Medical Image Processing Workflow Support on the EGEE Grid with Taverna

Ketan Maheshwari
University of Nice – Sophia Antipolis
CNRS-I3S Laboratory
06903 Sophia Antipolis, France
ketan@polytech.unice.fr

Paolo Missier
University of Manchester
School of Computer Science
Oxford rd, Manchester, M139PL, UK
pmissier@cs.man.ac.uk

Carole Goble
University of Manchester
School of Computer Science
Oxford rd, Manchester, M139PL, UK
carole@cs.man.ac.uk

Johan Montagnat
University of Nice – Sophia Antipolis
CNRS-I3S Laboratory
06903 Sophia Antipolis, France
johan@i3s.unice.fr

Abstract

Resource-intensive and complex medical imaging applications can benefit from the use of scientific workflow technology for their design, rapid implementation and reuse, but at the same time they require a Grid computing infrastructure to execute efficiently. In this paper we describe a technical architecture that bridges the gap between the Taverna workflow management system and the EGEE grid infrastructure. This is achieved through a novel Taverna gLite activity plugin that makes the interface between the multi-threaded, centralized workflow enactor and the massively distributed, batch-oriented grid infrastructure. The plugin significantly increases the performance of Medical Imaging workflows over an equivalent plain Taverna workflow.

1 Introduction

Workflows are commonly used to execute data- and compute-intensive medical-image analysis pipelines. Such workflows are typically composed of legacy image analysis software components interconnected through data dependencies. Although a number of scientific workflow systems support this task, many medical data analysis pipelines require large-scale Grid computing infrastructures to cope with their high compute and storage requirements. In this paper we focus on the combination of a specific workflow manager, i.e., Taverna [8], and a specific Grid infrastructure, namely EGEE (Enabling Grids for E-sciencE)[1]. We show how the flexibility in pipeline design provided by Taverna can be combined with the computational resources of EGEE, in order to efficiently support a variety of scientific experiments that involve medical-image processing. We expect the resulting architecture to benefit both the Taverna user community and the medical image analysis community, as it provides a new Grid access functionality to Taverna, at the same time increasing the usability of the EGEE infrastructure. Section 2 presents an overview of the Taverna workbench, the EGEE infrastructure and its gLite middleware capabilities. The core software component developed to bridge Taverna and EGEE is an "activity" plugin to the Taverna Workbench that interfaces to EGEE's gLite middleware. Its design and implementation are discussed in Section 3. The various error conditions encountered in a complex grid environment are countered by implementing strong fault tolerance mechanisms. In addition to the workflow enactment provided by Taverna, this enables users to: (1) fit their legacy executable/binaries and port them to the Grid, and (2) accomplish secure and transparent application-data transfer to and from the Grid. We evaluate the plugin through our cardiac image processing application use-case in Section 4. Related work is finally discussed in Section 5.

[1] http://www.eu-egee.org/

2 Tools and Environments

2.1 Taverna Workbench

The Taverna Workflow Management System[2] is an open source scientific dataflow manager developed by the ^{my}Grid consortium in the UK. A first version of Taverna was released in 2004, and has since enjoyed a broad adoption within the e-science community[3]. Taverna includes a GUI-based rich-client workbench for workflow design and a workflow enactment engine. A Taverna workflow consists of a collection of processors connected by data links, which establish a dependency between the output(s) of a processor and the input(s) of another. The processors are of different kinds depending on the application code to be invoked: typically, processors are Web Services, with input and output *ports* that correspond to the operations defined in the service's WSDL interface. Java classes and local scripts can be used as workflow processors, as long as they expose their signature as input and output ports as required by the model. An additional gLite processor kind has been defined in the context of this work. The engine orchestrates the execution of the processors in a way that is consistent with the dependencies, and manages the flow of data through the processors' ports. The overall workflow execution is data-driven, with a push model: a processor's execution is started as soon as all of its inputs are available. A new version of the Taverna workbench is released as Taverna version 2, henceforth referred to simply as *T2*. Some of the new features of T2 are directly relevant to the work described in this paper, namely:

A separation of the data space from the process space within the engine, whereby all the data involved in the execution is managed by a Data Manager through an external database, and is addressed by the engine only by reference. This facilitates the management of data-intensive workflows in a scalable way, as high volume data is only transferred into the engine space when needed by processors that are unable to resolve external data references on their own. The cardiac imaging application described in Section 4.1, which involves a potentially high data volume, is an example.

