PROCEEDINGS OF SCIENCE

The BioVeL Project: Robust phylogenetic workflows running on the GRID

Giacinto DONVITO*
INFN-Bari
E-mail: giacinto.donvito@ba.infn.it

Saverio VICARIO
CNR-ITB
E-mail: saverio.vicario@ba.itb.cnr.it

Pasquale NOTARANGELO
INFN-Bari
E-mail: pasquale.notarangelo@ba.infn.it

Bachir Balech
CNR-IBBE
E-mail: balechbachir@googlemail.com

Overview Altered species distributions, the changing nature of ecosystems and increased risks of extinction all have an impact in important areas of societal concern. Biologists and environmental scientists are asked to provide decision support for managing the biodiversity component of our environment at multiple scales (genomic, organism, habitat, ecosystem, landscape) to prevent and mitigate such losses. BioVeL want to address this needs offering a series of robust and reliable web services that could be managed with the suite of tools of the myGRID project. The project will propose workflow for these services that will ensure best practice and efficiency of usage. Within the first round of services produced by the project there are phylogenetic inference workflows. These workflows will provide to end user the capabilities to execute application that could easily exploit several kind of resources, like: EGI grid infrastructure, local batch farm or dedicated servers.

EGI Community Forum 2012 / EMI Second Technical Conference,
26-30 March, 2012
Munich, Germany

*Speaker.

PoS(EGICF12-EMITC2)029

1. Introduction to BioVeL project: E-solutions for the management of biodiversity in the 21 century

Scientists are pressed to provide convincing evidence of changes to contemporary biodiversity, to identify factors causing decline in biodiversity and to predict the impact, and to suggest ways of combating biodiversity loss. Altered species distributions, the changing nature of ecosystems and increased risks of extinction all have an impact in important areas of societal concern. Biologists and environmental scientists are asked to provide decision support for managing the biodiversity component of our environment at multiple scales (genomic, organism, habitat, ecosystem, landscape) to prevent and mitigate such losses. Generating such evidence and providing decision support relies, increasingly, on large collections of data held in digital formats and the application of substantial computational capability and capacity to analyse, model and simulate in association with such data. The aim of the BioVeL(*BioVeL*) project is to provide a seamlessly connected environment that makes it easier for biodiversity scientists to carry out *in silico* analysis of relevant biodiversity data and to pursue *in silico* experimentation based on composing and executing sequences of complex digital data manipulations and modelling tasks. In BioVeL scientists and technologists will work together to meet the needs and demands for *in silico* or *e-Science* and to create a production-quality infrastructure to enable pipelining of data and analysis into efficient integrated workflows. Workflows represent a way of speeding up scientific advance when that advance is based on the manipulation of digital data (Gil et al. 2007).

2. Biological goal of the Phylogenetic Services

Phylogenetic inference is at the base of the archiving system of biodiversity, namely Systematics. All classification level of systematics but species are based exclusively on phylogenetic inference, and in recent time mainly on molecular phylogenetic inference. The term phylogenetic inference encompasses all procedures necessary to reconstruct the historical relationship among organisms or portion of their hereditary information as genes. Such reconstructions deal both with the topological shape of the relation and the quantification of the divergence among organisms. As such the phylogenetics reconstruction is an exhaustive summary of the evolution that brought the diversity of organisms or genes under study and for this reason is a basic tool to interpret (i.e. qualify and quantify) biodiversity. Phylogenetic methods are used to reconstruct the biogeographic history of species and populations, annotate genes, reconstruct the mode and time of the evolution of species- or gene-specific features. However, phylogenetic approaches are still not fully integrated in biodiversity research, because of difficulties to handle large data sets, complex data/-analysis, and old standards. Phylogenetic inference, as all parametric statistical inference, is a delicate procedure that requires to follow good practices. In fact phylogenetic inference is sensitive to model misspecification and could even produce erroneous results using the correct model (see "long branch attraction" literature). For those approaches that use phylogenetic information from previous analysis is often difficult to use the phylogenetic output of a program as input for the further analysis because phylogenetic uncertainty could not be correctly reported. Furthermore, phylogenetic inference depend from the call of homology performed on the raw data, that on molecular data is framed as the sequence alignment problem. In summary the basic procedure

