Accelerating Medical Research using the Swift Workflow System

Tiberiu STEF-PRAUN, [b,1], Benjamin CLIFFORD [b], Ian FOSTER [a,b], Uri HASSON [b], Mihael HATEGAN [b], Steven L. SMALL [b], Michael WILDE [a,b], Yong ZHAO [a,b]

[a] *Argonne National Labs*
[b] *University of Chicago*

Abstract. Both medical research and clinical practice are starting to involve large quantities of data and to require large-scale computation, as a result of the digitization of many areas of medicine. For example, in brain research – the domain that we consider here – a single research study may require the repeated processing, using computationally demanding and complex applications, of thousands of files corresponding to hundreds of functional MRI studies. Execution efficiency demands the use of parallel or distributed computing, but few medical researchers have the time or expertise to write the necessary parallel programs.

The Swift system addresses these concerns. A simple scripting language, SwiftScript, provides for the concise high-level specification of workflows that invoke various application programs on potentially large quantities of data. The Swift engine provides for the efficient execution of these workflows on sequential computers, parallel computers, and/or distributed grids that federate the computing resources of many sites. Last but not least, the Swift provenance catalog keeps track of all actions performed, addressing vital bookkeeping functions that so often cause difficulties in large computations.

To illustrate the use of Swift for medical research, we describe its use for the analysis of functional MRI data as part of a research project examining the neurological mechanisms of recovery from aphasia after stroke. We show how SwiftScript is used to encode an application workflow, and present performance results that demonstrate our ability to achieve significant speedups on both a local parallel computing cluster and multiple parallel clusters at distributed sites.

Keywords. Brain research, Grid Computing, Workflows

1. Introduction

Abundant examples exist of highly computational medical research in domains of vital importance, and of infrastructures focused on supporting such research [8,13]. We describe and advocate here the use of Grid computing technologies toward this end, and present a case study in which these tools are applied to the needs of medical research.

An attractive approach to enhancing the productivity of medical research is to use existing Grid infrastructure. There are several good reasons for this, the most important being its ready availability through global cyberinfrastructure, and the substantial capa-

[1] Corresponding Author: Tiberiu Stef-Praun, 5640 S. Elllis rm. 405, University of Chicago, 60615, IL, USA

bilities that can be employed by Grid users. The Grid can be seen as a large, distributed computing resource used in common by a wide group of scientists. From the end-user's point of view, the Grid is a powerful, multi-user computer, with familiar resource sharing and user access mechanisms. Grid resources are shared according to specified policies or may be reserved upon request. The security of each user's applications and data is enforced by standard Unix permissions and by enhanced access control list security. There are storage services, large scale execution services, easy and efficient data movement utilities, and supporting tools that allow users to take advantage of large amounts of computing power. These capabilities can conveniently address specific requirements of medical research, such as access control to patient data, as mandated by HIPAA rules in the USA; high bandwidth to rapidly transfer large DICOM images from the patient's records [5]; and sophisticated image analysis algorithms to aid in the interpretation of medical conditions.

We describe our success to date in applications in imaging-based neurological research in which we seek to expand the scope and scale of our computing capabilities to study the neurological mechanisms of the process of recovery from aphasia due to stroke. The neuroscience behind the system described below is based on work in permutation tests for clustering analysis [1,11]. The aphasia recovery study (described below) involves the processing of many large functional MRI files through a set of domain-specific applications, and re-running a statistical analysis protocol over a large parameter space.

The benefits of using the Swift workflow system (described in detail below) for this application include large reductions in the data and computing resource management effort that is typpically required in modern scientific research. This is achieved by automating otherwise manual and labor-intensive processes. In addition to providing transparent and on-demand access to Grid resources, our workflows also exhibit reproducibility and provenance tracking of the data results, thus enabling collaboration in the actual research process, not only in sharing the results. The sharing and reuse of the actual research processes, as well as of the data, has already shown benefits in several other domains (e.g., physics, sociology, economics) in which we also have ongoing collaborations.

2. Grid Infrastructure

The purpose of the Grid software stack is to hide the complexity of the vast resources being made available to the users. In a sense this software can be seen as an operating system that provides the user with the desired functionality of transparent distributed data access and application execution.