Extensions points in the processor's invocation sequence, which can be programmed to enhance the performance of some of the processors. We exploit this feature by effectively programming a T2 processor to create jobs and inputs/outputs definitions for the gLite middleware.

[2]Taverna, http://www.mygrid.org.uk/tools/taverna/

[3]In 2008, ^{my}Grid has estimated the user base at about 350 organizations with 1000+ known users.

A plugin architecture for extending the engine's functionality, through a Service Provider Interface (SPI). In this work we have used SPI-based extensions to support the gLite job management lifecycle. **T2's data parallelism** offers two modes of workflow execution: 1) A single-stage pipeline, and 2) an advanced data parallelism mode. The data is pushed to each processor only for one executing instance in single-stage pipeline. In the advanced data parallelism mode multiple instances of execution are realized which are only limited by the number of data-items available.

T2 is designed to support the key phases of the e-science lifecycle, namely (1) the discovery and reuse of services and prior experiments, (2) assembling and encoding new experiments as dataflow, (3) executing the dataflow and monitoring its progress, (4) analyzing its results (primarily by capturing and querying information-rich provenance logs), and finally (5) sharing both services, dataflow, and results. By feeding data and processes back into a pool of reusable knowledge, the last step enables other scientists to undertake new in-silico experiments.

2.2 The EGEE Framework

EGEE is a premiere grid infrastructure for conducting e-science in Europe. It has diverse capabilities for data and computation management. Currently, the EGEE infrastructure supports 125 Virtual Organizations (VO) comprising of 9000 users across 50 countries. Approximately 20 petabytes storage and 80000 cores are at their disposal. EGEE facilitates the job management lifecycle. The EGEE infrastructure is composed of various components working in coordination at logical and physical level. The data storage of EGEE is realized in the form of *Storage Elements (SE)* that facilitate the physical storage. A Logical File Management System along with a cataloging service maintains the data on these SEs. The compute facilities are provided by the *Compute Elements (CE)*, which in most cases are a cluster of many *Worker Nodes (WN)*. A *Resource Broker (RB)* schedules the jobs to appropriate job queues based on the job requirements parsed as Job Description Language (JDL) description. The *Logging and Bookkeeping (LB)* component manages the state of jobs and job-queues in the system.

The gLite Middleware supports batch-oriented experiment execution management in the form of a managed job submission system. A simple JDL is used to construct the specification of a job to be submitted. Jobs are submitted to the queues based on the matching execution times and the average waiting time

for that queue. The job life cycle is handled without notification to the user who has to periodically poll the system to discover the progresses made. The data transfer between the user's localhost and the grid nodes is performed with secure protocols such as sFTP and gridFTP. Language bindings including C++ and java APIs are available in order to programmatically manage jobs.

3 The T2 gLite Plugin

A T2 gLite plugin was developed in order to interface the T2 workbench with the EGEE grid infrastructure. This plugin enables a T2 user to submit jobs to EGEE via gLite middleware in the form of workflow tasks. Job execution states are periodically checked and results are collected to localhost at the successful completion of each task. These tasks are enacted in parallel and run asynchronously over the EGEE infrastructure. The design and implementation of the plugin are motivated by the following goals: 1) Leverage the powerful data parallel modes of T2 workflow enactment, 2) Execution of multiple tasks simultaneously over the batch-oriented EGEE infrastructure and 3) Robust and fault-tolerant interface which can handle various error conditions commonly encountered in a complex Grid environment.