of sample selection, alignment, phylogenetic inference and downstream analysis need to be completed with a step of quality control that would check stability and biological consistency of the alignment, a step of control of convergence to a unique solution of the phylogenetic inference, and a *post hoc* validation of the model or models used. Special attention should be brought the capability to deal with large data sets. Especially since the emergence of high throughput sequencing methods, access to computing power is one of the central problems in phylogenetics. In fact, being phylogenetics a NP (non polynomial time) complete problem and given that number of possible tree topology grow factorially with number of sequences only the enumeration of very large numbers of possible solution guarantee an exact solution. Nowadays different approximate (i.e. heuristic) solutions are available for large data set both based on maximum likelihood approach (i.e. RaxML, Garlie), Bayesian Markov Chain Monte Carlo (i.e. MrBayes, BEAST) or Approximated Bayesian Computation (i.e. ABCtoolbox , SPLATCHES). Still all these approaches require a large number of computation of the likelihood of different phylogenetic hypotheses (in the order of 10^5) or coalescent histories (in the order of 10^7) for a analysis in the order of few hundreds of sequences. A part the complexities residing in the tree shape itself phylogenetic inference need to deal with the complexity of the model of the character that evolve on the tree. Deep-medium divergence for a correct representation of the data need complex substitution model that for example in phylogenomics data sets need to partitioned model with potentially different substitution matrices (i.e. single base, doublet, amino acid, codon), although with the simplifying assumption that demographic fluctuation would be erased by coalescence events. On the contrary Phylogeography model would concentrate its complexities on the interaction between demography and tree shape (topology and branch lengths).

Moreover, a related problem is the complexity of initial data mining procedures. To reconstruct each phylogeny of gene family, a complex substitution model need to implemented. Today we get millions of sequences as raw data, but we have few tools to overview and process (i.e. data mine) high-throughput data, as Next Generation Sequencing data. In light of these challenges the phylogenetic service set of BioVeL would like to produce 4 type of workflows: a molecular phylogenetic inference set, phylogenetic diversity set, phylogenetic molecular evolution set, phylogenetic character mapping set. The work presented here is the first instance of the a workflow of the first type that deal with producing a phylogenetic inference of set of unaligned protein coding nucleotide sequences. First the workflow find the most likely protein domain common to all nucleotide sequence trying all coding frame and all user selected genetic codes using the implementation of Hidden Markov model of HMMer 3.0 and the PFAM db. Then it perform the alignment based on that profile, filtering sites with low fit on the protein profile. The file is then sent to perform a phylogenetic inference with the program MrBayes. To do so the file is formatted in Nexus format and a user specified model of evolution and details of the markovian integration is attached to the file. The MrBayes services set up automatically the best MPI set up to the user requests on the markovian integration and is sent to an optimal, in term of efficiency of use, number of CPUs.

3. Technical and users requirements

In developing this framework we started with the analysis of the technical and users requirement in order to find the best solutions for our use-case.

3.1 Authentication and security

We decided to split the function of identification of the user for access to specialized computing resources within the grid and the security of the system. This to allow for user without special access to resource to go on a web page and to submit queries or calculation with a simple web form, or web service without have to register or request for a certificate. For the purpose of this paper we will describe only how we maintained the level of security required in the system without user identification. The problem of differential access to computing resource is postponed and left to the BioVeL project as a whole or more precisely to the agreement to be settled between the Lifewatch ESFRI (in which BioVeL is included) and EGI. We didn't implemented any security routine within the workflow that access the webservices because one of the point of interest of the project is to allow the user to modify the workflow if needed. So the security was enforced only in the Web Services that compose the workflows.

3.2 Client requirements

One other important requirement is about the client used by the end users. The BioVeL project as decided that the unique supported client, for managing the complex workflow requested by the scientific analysis is Taverna workflow manager. This means that we had to find the best implementation that could work with Taverna. We soon understood that this kind of technology could be interesting also for other users that could or would not exploit such a powerful tool like Taverna, but they only have some Command Line Application like curl or a simple web browser. In order to support both the use cases we have found that the best solution is to provide a REST interface with get method.

4. Overall description of Job Submission Tool

In order to satisfy the requirement of the project to submit from a Web Services jobs to the grid and local computing infrastructure, we expanded the functionality of Job Submission Tool (JST; Tulipano et al. 2011). JST has been developed years ago to allow the submission of large numbers of jobs and keep track of all of them in an unattended way but was not able yet to accept job submission through a Web Service.