In large distributed systems, the granularity of the atomic actions changes in scale: instead of CPU-based operations, users describe their work in terms of application invocations. They use existing applications and existing data, customize them, and iterate over them until the sought-after analysis results are obtained. At the next level, the distributed system middleware consists of software that transparently manages the execution of applications on computers that are typically managed by local resource management systems such as Condor [9] or PBS [17]. On top of these computing clusters, the Grid software provides an access layer that gives users uniform interface to distributed resources, and provide tools to manage data transfers, applications execution, and many other features that hide the heterogeneity of resources across distributed clusters. The

current standard for homogenizing the resources is described by the Open Grid Services Architecture (OGSA), and implemented by the Globus Toolkit, among others. At this level of the Grid software stack, users can access remote data and execute remote applications, but they still have to be aware of the networked nature of Grids, and must manage the executions of their applications. A higher-level component that abstracts the distributed nature of Grid resources is the Swift system [7], which maps "virtual local applications" to their corresponding physical installation on remote Grid sites.

2.1. An Application User's View

Scientific researchers do not, in general, want to be aware of computing infrastructure: they want easy to use, high-performance applications that deliver fast and accurate results with little effort, and minimal disruption of their scientific thought process. This dictates that their applications must be reliably and transparently allocated the resources necessary to solve the problems at hand. Our solution to making tools available on the Grid in a fashion which hides the complexities of manually managing different computers is to *virtualize the location* of the tool through a level of indirection. This indirection is implemented in Swift in an internal directory consisting of the application and computing-site descriptors. With Swift, Grid users can consider their applications to be virtually local to them: the selection of the site to execute the application and the transfer of input and output data and parameters to the site is handled transparently by Swift.

In addition to the transparency of accessing the tools, researchers benefit from the ability to create complex functionality by composing simple tools that each solve some subproblem of the researcher's agenda. These gains are further enhanced by the expressivity of SwiftScript, which allows such complex algorithms to be expressed in clear, simple, and high-level logical terms, rather than in low-level physical details.

Treating the existing applications that the scientists use as the *atomic computation units* of the workflow describing the algorithm, we have built a workflow execution engine around the concept of *data flow analysis*: whenever the input data for an atomic computation unit in the workflow becomes available, the engine selects a Grid resource and sends out the computation and the data to that site. At the end of the computation, the engine copys back the results, to make them available to subsequent workflow steps that depend on this result as their input, or copies them to a repository for archival, dissemination, or later analysis.

2.1.1. SwiftScript Language Constructs

SwiftScript extends the earlier Virtual Data Language [7] with support for dataset typing and mapping, dataset iteration, conditional branching, sub-workflow composition, and other advanced features. SwiftScript support for standard programming language control constructs make it easy to script the execution of applications and thus to automate the research process. For example, loops are used to iterate through parameter sweeps, mappers to associate inputs and outputs with actual file names, and arrays to store groups of similar datasets. We describe in the implementation section below how SwiftScript can be used to naturally and effectively express workflows describing neuroscience research tasks.

3. Aphasia and Brain Research Tools

Stroke, in addition to being the third highest cause of death in the United States, is the leading cause of disability among adults. (American Heart Association. (2003). 2003 Heart and Stroke Statistical Update. Dallas, Texas: American Heart Association.) Thus there is intense interest in the clinical research community in understanding the neurological mechanisms involved in recovery from stroke. One such research area, in which we are involved, specifically focuses on analysis of the recovery phase from *aphasia caused by stroke,* and effects on the neurobiological aspects of the patient. In our aphasia-recovery studies, we apply fMRI to analyze neural activity (BOLD response) in the brains of subjects after stroke, in response to various congnitive stimuli.

3.1. The Research Problem

The medical research behind testing the SwiftScript workflow technology on the grid is the study of the stroke recovery process in a set of patients. The study uses fMRI brain images data of the patients subjected to various stimuli to detect neural activations in the brain as a result of the experimental conditions. However, given the practically limited number of patients available for a typical imaging study, the results of the activation detection process are likely to suffer from the uncertainty of random brain activations. Thus, besides the actual activation detection, the research plan also contains of a verification phase to analyze the validity of the results. This step involves assessment of a null hypothesis about the results obtained from the experiment's data using random modifications of the original fMRI readings.