3.1 Design Considerations

The T2 workbench is a centralized multi-threaded execution environment while the EGEE grid provides a massively distributed computing infrastructure running isolated batch processes without progress notification mechanism. The former is an event-driven lightweight tasks orchestrator while the latter deals with compute intensive user jobs through an active polling mechanism. The main design challenge constitutes of coupling both environments. The gLite plugin has to transform asynchronous T2 calls to the EGEE poll-based jobs management system and notify back the T2 core on batch execution completion. Figure 1 shows a schematic of the interaction between T2 and EGEE via the gLite plugin. The T2 engine enacts the gLite processor which in turn contacts the gLite middleware. The plugin prepares the job, then submits it to EGEE, obtains a jobID and polls for the job status in an independent thread until its completion. As T2 activities are naturally parallelized whenever possible, other activities (with no data dependencies on the EGEE job) can be executed asynchronously in the meanwhile. Upon job completion, the results are sent to the T2 data manager. Throughout this interaction, the T2 engine remains unaware of where the job has been executed. Different types of processors can be mixed within a single workflow. In terms of performance, the choice of executing a fraction of local versus remote processes may have a very significant impact.

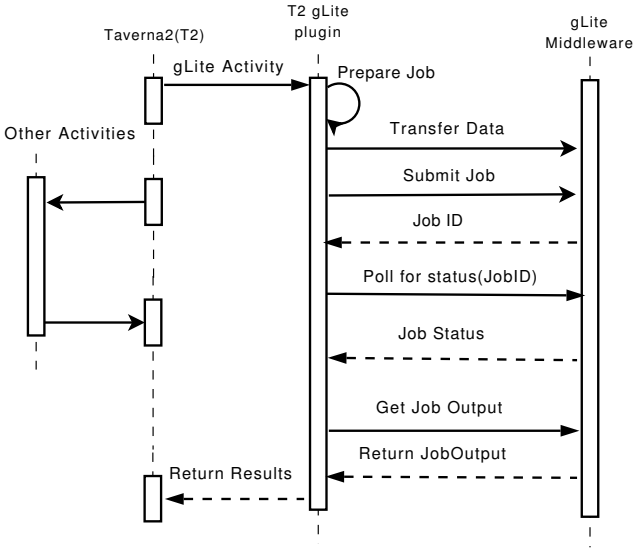

Figure 1. Interaction of the T2 gLite Plugin with gLite during job submission

Security and Confidentiality In order to successfully submit a job to EGEE, the user must provide a valid VO membership and proper authentication credentials, which must be delegated to the WMS-proxy service endpoint. This is usually done once and a credential may need to be renewed for long running workflows.
Medical images are often anonymized or pseudonymized prior to grid transfer and execution to fulfill medical privacy requirements. DICOM images in particular may be anonymized by clearing a list of well identified header fields. For data with stringent security requirements, gLite proposes a file encryption keystore and API which enables the encryption of files for storage and transfer (gLite Hydra service). Files are decrypted, if the user is authorized to access the file keys, on the worker node processing the files only. There also exists an EGEE Medical Data Manager (MDM) that encompasses DICOM files handling, including files registration, headers protection in secured database and data files encryption [6]. However, the plugin described does not address the problem of resources authorization and it is dealt with by the gLite middleware. This work

makes the assumption that a securely accessible gLite infrastructure is deployed and available.

3.2 Implementation

The plugin uses the java API of gLite WMS (Workload Management System) in order to interact with EGEE. In the current work we use the gLite WMS proxy's java API for client side interactions with EGEE. The java API contacts the service endpoint of the WMS Proxy services in order to submit a job to the EGEE. T2's plugin extension framework acts as a ready made container for the activity plugin. This container is used as a wrapper to the desired functionality. The gLite plugin provides an EGEE user with the following functionality for Job lifecycle management via gLite middleware:

Job Description and System Properties: The workflow designer may configure any T2 task as a gLite job. Typical properties include the executable, arguments, standard streams and sandbox. These properties are then passed into the gLite API. A prewritten JDL file can be directly chosen from the filesystem.

Data Transfer: The data is transferred to the Storage Elements of the EGEE and identified with a unique ID managed by the Data Management System on EGEE. The transmission protocol used is the standard gridFTP.

Job Submission: Job is submitted to the WMS proxy endpoint corresponding to the user's VO. gLite returns a unique jobID that is held by the plugin for future reference, and to poll EGEE for its completion status.

