

# From gridified scripts to workflows: the FSL Feat case

Tristan Glatard<sup>1,2,3</sup> and Sílvia D. Olabbarriaga<sup>1,2,3</sup>

<sup>1</sup>Radiology, <sup>2</sup>Clinical Epidemiology, Biostatistics and Bioinformatics  
Academic Medical Centre, University of Amsterdam, NL

<sup>3</sup>Institute of Informatics, University of Amsterdam, NL

## Abstract

In this paper we investigate the benefits of moving from monolithic script implementations to workflows for performing functional magnetic resonance imaging (fMRI) experiments on the grid. In particular, we focus on the case of the FSL fMRI Expert Analysis Tool (*Feat*). Concrete pros and cons of both approaches are provided to fMRI analysts, and lessons learned from the study of a quite complex grid workflow use-case are reported to grid scientists. Both usability and performance are studied. The problem of output presentation is described in details. Simulations results show that the performance improvement yielded by the workflow implementation in a job farming experiment is limited, whereas it is significant for parameter sweeps. Even if hard-coded ad-hoc solutions can yield benefits for some specific use-cases, using workflows for such a widely scoped application is still challenging in terms of description, usability and performance.

## 1 Introduction

Experiments performed in functional magnetic resonance imaging (fMRI) are nowadays reaching a scale that motivates the adoption of grid technology to address performance and experiment management issues [9, 1]. Meanwhile, workflow has emerged as a paradigm for gridifying applications in medical image analysis [10] as in several other scientific fields [3, 2]. In this paper we discuss the implementation of fMRI analysis using a workflow approach.

The FMRIB software library (FSL) [12], Statistical Parameter Mapping (SPM) [4], Brain Voyager [7] and Analysis of Functional NeuroImages<sup>1</sup> (AFNI) are among the representative packages used for the analysis of fMRI data. In this work, we focus on the FSL fMRI Expert Analysis Tool *Feat*, a tcl script gluing together calls to several FSL command-line utilities to implement fMRI analysis. This script is highly parametrised, at the same time hiding details of the complex image analysis pipeline underneath and enabling the user to choose subsets of the analysis to be performed in a particular invocation. Nevertheless, it is fair to say that the *Feat* application as a whole is seen by the users as a “monolithic program” that is invoked with a large list of parameters grouped into the “design file”. One can think of at least two advantages of moving from such a monolithic to a workflow approach: usability and performance. *Firstly*, workflows allow more flexibility to the application. For instance, replacing part of the application (e.g., a registration method) for

---

<sup>1</sup><http://afni.nimh.nih.gov/afni>

experimental purposes is easier with a workflow. Grid workflows also improve application management such as version maintenance (through the use of remotely maintained components) or error detection and handling. Moreover, workflow descriptions are also more suitable for execution on grids, as parallelism is naturally expressed. *Secondly*, workflows are expected to yield gains in terms of execution time due to exploitation of intrinsic parallelism in a trivial way (and without user intervention). Performance improvement may come from computation savings, in particular in parameter sweep experiments. In monolithic implementations, the whole analysis is computed for each parameter combination, even if some steps do not depend on the varied parameters (such as performed in [9]), whereas such a redundancy is naturally avoided with workflows.

Based on the experience of porting a representative application as a workflow, this paper discusses concrete pros and cons about moving from a script-based to a workflow implementation of FSL *Feat*, while reporting some lessons learned from the study of a quite complex grid workflow use-case. A workflow implementation of the *Feat* application is first described in Section 2. In the rest of the paper we then address two aspects that turned out as important in the context of this work, namely the organization of outputs generated by the workflow (Section 3) and performance analysis (Section 4). Section 5 closes the paper with preliminary conclusions of this exercise.

## 2 Workflow description

We focus on *Feat* “first-level” analysis, which aims at calculating brain activation maps of the fMRI scan of a subject in response to stimuli imposed during the acquisition. Using the General Linear Model (GLM), *Feat* determines the voxels of the fMRI scan whose intensity variation along time is correlated to the stimulus signals. Stimulus signals are also called *model Explanatory Variables* (EVs) and linear combinations of them are denoted *contrasts*, being used for example to study the independence of the EVs. Normalization of the fMRI scan to a standard or template brain is also performed to allow group studies in further steps (“high-level” analyses in *Feat*). We considered FSL version 3.3 for this study. Recently, the FSL developers in Oxford released version 4.0 of the software package in which *Feat* leverages the Sun Grid Engine to job farm across subjects.