3.2. The Scientific Methodology

In fMRI studies, data are sampled from spatial locations in a resolution measured in voxels. Statistical analysis in a typical experiment with two conditions (e.g., viewing circles vs. viewing faces) is based on the following steps:

1. Spatially align all the brain images from an experimental run.
2. For each subject, for each voxel, establish the activity level (BOLD response) for each condition (2 data points), and save the difference in activity (delta).
3. At the group level, analyze these delta values to establish, for each voxel, whether subjects' deltas differ from zero. This is performed by calculating whether the delta vector for each voxel (of length N = number of subjects) has a mean that is reliably greater than 0, using a t-test (i.e., testing if the two conditions differ reliably).
4. on the group level: Once we establish for each voxel whether there is a reliable difference between the two conditions, find reliable clusters of activity.

Because there are many thousands of voxels in our brain images, some would be "active" just by chance (e.g., if data were randomly sampled). The permutation algorithm identifies which clusters of neural activity are not likely to be found by chance. In brief, the method tests the null hypothesis that the clusters of activation found in the dataset are indeed likely to be found by chance. The null hypothesis asserts that if we were to "switch" the labels of the conditions for one or more participants, and calculate the delta values in each voxel, we would get equally large activations. To test this null hypothesis,

for one or more participants (in all possible combinations), we interchange the labels of the two conditions, re-calculate the reliability of delta in each voxel (step 3), and evaluate the clusters we find. If the clusters in our data are greater than the majority of the clusters found in the permutations, then the null hypothesis is refuted and we conclude that the clusters of activity found in our study are not likely to be found by chance.

4. Grid Implementation

We coded the algorithm described above in SwiftScript and then installed on the Grid the software applications that were previously used on desktop workstatins to solve the original problem.

In the aphasia-recovery study, the main tools used were the *R* [14] Statistical Package, used to generate the data for the null-hypothesis testing, and the *SUMA* [16] tool, part of the AFNI [2] package, for computing the clustering of neural activity levels.

The input files can be separated into two classes. The first group consists of experiment-dependent inputs, such as the files that contain the brain activity measurements from the experiments (the `origBrain` file). The second group consists of files that are required by the tools involved in the processing, such as the full standard brain files `brainFile, specFile` needed by AFNI to map the experimental measurements.

There is a special set of files which result as a by-product of the data-processing focus of the SwiftScript workflow language. These are the intermediary files, that are produced by the application components that make up the final workflow, and which are being fed as inputs to the subsequent blocks in the workflow. In the example below, they have names like `randomBrain, randomCluster, dsetReturn, clusterThresholdsTable`.

Following the *location virtualization principles* described earlier, these file names are *mapped transparently* from real files that exist on the computer running the workflow to logical names that the SwiftScript program uses to describe the workflow data entities.

4.1. SwiftScript Representation of the Aphasia Algorithm

The SwiftScript description of the algorithm first defines the data types of each dataset (file) that participates in the workflow. For clarity, we define a unique type for each file containg syntactically and/or semantically different kinds of data:

```
type file {}
type fileNames{ file f[]; }
type script {}
type brainMeasurements{}
type precomputedPermutations{}
type fullBrainData {}
type fullBrainSpecs {}
type brainDataset {}
type brainClusterTable {}
type brainDatasets{ brainDataset b[]; }
type brainClusters{ brainClusterTable c[]; }
```

Having defined the types of the data entities in the workflow, we define the procedures that process the input files. Some procedures serve as interface wrappers for external programs, and map the input and output parameters used in the SwiftScript workflow to the actual physical arguments of the application program.

```
// Procedure to run R statistical package
(brainDataset t) bricRInvoke (script permutationScript, int iterationNo,
               brainMeasurements dataAll, precomputedPermutations dataPerm){
       app { bricRInvoke @filename(permutationScript) iterationNo
                         @filename(dataAll) @filename(dataPerm); }}

// Procedure to run AFNI Clustering tool
(brainClusterTable v, brainDataset t) bricCluster (script clusterScript,
int iterationNo, brainDataset randBrain,
fullBrainData brainFile, fullBrainSpecs specFile) {
       app { bricPerlCluster @filename(clusterScript) iterationNo
                             @filename(randBrain) @filename(brainFile)
                             @filename(specFile);}}

// Procedure to merge results based on statistical likelhoods
(brainClusterTable t) bricCentralize ( brainClusterTable bc[]) {
       app { bricCentralize @filenames(bc); }}

(brainDataset t) makebrain (brainDataset randBrain,
brainClusterTable threshold, fullBrainData brain,
fullBrainSpecs spec){
        app {   makeBrain @filename(randBrain) @filename(threshold)
                          @filename(brain) @filename(spec); }}
```