Job Status Polling: As EGEE currently provides no notification mechanism to notify clients of the state of a submitted job, clients must periodically poll the jobs for their status, until either the job is completed, or an error is generated. The polling frequency of the gLite plugin can be configured in order to avoid either excessive overhead, or unnecessary delays in retrieving the results.

Timeouts and Resubmission: The plugin includes a number of fault recovery measures, in order to reduce the probability of workflow failure (workflows execute in a distributed environment). These include: (i) a job resubmission policy (after a certain waiting time the jobs are resubmitted [5]), (ii) a Round-Robin selection policy for EGEE Workload Manager Services, and (iii) resubmitting the data transfer requests in case of failures.

Results Collection: Job completion status and output sandbox as specified in the Job Description are collected into the local file system. The output sandbox usually contains the standard output/error files and relatively small output files if any.

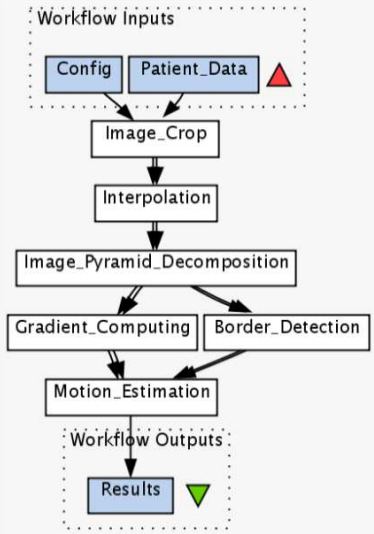

Figure 2. A cardiac workflow with application tasks, dataflow links, inputs and outputs

To conclude, the following steps summarize a step-by-step procedure of creation, execution and reuse for a T2 workflow with gLite processors:

1. Create a workflow using the Taverna workbench.
2. Add gLite processor(s) to the workflow.
3. Set/configure the properties of the gLite processor.
4. Create a grid proxy certificate for authentication.
5. Delegate the proxy to gLite middleware (builtin).
6. Add inputs/outputs and other local/gLite processors to the workflow.
7. Save, submit and reuse the workflow.

4 Use Case and Evaluation

4.1 Use Case Application

The cardiac-image processing application [1] analyzes time series of 3D heart images to analyze the myocardial motion. The sources images are sets of 2D slices encoded in the DICOM file format segregated into a hierarchy of patient-wise and time instant-wise data. Diverse compute load characteristics are found among the tasks forming the cardiac workflow. On the

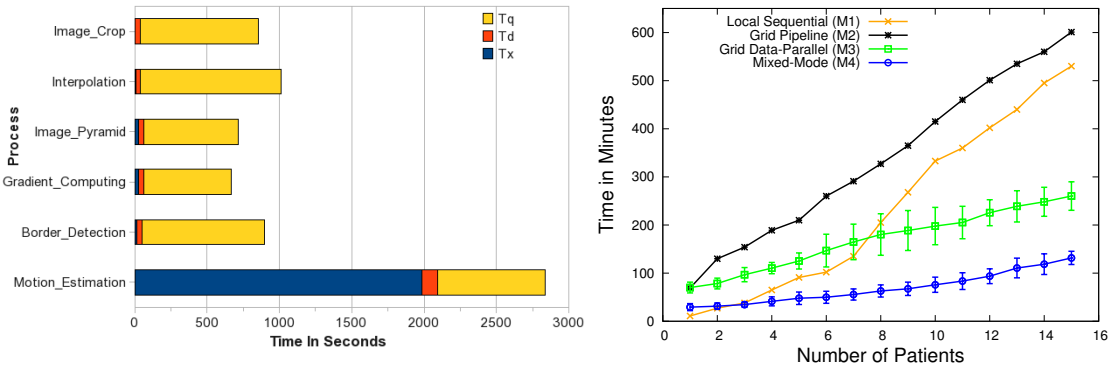

Figure 3. Left: Average Grid Profile of Cardiac Processes (Tx=Execution Time, Td=Data Transfer time, Tq=Time in Job Queue). Right: Performance Results of Cardiac Image Processing Workflow.