### 2.1 Implementation

Figure 1 presents an impression of our implementation of the *Feat* application workflow. A valid description of the workflow in the Scufi language [8] is available from the myExperiment community web site<sup>2</sup>. This workflow runs on the grid infrastructure of the Dutch VL-e project<sup>3</sup> and its output has been checked to be identical to the one produced by the script version of *Feat*. Details about Scufi are available on the Taverna website<sup>4</sup>. The workflow consists of 30 grid processors corresponding to executables from FSL that were wrapped for the grid using the Generic Application Service Wrapper (GASW) [5]. Processors implementing basic functions like arithmetic operations or string concatenation were wrapped as local Java Beanshells<sup>5</sup>. The 41 inputs correspond to *Feat* parameters, as specified in the design file used by the script version. The 65 outputs are resulting files stored by *Feat* in a directory tree.

Three sub-workflows are identified in figure 1: normalization, pre-processing, model computation. The

<sup>2</sup><http://www.myexperiment.org/workflows/176>

<sup>3</sup><http://www.vl-e.nl>

<sup>4</sup>[http://www.mygrid.org.uk/usermanual1.7/scufi\\_language\\_wb\\_Features.html](http://www.mygrid.org.uk/usermanual1.7/scufi_language_wb_Features.html)

<sup>5</sup><http://beanshell.org/>

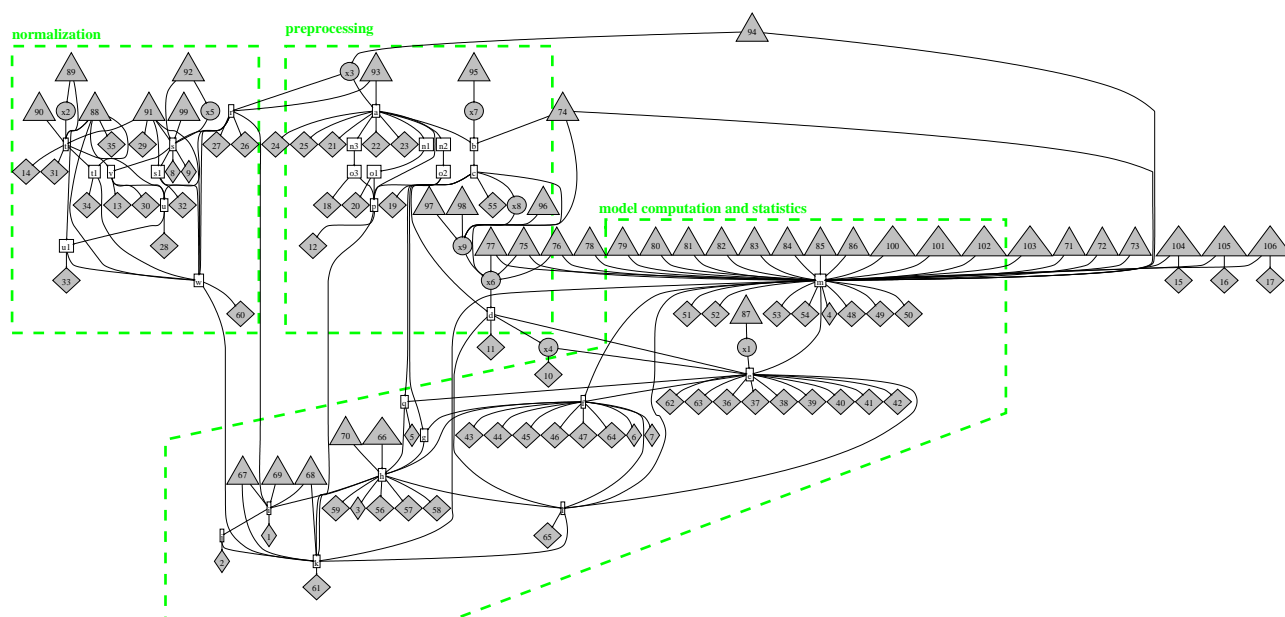


Figure 1: Workflow corresponding to the first-level analysis implemented by the *Feat* application from FSL. Triangles denote inputs and diamonds outputs. Boxes are processors executed on the grid and ellipses are local ones. For the sake of the legibility, we represented multiple data dependencies by single arrows, and processor ports and iteration strategies are omitted. Green dotted boxes indicate sub-workflows.