Other procedures use more complex language constructs such as iterations and conditional constructs to combine several atomic application invocations.

```
// Procedure to iterate over the data collection
(brainClusters randCluster, brainDatasets dsetReturn) brain_cluster (
fullBrainData brainFile, fullBrainSpecs specFile) {
    int j[]=[1:2000];
    brainMeasurements     dataAll<fixed_mapper; file="obs.imit.all">;
    precomputedPermutations dataPerm<fixed_mapper; file="perm.matrix.11">;
    script                  randScript<fixed_mapper; file="script.obs.imit.tibi">;
    script                  clusterScript<fixed_mapper; file="surfclust.tibi">;
    brainDatasets           randBrains<simple_mapper; prefix="rand.brain.set">;
    foreach int i in j {
        randBrains.b[i] = bricRInvoke(randScript,i,dataAll,dataPerm);
        brainDataset rBrain=randBrains.b[i];
        (randCluster.c[i],dsetReturn.b[i]) =
            bricCluster(clusterScript,i,rBrain, brainFile,specFile);
} }
```

Having declared the data types and the procedures that will process the data, we must define the dynamic mapping of the logical file names used in SwiftScript to actual on-disk file resources. This mapping can range from simple name-to-file mapping to database-select operations or the matching of multiple files by a regular expression, based on the choices available in an extensible library of mapper implementations.

```
fullBrainData       brainFile<fixed_mapper; file="colin_lh_mesh140_std.pial.asc">;
fullBrainSpecs      specFile<fixed_mapper; file="colin_lh_mesh140_std.spec">;
brainDatasets       randBrain<simple_mapper; prefix="rand.brain.set">;
```

```
brainClusters       randCluster<simple_mapper; prefix="Tmean.4mm.perm",
                                suffix="_ClstTable_r4.1_a2.0.1D">;
brainDatasets       dsetReturn<simple_mapper; prefix="Tmean.4mm.perm",
                                suffix="_Clustered_r4.1_a2.0.niml.dset">;
brainClusterTable   clusterThresholdsTable<fixed_mapper; file="thresholds.table">;
brainDataset        brainResult<fixed_mapper; file="brain.final.dset">;
brainDataset        origBrain<fixed_mapper; file="brain.permutation.1">;
```

The actual workflow consists simply of invocations of the high-level procedures defined above:

```
// Main program: launches the entire workflow
(randCluster, dsetReturn) = brain_cluster(brainFile, specFile);
clusterThresholdsTable= bricCentralize (randCluster.c);
brainResult=makebrain(origBrain,clusterThresholdsTable,brainFile,specFile);
```

Note that this simple description, at which level most researchers will work, enhances productives by abstracting and automate many complex tasks. For the scientific research that we described above, the two thousand invocations of the block in the `braincluster` function were determined individualized processing of the `bricRInvoke` function, depending on the parameter `i`. Also, the workflow performs automatic synchronization of the many subtasks involved, waiting for the result of these two thousand executions to finish before continuing with the merging (`makebrain`) procedure.

4.2. The Swift Environment, and Grid Application Deployment

For completeness, we summarize the additional infrastructure that enables the transparent execution of the workflow described above. The application components (containing the problem-solving algorithms) that are invoked in the workflow, must be installed at the sites that are to be involved in the computation. This step is generally done once, as part of application deployment. In our case we installed the applications `bricRInvoke, bricPerlCluster, bricCentralize,` and `makebrain` at several sites, which we recorded in Swift catalogs. Swift uses these catalog entries to choose on which sites, and to what degree of parallelism, to invoke the applications.

Internally, Swift uses Globus [6] software for important functions such as authentication with remote sites, data transfer, and remote task invocation. We run our applications on several sites spanning the Teragrid, Open Science Grid, and independent institution clusters. The architecture of the infrastructure involved in executing one's workflow on the Grid is depicted in Figure 1.

Other Swift facilities allow the user to resume the workflow from the point of any failure, to cluster short-running applications for more efficient remote execution, and to visualize the progress of workflow execution. Figure 2 shows a snapshot of the executing workflow.