one extreme are the tasks that need to transfer only a few string values while on the other are the tasks requiring to transfer image data in the order of a few MBs. From the compute requirements point of view the tasks range from a minimal sequential algorithm to MPI based parallel algorithm requiring multiprocessor support and running for up to 12 hours. A pipeline showing these tasks is shown in Figure 2. *Image_Crop* crops all the DICOM files into a Region of Interest bounded by Cartesian coordinates. *Interpolation* interpolates a series of 2D images corresponding to one volume to form one 3D image. *Pyramid_Decomposition* produces images at successively lower resolution from an image. *Gradient_Computing* computes the motion gradient from points on the cardiac image. *Border_Detection* is an edge detection filter that outputs the cardiac image border based on the pixel contrast threshold values. *Motion_Estimation* estimates the quantitative motion of points on the image across the myocardial motion.

4.2 Results and Evaluation.

This section presents the scalability results for the Cardiac Image Processing workflow executed on the EGEE grid. Performance is measured in terms of workflow execution time for an increasing number of patients data. Four different execution modalities are considered as described below. The design of the 4 workflows corresponding to the 4 modalities is simplified by the fact that one workflow template can be used as a base, to be modified according to design the other ones.

Execution Modalities. Different execution mode are considered for performance assessment: (M1) pure sequential run on a local processor used as a baseline; (M2) pipelined workflow mode in pure grid execution; (M3) advanced data-parallel mode in pure grid execution; and (M4) mixed execution mode, mixing data-parallel grid processors and local processors. Performance measurements as a function of the number of patients are reported in the right of figure 3 for all four modes. The M3 and M4 curves display the average execution time over 5 runs $\pm$ 1 standard deviation to take into account grid execution time variability (heterogeneous resources and variable workload conditions).

Application Execution Profiles. The left side of figure 3 shows the average grid profile of the processes involved in the workflow. *Tx* is the execution time of process on a grid Worker Node, *Td* is the time required to transfer the data from grid Storage Elements to the worker node and back after process execution, and *Tq* represents the time spent inside the job queue after the job has been submitted and before the start of its execution. As seen from the histogram, the "Motion Estimation" process execution time largely dominates the complete workflow execution time. In addition, the grid processes turnaround time (sum of Tx, Td and Tq) shows a very significant overhead, as compared to the local execution time (Tx), especially considering the shortest processes. This overhead includes the time spent in setting up the tasks to be submitted as jobs to gLite (generating JDL and a wrapper script definition providing pre and post processing steps), communication latency and data transfer times between the workflow engine and the EGEE nodes, and batch queue time.

Performance Analysis The sequential workflow execution time (M1 curve) increases almost linearly with the number of patients, although the execution

time is dependent on each data set and not completely regular. The M2 curve shows the execution of this workflow on EGEE using the pipeline modality. The poor performance of this modality is explained by the overhead incurred, in the form of data transfer and waiting times at job queues. Further, as the Motion Estimation process is dominating the workflow, the enactor waits for that stage of the pipeline to be completed before a next execution instance starts. This problem is solved by the data-parallel modality as shown by the M3 curve. The workflow executes concurrently for all patients in this modality. The performance gain obtained is in the order of three-fold better that of the simple pipeline and local execution. When 8 patients or more are processed concurrently, the grid parallelism gain overcomes the overheads and M3 out performs M1. The performance is increased further with the "mixed-mode" execution (M4 curve). This mode is by far the best enactment strategy as it executes lightweight tasks through locally invoked execution threads avoiding costly data transfers and job queue waits. Only the heavy weight 'Motion_Estimation' task is ported to EGEE in this case. A combination of data parallel grid execution and local execution can significantly increase the overall performance of a workflow execution. Care should be taken though as to set a limited number of local processors while using the "mixed-mode" workflow. Too many local threads might saturate the local resources.