normalization of the fMRI scan (input 93<sup>6</sup>) to the standard brain (input 88) is composed of 3 registration steps. First, a high resolution anatomical scan of the subject (input 91) is independently mapped to the standard brain and to the fMRI scan. Then, both registration results are combined to produce the fMRI-to-standard-brain transformation. The inner sub-workflow deals with pre-processing of the fMRI scan, including motion and time correction, brain extraction and spatial filtering (a, b, c and d processors). The remaining sub-workflow corresponds to the computation of the model and activation statistics. They are then rendered (thresholded, clustered and overlaid on the fMRI scan) and an analysis report in HTML is finally generated. This workflow was executed on an analysis example involving all the implemented steps and validated against results obtained with the script implementation. We used a modified version of the Scuff “dot product” iteration strategy [6] to correctly describe job farming over several fMRI scans and sweeps over model parameters.

We chose to adopt a quite fine granularity to represent this workflow because it allows for more flexibility, since each component of the workflow (from the registration method to the results plotting format) can be customized. In a next step, granularity could be coarsened by gathering components in expandable sub-workflows.

Expressing a tool like *Feat* as a workflow is quite unwieldy, even when the underlying principle of the analysis is grasped. The application comes out of the FSL package as a 3,500 lines script, and its translation into a workflow representation is quite error-prone and difficult to validate. In the future, this kind of translation step could be avoided by directly developing such high-level applications as workflows from basic components of the software suite.

<sup>6</sup>The input numbers are given for the sake of completeness, since they are only well visible in the digital version of the paper

## 2.2 Discussion

A first glimpse at this workflow suggests that performance gains could be expected by exploiting intrinsic parallelism between sub-workflows. Sweeps over model parameters (inputs of processor 111) may also save CPU time and data storage size with respect to the monolithic implementation by computing normalization and pre-processing only once. In Section 4 we discuss in more detail the potential performance gain based on a simulation study. Note that the *Feat* script also allows for selectively performing analysis steps; doing so for large experiments, however, involves some error-prone parameter settings, whereas a workflow engine manages it automatically.

Limitations remain though. Firstly, the management of workflow output proved to be more difficult than expected (see more details in Section 3). Secondly, the translation of the script into a workflow described in Scufi was not trivial. Some steps of the *Feat* first-level analysis have not been implemented in the workflow yet, for example  $B_0$  unwarping, contrast masking, denoising and perfusion subtraction, which would increase the complexity and size of the workflow significantly. Additionally, we detected limitations in the expressiveness of the adopted workflow language to describe variable parameters. Generally speaking, “dynamic” workflow patterns are still difficult to describe in Scufi. Take the example of the number of EVs and contrasts, which have impact on model-related steps and the number of inputs and outputs. The number of input stimulus files (in case of custom-shaped stimuli signals) and the number of other parameter instances (such as the phase of the signal convolved to the stimulus) depend on the number of EVs. The cardinality of some other parameters may even depend on both the number of contrasts and EVs, such as for instance the weights given to the EVs in the contrast vectors. Ideally, all parts of the workflow should be dynamically adjusted to the number of contrasts and EVs. In practice, these are fixed parameters in the workflow, limiting its applicability to this particular fMRI experiment.

Lists manipulations could be envisaged to allow dynamic EVs and contrasts cardinality. Yet, if even possible, correctly assembling/splitting lists all along the workflow would burden the description a lot. This kind of problem is not specific to this particular fMRI use-case. It has been identified on several other applications and thus deserves thorough investigations.

Subtle changes in the experimental setup can also have significant impact on the workflow description. For instance, this implementation considers that each fMRI scan has one anatomical scan associated with it. One could wish to run experiments where a single anatomical scan is registered to all the fMRI scans of the same person. To be used for such a case, the workflow iteration strategies must be adapted.