4.1 Anatomy of Job Submission Tool

As shown in the Figure 1, the system, after the expansion, is made of four macro-elements:

- Managment DataBase (MBD): MySQL Database of Job storage and configuration parameters
- FrontEnd: Rest Services accessible from the outside
- BackEnd: Application daemon that runs in the background for the job submission and execution both on local machine and on grid environment
- Exchange Files Server: A WebDav server that allow the user and the Working node to exchange data files among them

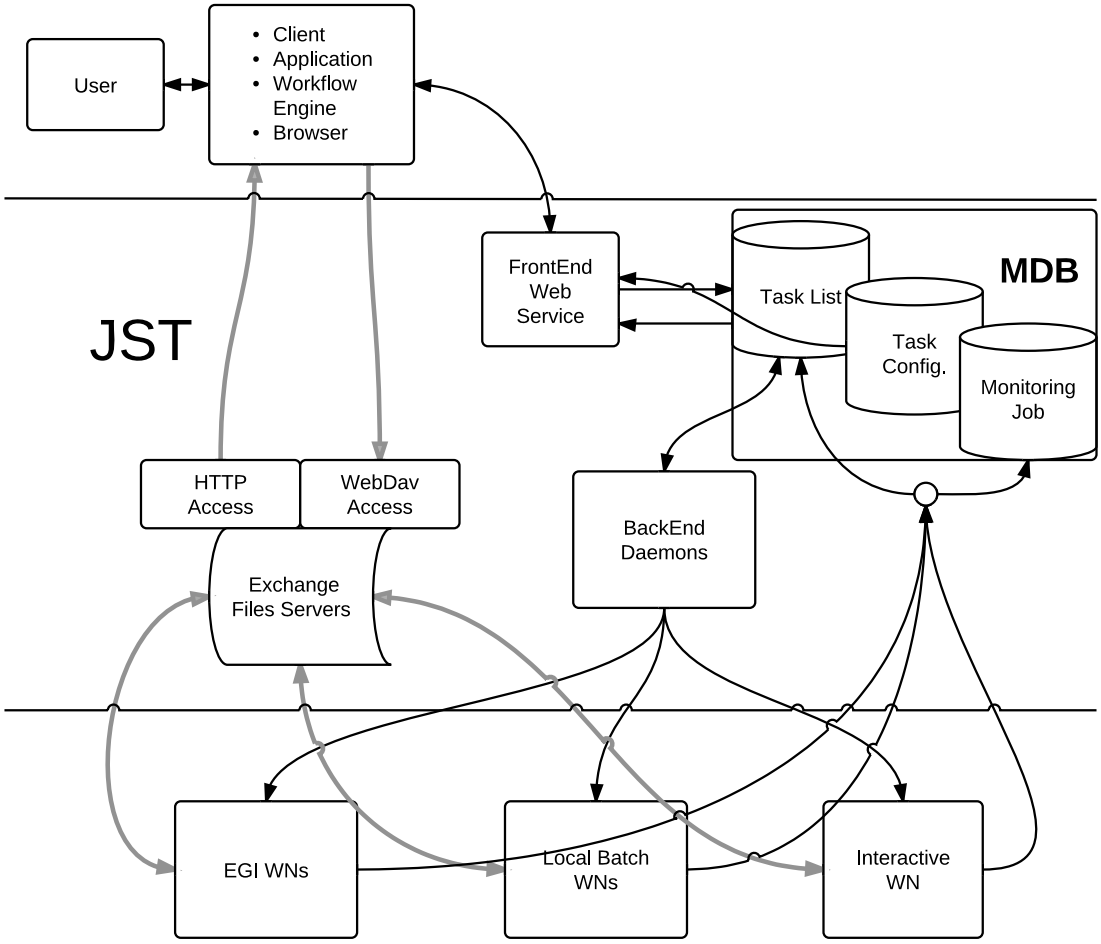

Figure 1: Interaction between elements. The graph include the four JST element together with the user, client and working nodes (WN) elements. The arrows indicate the flow of information. Gray, thick arrows indicate the flow of large files, while thin and black arrows indicate flow of small text file of various format.

The components called FrontEnd and BackEnd have been written in Java language using the Framework Java EE 6.0; in particular the FrontEnd also uses the Framework Jarsey JAX-RS and the Web Server Apache Tomcat.