Data Management. Relatively small application data is managed by gLite middleware directly through "sandboxes". However, large application data needs to be managed separately. T2's reference handling mechanism ensures that unnecessary data transfer does not take place between enactor and remotely executing processes. We use EGEE Data Management facility that enables registration of data into an EGEE catalog and store it into its Storage Elements (SE). This way the data needs to be transferred only once across the whole workflow. At the beginning of the workflow, the data is transferred to EGEE from localhost and at the end of the workflow, it is transferred back to the localhost.

5 Related Work

Our effort complements the past research on other infrastructures with different approach. MOTEUR [2] is a data-intensive, fully asynchronous workflow engine for scufl-based workflows. MOTEUR provides a direct interface to grid middlewares including the Grid5000-OAR and EGEE-gLite. The workflows are submitted to the middleware via a generic web service application wrapper [3]. However, currently, MOTEUR lacks a sophisticated user interface. A caGrid [9] plugin is a service interface to the Globus Toolkit that exposes the caGrid services to the T2 workbench. An ARCGrid [4] plugin to T2 provides access to a middleware based portal and optionally directly to the NorduGrid via an enhanced GridFTP interface.

6 Conclusion and Perspectives

The current work successfully demonstrates advantages of Medical-Image processing applications ported to the Grid infrastructures via workflow enactment. While we have considered Medical-Imaging, this could be well applicable to other applications area using data pipeline.
It is far from trivial to efficiently bridge a workflow enactor and the Grid infrastructure. The data-flow and task representations differ greatly on the two domains posing challenges to end-users. We show that there could be different enactment strategies depending upon the nature of different tasks and that those tasks could be selectively scheduled on local or Grid resources. A simple resubmission policy has significantly increased the overall reliability of the workflow execution on EGEE.
Currently it is required by the workflow designer to judge the complexity of each involved task in a workflow and a manual judgment for its target execution environment. This is not trivial and requires special efforts for each case. Developing heuristics for selection of tasks for local or Grid execution considering their execution profiles and load capacity of local environment will ease the decision for workflow enactment strategies.
Work is ongoing for optimization of the job status polling, which can be addressed by polling together a collection of submitted jobs, in order to minimize network traffic.

Acknowledgments

This work is supported by the French ANR GWENDIA project under contract number ANR-06-MDCA-009. We are thankful to ^{my}Grid developers team for their excellent technical support and CREATIS (CNRS-INSERM) laboratory for providing application components. We thank Oleg Sukhoroslov for providing jLite java API.

References

[1] B. Delhay, P. Clarysse, and I. E. Magnin. Locally adapted spatio-temporal deformation model for dense motion estimation in periodic cardiac image sequences. In *FIMH*, pages 393–402, Salt Lake City, USA, 2007.

[2] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *International Journal of High Performance Computing and Applications (IJHPCA)*, 2008.

[3] T. Glatard, J. Montagnat, and X. Pennec. Efficient services composition for grid-enabled data-intensive applications. In *IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, pages 333–334, Paris, France, June 2006.

[4] H. N. N. Krabbenhöft, S. Möller, and D. Bayer. Integrating arc grid middleware with taverna workflows. *Bioinformatics*, March 2008.

[5] D. Lingrand, J. Montagnat, and T. Glatard. Estimating the execution context for refining submission strategies on production grids. In *Assessing Models of Networks and Distributed Computing Platforms (ASSESS, CCgrid'08)*, pages 753 – 758, Lyon, May 2008. IEEE.

[6] J. Montagnat, . Frohner, D. Jouvenot, C. Pera, P. Kunszt, B. Koblitz, N. Santos, C. Loomis, R. Texier, D. Lingrand, P. Guio, R. Brito Da Rocha, A. Sobreira de Almeida, and Z. Farkas. A Secure Grid Medical Data Manager Interfaced to the gLite Middleware. *Journal of Grid Computing (JGC)*, 6(1):45–59, Mar. 2008.

[7] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, November 2004.

[8] T. Oinn, M. Greenwood, M. Addis, and M. N. A. et al. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, August 2006.

[9] W. Tan, R. Madduri, K. Keshav, B. E. Suzek, S. Oster, and I. Foster. Orchestrating cagrid services in taverna. *The 2008 IEEE International Conference on Web Services (ICWS 2008),Beijing, China*, September 2008.