## 3 Output organization

In order to be usable for real applications, a workflow system should not only be able to compute the results but also to deliver them in a suitable format. The workflow represented on figure 1 has 65 outputs that are stored as grid files for each run of the application. The workflow engine automatically stores the generated outputs as grid files with unique names, keeping track of provenance data. Even small parameter-sweeps or job farming experiments easily produce thousands of files. Depending on the nature of the underlying experiment, one could be interested only in a subset of those files that should be rapidly retrievable and accessible. Moreover, users may want to represent results from the same workflow in different ways. For example, one could focus on the influence of some parameter for a given contrast whereas the other would compare various contrasts for a given parameter set. An SPM user may also want to compare his/her results with output from FSL, thus requiring a different results organisation than a traditional FSL user.

Image analysis software has been addressing this problem for years by nicely organising output files in meaningful directory structures. For instance, the *Feat* application adopts the structure sketched in figure 2. Such a structure is semantically rich for the user: for example, checking the correctness of the registration procedure can be done easily by picking up files from directory *reg*. Summaries such as HTML reports can also be produced by the applications, thus helping to keep track of the various results produced by the execution (for instance *.../report.html*). However, when dealing with an experiment involving several runs of the application, it is then up to the scientist to organise the outputs in the most appropriate manner. Experienced users and other programs, however, may still require the *Feat* “native” output structure.

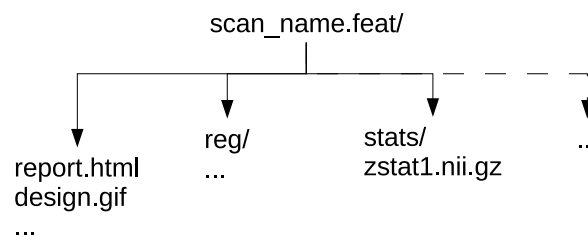


Figure 2: Directory structure adopted by the *Feat* script implementation to store results. Extensive description is available from the FSL website.

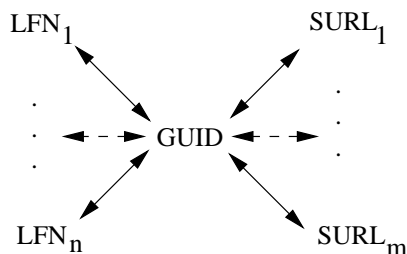


Figure 3: Classical data management architecture on production grids. A file is uniquely identified by a Grid Unique Identifier (GUID) to which  $n$  Logical File Names (LFN) may correspond in a file catalog (LFC). The file may be physically replicated on one or several sites, each of them providing a Site Unique Resource Locator (SURL).

Obtaining such a directory structure from grid workflow executions is not trivial. In this section we discuss alternatives to organize outputs of the workflow implementation. We assume that a workflow is made of *independent components* that can read/write files on a grid infrastructure. Files are supposed to be managed according to some grid middleware featuring (i) replica management among several grid sites and (ii) unified view of distributed storage through a unique file catalog. Figure 3 depicts the terminology used to refer to the various representations of a file, following the gLite middleware<sup>7</sup>.

For instance a framework with such features could be a Scuf workflow made of Web-Services that submit jobs to the EGEE grid<sup>8</sup>. Each workflow component may be iterated several times during the execution, thus replicating part of the output directory structure. For instance, sweeping over a model parameter would generate several *stats* directories corresponding to

a single *reg* directory. One could then choose either to create several *stats* directories in the same analysis directory (*analysis.feat/{reg,stats-1,stats-2,...}*) or to replicate the analysis directory for all the parameters (*analysis-1.feat/{reg,stats}*, *analysis-2.feat/{reg,stats}*, *...*). Such choices should be made by the user. In any case, uniqueness of the file names produced by the workflow components should be guaranteed, which is difficult as components are independent from each other. Moreover, write permissions must be taken into account on shared environments.

<sup>7</sup><http://glite.web.cern.ch>

<sup>8</sup><http://www.eu-egee.org/>

### 3.1 Existing approaches to workflow output organization

The most convenient way for a workflow to store grid files is to use exclusively GUIDs. No name collision will occur among independent components and catalog permission problems are avoided. However, the file organization provided by GUID is completely meaningless and absolutely intractable for large numbers of files.