5. Results

5.1. Benefits of Grid Computing in Health-related Research

To measure the benefits of using workflow systems to manage research data analysis in Grid environments, we recorded the execution time of the same workflow instance in

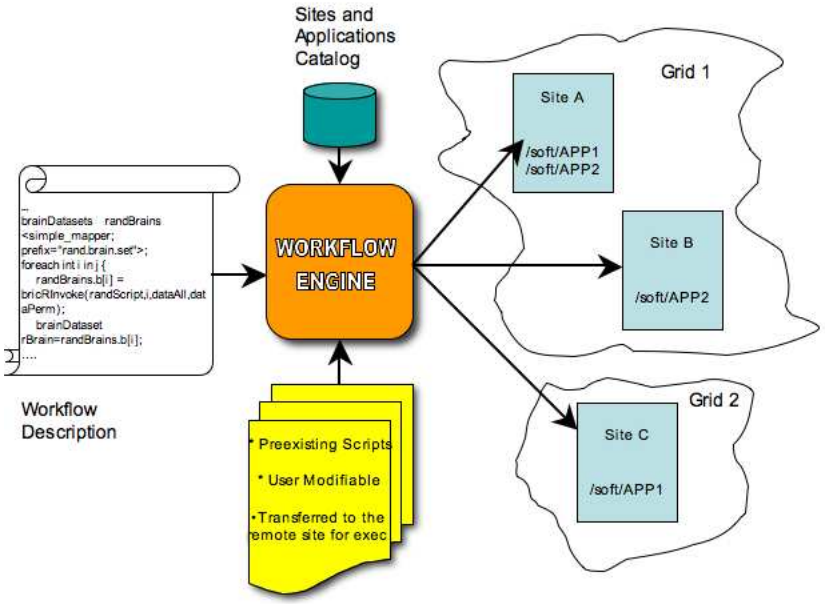

Figure 1. The components of a workflow based application

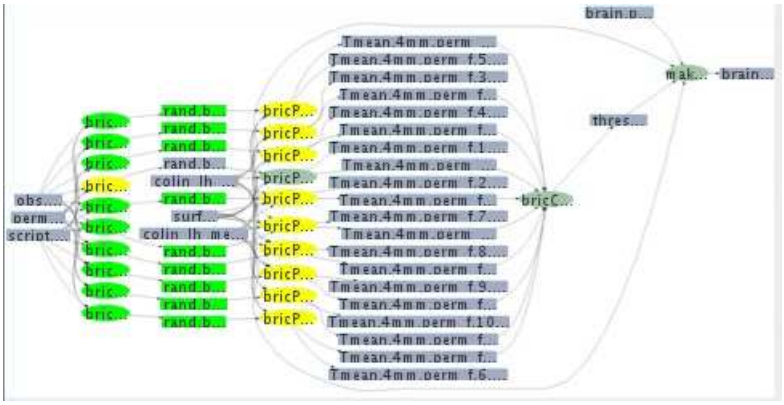

Figure 2. The execution of a small subset of the aphasia study analysis workflow. The colors of the boxes indicate if the task has completed (green) or is executing (yellow). Lines represent data dependencies.

both a local workstation and a distributed Grid environment, and provide initial results in Table 1 below. The performance gains depend primarily on the parallelism that the workflow exhibits (in this case we had two thousand parallel execution threads), on the available number of sites that could execute the applications that made up our workflow (in our case, three sites), and the number of simultaneous jobs executed by those sites (which depends in turn on local cluster sizes, on cluster resource management policies, and on contention on the cluster from the other users that share them).

Other than speedup results, the current implementation allows the researcher to modify the scripts that are used in this workflow, as we chose a model where these scripts,

Table 1. Timing measurements for executing grid versus local execution of the aphasia workflow

Local	Grid
3 min/1 job instance	5 min / single instance run on the Grid
300 min / 100 job instances	50 minutes / 100 job instances

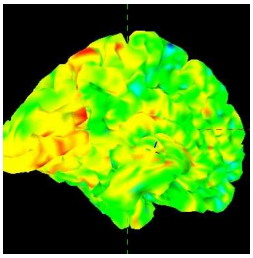

Figure 3. An intermediate stage of activation analysis as processed by the workflow

containing the actual scientific procedures, are deployed on demand, dynamically to the Grid sites.

While we used only three sites in this study, we could increase that number to improve the workflow's speedup significantly. We note a major benefit if this approach here: other researchers using the same tools that were used in this work could readily use the already deployed applications that we used as well (SUMA, R), or simply obtain the current workflow definition and execute it without the need of any special setup. Swift also allows us to visualize both the workflow' s execution (Figure 1) and the "real-time" display of the activations on the brain, displayed in Figure 3.

