources: Experiments with the Montage Workflow. In: Grids, P2P and Services Computing, Springer, pp. 119–132 (2010)
7. Generic Workflow Execution Service, `http://www.gridworkflow.org/kwfgrid/gwes-web/`
8. Condor DAGMan, `http://www.cs.wisc.edu/condor/dagman`
9. Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. In: IFIP International Conference on Network and Parallel Computing. LNCS, vol. 3779, pp. 2–13. Springer-Verlag (2006)
10. gLite - Lightweight Middleware for Grid Computing, `http://glite.cern.ch`
11. Ruiz, M., Jones, N., Grethe, J.S.: Simplifying the Utilization of Grid Computation using Grid Wizard Enterprise. In: MICCAI Grid Workshop Proceedings. New York (2008)
12. Troger, P., Rajic, H., Haas, A., Domagalski, P.: Standardization of an API for Distributed Resource Management Systems. In: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007). pp. 619–626 (2007).
13. Risser, L., Vincent, T., Ciuciu, P.,Idier, J.: Robust extrapolation scheme for fast estimation of 3D Ising field partition functions. Application to within-subject fMRI data analysis. In: Yang, G.-Z. (ed.), MICCAI 2009, LNCS, vol. 5761, pp. 975–983, Springer, Heidelberg (2009).
14. Vincent, T., Risser, L., Ciuciu, P.: Spatially adaptive mixture modeling for analysis of within-subject fMRI time series. In: IEEE Trans. Med. Imag. vol. 29, pp. 1059–1074 (2010)
15. Rivière, D., Mangin, J.-F., Papadopoulos-Orfanos, D., Martinez, J.-M, Frouin, V., Régis, J.: Automatic recognition of cortical sulci of the Human Brain using a congregation of neural networks. In: Medical Image Analysis. vol. 6, no. 2, pp. 77–92 (2002)
16. Perrot, M., Rivière, D., Mangin, J.-F.: Identifying cortical sulci from localizations, shape and local organization. In: Proceedings of 5th IEEE ISBI. pp. 420–423 (2008)
17. Perrot, M., Rivière, D., Mangin, J.-F.: Cortical sulci recognition and spatial normalization. In: Medical Image Analysis. In press (2011)