Information about the path of a result in the catalog could be set into the description of the workflow component that produces it. For instance, it could be specified that the `flirt` component of the *Feat* workflow always writes LFNs in `$diri/reg`, where `$diri` is given as its input. This is the solution adopted in the GASW wrapping system that we used [5]. In practice, inconsistency problems rapidly arise from such a strategy and a global check of output paths is required. At some point, unique identifiers or workflow global variables (*e.g.* specifying the root directory of a workflow run) need to be generated. Besides, hard-coding the directory structure either in the component description or in the workflow representation itself is cumbersome and drastically limits reusability.

Another approach is to use workflow provenance to keep track of the relations between the produced data. The user is then able to navigate in a graph whose vertices are files and edges relations between them [13]. For the user the name of a file and its path in the result directory tree makes more sense than the list of the inputs that were used to produce it. Navigating across large data provenance graphs is also difficult.

A higher-level approach relies on results annotation and metadata browsing [11]. This better captures the semantic of an experiment than a directory structure. One could then retrieve files based on requests like "Find anatomical-to-standard-brain transformations computed using X degrees of freedom". However, in a grid environment, metadata consistency issues coming from file move, deletion or replication are difficult to handle.

None of the existing solutions completely satisfy the need to provide alternative and user-friendly ways to organize the files generated by a workflow.

## 4 Performance comparison of workflow and monolithic implementations

Comparing the performance of workflow and monolithic implementations can be envisaged on two different use-cases, job farming and parameter sweep. *Job farming* corresponds to the iteration of the complete workflow on a set of input data whereas *parameter sweep* corresponds to the iteration of the workflow on a range of parameters, given a single dataset. We assume that job farming replicates the whole workflow and parameter sweep only repeats parts of it. This holds in the FSL *Feat* workflow presented in Section 2.

The *makespan* of a workflow is the considered performance metric. It denotes the total elapsed time between the submission of the first job of the workflow and the completion of the last one. The number of computing resources (denoted  $n_R$ ), the number of processed files ( $n_F$ ) and/or parameters ( $n_P$ ) are considered. In the following, we present some simulations with respect to those parameters, also studying the impact of data transfers and latency. They were conducted on the *Feat* FSL workflow presented in section 2 and performed by implementing a classical list scheduling algorithm. Resources are supposed to be homogeneous. Task execution times used in the simulations were measured on a Dual-Core AMD Opteron 2613.427 MHz with 3.5GB of RAM. Data transfer times have been measured between a cluster and an external Storage Element of the v1emed Virtual Organisation of the Dutch Virtual Laboratory for e-Science project<sup>9</sup>.

---

<sup>9</sup><http://www.v1-e.nl>

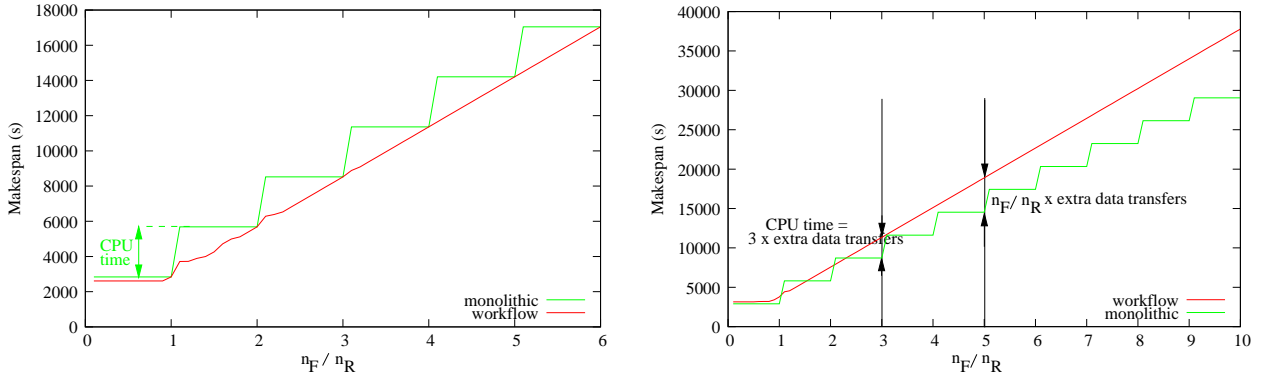


Figure 4: Simulation of the makespan of the *Feat* application for  $n_R = 10$  resources. Left: ideal case. Right: including data transfers.