6. Related Work

Swift has its origins in the GriPhyN Virtual Data System (VDS) [7], originally designed to automate the analysis of the large quantities of data produced by high energy physics experiments. Another VDS component, Pegasus [4], implements specialized strategies for scheduling tasks on computing sites.

Much work on workflow for eScience has focused on the orchestration of web service invocations, as supported, for example, by BPEL and by Taverna [12], which implements a BPEL subset. Kepler [10] is used for similar purposes. We view Swift as addressing a different problem than these systems, namely the orchestration of large numbers of calls to application programs, and their practical and transparent execution in a distributed Grid.

GenePattern [15], like Swift, focuses on the composition of application programs. It differs in its graphical programming approach, and its lack of support for large-scale parallel processing. Google's MapReduce [3], like Swift, focuses on the large-scale analysis of large quantities of data. Swift differs in its support, via XDTM, of diverse file system structures, and its support for task-parallel as well as data-parallel execution.

7. Summary

We have introduced a tool, Swift, that supports the parallel and distributed execution of computationally demanding and data-intensive scientific computations. Using an example from a clinical study of aphasia recovery, we have described how Swift allows (via its scripting language, SwiftScript) for the concise representation of complex algorithms, for the efficient execution of those algorithms on parallel and distributed ("grid") computing systems, and the subsequent exploration and assessment of the workflow's execution history.

References

[1] E. T. Bullmore, J. Suckling, S. Overmeyer, S. Rabe-Hesketh, E. Taylor, and M. J. Brammer. Global, voxel, and cluster tests, by theory and permutation, for a difference between two groups of structural mr images of the brain. *IEEE Transactions on Medical Imaging*, 18(1):32–42, 1999.

[2] Robert W. Cox. Afni: software for analysis and visualization of functional magnetic resonance neuroimages. *Comput. Biomed. Res.*, 29(3):162–173, 1996.

[3] J Dean and S Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[4] E Deelman, J Blythe, Y Gil, C Kesselman, G Mehta, M Su, K Vahi, and M Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, 2004.

[5] S.G. Erberich, M. Bhandekar, A. Chervenak, M.D. Nelson, and C. Kesselman. Dicom grid interface service for clinical and research pacs: A globus toolkit web service for medical data grids. *DICOM grid interface service for clinical and research PACS: A Globus toolkit web service for medical data grids. International Journal of Computer Assisted Radiology and Surgery*, 1, 2006.

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 2001.

[7] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. *International Conference on Scientific and Statistical Database Management*, 2002.

[8] K. K. Kakazu, L. W. Cheung, and W. Lynne. The cancer biomedical informatics grid (cabig): pioneering an expansive network of information and tools for collaborative cancer research. *Hawaii Medical Journal*, 63, 2004.

[9] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[10] B. Ludäscher, I Altintas, C Berkley, D Higgins, E Jaeger, and M Jones. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10), 2005.

[11] T. E. Nichols and A. P. Holmes. Nonparametric permutation tests for functional neuroimaging: A primer with examples. *Human Brain Mapping*, 15(1), 2002.

[12] T Oinn, M Greenwood, M Addis, M. N Alpdemir, J Ferris, K Glover, C Goble, A Goderis, D Hull, D Marvin, P Li, P Lord, M R. Pocock, M Senger, R Stevens, A Wipat, and C Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 8(10), 2005.

[13] J. Phillips, R. Chilukuri, G. Fragoso, D. Warzel, and P. A. Covitz. The cacore software development kit: Streamlining construction of interoperable biomedical information services. *BMC Med Inform Decis Mak*, 6(2), 2006.

[14] R Development Core Team. *R: A Language and Environment for Statistical R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005.

[15] M. Reich, T. Liefeld, J. Gould, J. Lerner, P Tamayo, and JP. Mesirov. Genepattern 2.0. *Nature Genetics*, 38(5):500–501, 2006.

[16] Z.S. Saad, R.C. Reynolds, B. Argall, S. Japee, and R.W. Cox. Suma: an interface for surface-based intra- and inter-subject analysis with afni. In *Biomedical Imaging: Macro to Nano, 2004. IEEE International Symposium on*, volume 2, pages 1510– 1513. Dept. of Health and Human Services, Nat. Inst. of Health, Bethesda, MD, USA;, 2004.

[17] OpenPBS team. http://www.openpbs.org/.