#### 4.1 Simulations for the job-farming case

**Ideal case.** In the ideal case, data transfers are neglected and any free resource is supposed to be immediately available for computation (no latency). Under those assumptions, the workflow implementation cannot be less performing than the monolithic one, provided that a decent scheduling algorithm is implemented and neglecting the workflow management system overhead. Figure 4-left plots the evolution of the makespan of the *Feat* application with respect to the  $\frac{n_F}{n_R}$  ratio for  $n_R=10$  resources. The monolithic implementation exhibits a step shape. The height of each step is constant, and corresponds to the total CPU time to execute all the tasks in one run of the workflow (denoted  $T_{WF}$ ). The workflow version reaches linear speed-up from  $\frac{n_F}{n_R}=3$ . Both implementations perform the same for integer fractions  $\frac{n_F}{n_R}$ , in which case an optimal usage of resources is guaranteed. Apart from those particular cases, the workflow version clearly outperforms the monolithic one. Best gains are achieved for  $\frac{n_F+1}{n_R}$  and are upper-bounded by  $T_{WF}$ . This gain remains constant with respect to  $n_F$ . This performance improvement comes from the exploitation of the intrinsic parallelism in the workflow, in particular due to the registration steps. Note that finer granularity of the tasks also allows better scheduling.

**Impact of the data transfers.** Figure 4-right plots the evolution of the makespan of the *Feat* application taking into account data transfers and for  $n_R=10$ . This simulation has been obtained by increasing workflow tasks sizes by the duration of the data transfers they require, which assumes that data transfers are homogeneous among the grid nodes. Although not completely realistic, this assumption can be reasonable, in particular when considering production grids, where input/output data has to be fetched/registered from external data storage servers. Both the workflow and the monolithic implementations exhibit comparable performance for  $\frac{n_F}{n_R}$  in  $[0,3]$ . Beyond this interval, the monolithic version performs better. Extra data transfers coming from the workflow implementation can be visualised on the graph at integer values of  $\frac{n_F}{n_R}$ . In particular, we can notice at  $\frac{n_F}{n_R} = 3$  that the overhead coming from those extra data transfers is about a third of  $T_{WF}$ .

**Latency and data transfers.** Here latency is defined as the time needed to access a free CPU on the infrastructure. It includes for instance the submission and scheduling time, but excludes queuing time in case of a loaded infrastructure. Figure 5-left depicts the comparison of monolithic and workflow implementations taking into account both data transfers and various values of the latency and for  $n_R = 10$  resources. As it

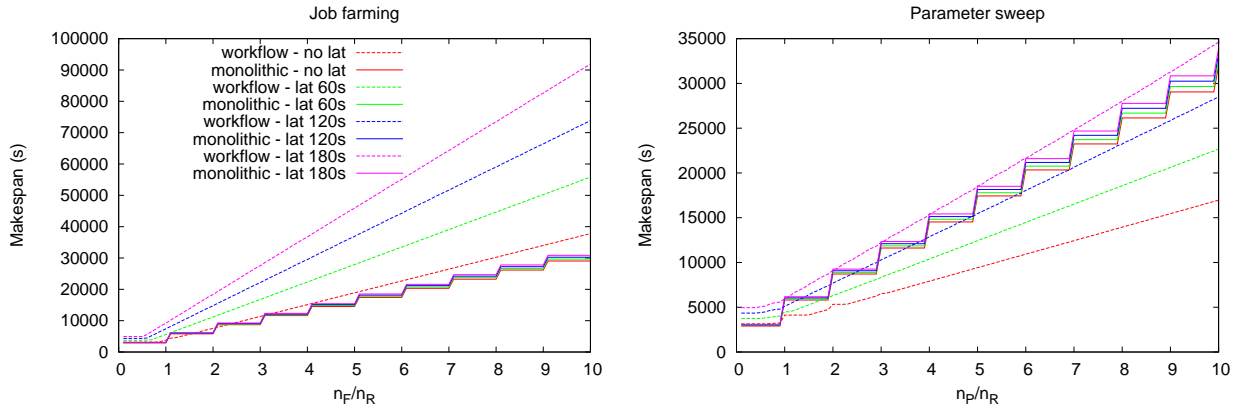


Figure 5: Simulation of the makespan of the *Feat* application, taking into account data transfers, for latency ranging from 0s to 180s. Colours represent different values of the latency. Left: job farming; workflow performance is crippled by data transfers and latency. Right: sweep over a model parameter; the workflow version outperforms the monolithic one for latency values lower than 3 minutes.

could be expected, the impact of the latency is much more dramatic on the workflow implementation than on the monolithic one. For a 1-minute latency (green curves), the performance of the workflow implementation is still comparable to the monolithic one for  $\frac{n_P}{n_R}$  in  $[0,1]$  while this range is  $[0,3]$  without latency (red curves). Then, the monolithic version rapidly outperforms the workflow, in particular at higher latency values (blue and pink curves). As already mentioned, we chose to adopt a quite fine granularity in this workflow, for reusability reasons. Coarser granularity would limit the impact of latency and data transfers, for example, by job grouping strategies as adopted in [5]. Yet, powerful strategies are still challenging to elaborate in the general case.

**Increasing CPU performance.** Increasing average CPU performance of the infrastructure certainly will affect the gain yielded by the workflow implementation. It can be noticed from left of figure 4 that a reduction of the CPU time  $T_{WF}$  will decrease the height of the steps in the monolithic curve, thus reducing the gap to the workflow red curve. Similar conclusion can be made when data transfers are taken into account: in this case, their impact would be increased by a improvement of the average CPU performance of the infrastructure.

#### 4.2 Simulation for the parameter sweep case

Figure 5-right plots a simulation of the makespan of the application for the parameter sweep use-case for  $n_R=10$  resources. It takes into account data transfers and represents the makespan for several different latency values (denoted by the different colours). In spite of the data transfers, the workflow implementation clearly surpasses the monolithic one when resources are accessible without any latency (red curves). Besides, this gain increases with respect to  $\frac{n_P}{n_R}$ . The performance of the workflow implementation then gets closer to the monolithic one when the latency value increases (green and blue curves). When the latency reaches 3 minutes (pink curves), the monolithic version starts to outperform the workflow one for all values of  $\frac{n_P}{n_R}$ .

### 4.3 Discussion

The fact that grid latency impacts job-farming much more dramatically than parameter-sweep comes from differences in the number of grid jobs implied in those use-cases. Adding a file to a job farming experiment (*i.e.* incrementing  $n_F$ ) leads to the submission of 30 additional grid jobs (1 per workflow processor) whereas adding a parameter to a sweep experiment (*i.e.* incrementing  $n_P$ ) only generates 10 jobs (1 per processor of the model computation sub-workflow on figure 1).

Figure 5 gives an idea about how the monolithic and the workflow implementations would compare on different kind of infrastructures. Red curves might correspond to clusters or small scale grids, and green to pink curves would denote grids with increasing sizes or loads. In practice, we commonly observe a one to two minutes latency on the infrastructure of the Dutch VL-e project. Thus, this simulation suggests that, in most practical cases, sweeping over a model parameter would benefit from a workflow implementation.

The simulation results presented here were computed only for  $n_R=10$  resources. A formal proof is out of the scope of this paper, however we can show that only the  $\frac{n_E}{n_R}$  ratio (or  $\frac{n_P}{n_R}$  for parameter sweeps) matters to the performance. Moreover, we can state that once the workflow makespan has reached linear behavior (as it is the case on figures 4 to 5), it will no longer deviate from it for larger values of  $\frac{n_E}{n_R}$ .

## 5 Conclusions

We presented a grid workflow implementing the *Feat* application of the FSL package. The description of this quite complex application was possible using a state-of-the-art workflow language, Scufi. The workflow implementation was validated on the grid infrastructure of the Dutch VL-e project.

This exercise was important to identify the pros-and-cons of workflow implementation for complex (image analysis) applications. Contrary to the expectations, simulations reveal that the performance improvement yielded by the workflow implementation in a job farming scenario is limited. Only a fixed gain is obtained on ideal grid infrastructures, and in practice, data transfers and latency rapidly take it down. Reducing the granularity of the workflow, for example by grouping of components, could be beneficial, however it is difficult to achieve in the general case. Conclusions are quite different when considering a model parameter sweep experiment, however. Despite the data transfers, the workflow implementation outperforms the monolithic script, even for large latencies.

This study also identified additional problems to be tackled. Firstly, the generality of this implementation is still limited. Various *Feat* use-cases require ad-hoc workflow modifications to implement changes *e.g.* in the number of contrasts or in the experimental set-up. This workflow rapidly produces thousands of files for a simple experiment, which makes intuitive output organization a challenging problem. Hard-coding a directory structure in workflow components is feasible, but it drastically limits the reusability of both the workflow and its components. We are currently investigating how to map data-level workflow provenance to a grid file catalog directory structure that is more intuitive to the end-user. Lessons learned from this study lead us to state that the results structure specification should be independent (i) from the workflow components, (ii) from the workflow description and (iii) from the workflow execution.

Thus, using workflows for such a widely scoped application is still challenging in terms of description, usability and performance. Hard-coded ad-hoc solutions can yield good benefits for some specific use-cases but further research to improve generality and reusability has to be considered. All in all, well designed (monolithic) applications such as FSL Feat are still hard to beat with generic workflow management systems.

Expected benefits of workflow management on the work style of neuroscientists make this challenge worth

considering though. Apart from the performance and usability items mentioned in this paper, one could forecast advantages ranging from data and knowledge management to experiment setup and execution. Data produced by a workflow could be automatically annotated, enabling queries such as "find any experiment conducted on brain region X considering fMRI task Y". Moreover, once analysis packages are described as workflows, customizing them to particular use-cases becomes accessible for end-users, who will then be able to conduct analysis experiments more autonomously.

## 6 Acknowledgements

This work was carried out in the context of the VL-e project, which is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ). We acknowledge Marie-José van Tol, Remi Soleman and Aart Nederveen for providing concrete large-scale detailed use-cases of fMRI analysis motivating this work. We also thank Kamel Boulebiar for smart technical developments related to gLite data management.

## References

- [1] R. Andrade et al. A Grid Framework for Non-Linear Brain fMRI Analysis. In *HealthGrid*, pages 299–305, Geneva, 2007.
- [2] D. Churches et al. A Parallel Implementation of the Inspiral Search Algorithm using Triana. In *Proceedings of the UK e-Science All Hands Meeting*, Nottingham, UK, Sept. 2003.
- [3] E. Deelman et al. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1(1):9–23, 2003.
- [4] K. Friston. *Statistical parametric mapping and other analysis of functional imaging data*, pages 363–385. AcademicPress, 1996.
- [5] T. Glatard et al. A Service-Oriented Architecture enabling dynamic services grouping for optimizing distributed workflows execution. *Future Generation Computer Systems*, 24(7):720–730, 2008.
- [6] T. Glatard et al. Flexible and efficient workflow deployment of data-intensive applications on grids with MO-TEUR. *International Journal of High Performance Computing and Applications (IJHPCA)*, 22(2), 2008.
- [7] R. Goebel. *BrainVoyager: Ein Programm zur Analyse und Visualisierung von Magnetresonanztomographiedaten*. Göttingen: Gesellschaft für wissenschaft, 1997.
- [8] T. Oinn et al. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics journal*, 17(20):3045–3054, 2004.
- [9] S. Olabarriaga et al. Parameter Sweeps for Functional MRI Research in the Virtual Laboratory for e-Science Project. In *Workshop on Biomedical Computations on the Grid (Biogrid'07)*, pages 685–690, Rio de Janeiro, May 2007.
- [10] D. Rex et al. The LONI Pipeline Processing Environment. *NeuroImage*, 3(19):1033–1048, July 2003.
- [11] N. Santos et al. Distributed Metadata with the AMGA Metadata Catalogue. In *Workshop on Next-Generation Distributed Data Management (HPDC'06)*, Paris, France, June 2006.
- [12] S. Smith et al. Advances in functional and structural MR image analysis and implementation as FSL. *NeuroImage*, 23(1):S208–S219, 2004.
- [13] J. Zhao et al. Using Semantic Web Technologies for Representing e-Science Provenance. In *Third International Semantic Web Conference (ISWC2004)*, pages 92–106, Hiroshima, Nov. 2004.