The user can interact with the system in several ways, can call the FrontEnd component by Command Line Interface clients (eg. curl or wget), by all the Web browser or by Workflow manager (eg. Taverna). Each of the step could be executed by means of the same tool or using different tools, indeed the FrontEnd provides a client independent layer for the grid job scheduling system.

4.2 Job submission Tool at work

At the user submission phase all the jobs are identical: as a matter of fact, when a job is submitted it does not know which task(s) it has to execute. FrontEnd access the table ConfigTask (1) of the MDB to retrieve all the information and the pieces of software required to perform the job and define how many and of what type of tasks are needed. Using this information FrontEnd fill

ConfigTask	
NAME	CHAR
LINK	CHAR
DATE_NOW	DATE
CPUs	CHAR
COMMENT	LONG_TEXT
STATUS	CHAR

Table 1: ConfigTask schema

a record for each task in the Tasklist (2) in MDB. For the proper monitoring of the task assignment and job termination, several parameters are associated to each task, as you can see in the reported schema of the tables. Few of the most important field are detailed described here:

- The status of the task: can be set "Free" (not assigned), "Running" (assigned) and "Done" (job terminated). If the status is "Free" the task can be assigned to a job. Also if the state "Running" was there for more then a fixed time interval, meaning that probably the job has failed, the task can be reassigned to a new job. If the state is "Done" or is "Running" for less then the fixed time interval, the task is ignored during the tasks assignment process.
- The dependencies of each task. It is possible to flag the tasks that need the execution of a different task before its execution (dependency): only tasks with no dependencies or with all the tasks, from which they depend on, in the "Done" status, can be executed.
- Priority. It is possible to assign an arbitrary priority to each task. Priority is used to select the task that has to be executed first.
- Job provenance. It is possible to know which job has actually performed which task.
- The task description: it can be a string or a link to a specific script to execute, in this way it is possible to better control what is being executed on the WN. As matter of fact, it is possible to change the execution (i.e. if a bug is discovered, or there is a need for a new optimization) also after the submission of the jobs.
- Number of failures. When a task fails, the system logs the event in order to avoid the resubmission of always failing tasks. It is possible to set a maximum number of resubmissions.
- Date and time of execution.
- The username of the user that is submitting the task together with his mail address
- The information on the infrastructure where the task should be executed (grid, local farm, interactive server)

The BackEnd daemon is used to submit the task at a given rate, which could be tuned. As matter of fact several daemons were used in parallel. Each one could use more than one gLite WMS in order to avoid that a failure of a single WMS could stop the submission procedure. The

TaskList	
ID	INTEGER
NAME	CHAR
PRIOR	INTEGER
STATUS	CHAR
PROVENANCE	CHAR
LINK	CHAR
DATE_NOW	DATE
FAILURE	CHAR
arguments	LONG_TEXT
COMMENT	LONG_TEXT
JOB_ID	CHAR
MAIL	CHAR
OUTPUT	CHAR
OUTPUT_TYPE	CHAR
FLAG	CHAR

Table 2: TaskList schema

job_mon	
TIME_LOCAL	timestamp
JOB_ID	CHAR
JOB_NAME	CHAR
NOME_VAR	CHAR
TIPO_VAR	CHAR
DESCRIPTION	LONG_TEXT
HOST	CHAR
FLAG	CHAR

Table 3: Job Monitoring schema. The columns are name and type of the field in the table. The field are: date and time on the WN that sends the monitoring information; job id and name; name and type of the monitored variable; description of the variable; host that is sending the information; Id of the task

daemons automatically stop the jobs submission when no more unassigned tasks are found in the TaskList table. The jobs submitted to the grid contain a job wrapper. The job wrapper and the script that contains the executable can sends to the MDB server on the job_mon table (Table 3) every kind of information that can be used to control which jobs are running and, which task they are performing and which operation they are performing on the WN. This table can be separated from the one hosting the task list, the MDB, if the scalability of the DB itself become an issue. A record in this table represent a single variable monitored in a given task and job, so several records could match a given task.

This general schema leaves the user with the possibility to monitor any job operation by adding new variables as needed without changing the DB schema. If required this feature provides the information on how many jobs are running at a given time. Indeed, the job-wrapper could be

configured to send regularly monitoring information at fixed time intervals. Information on the execution of each task is logged in the central MDB according the parameters mentioned above. Only if all steps are correctly executed, the status of that particular task is updated to "Done". In this way the MDB provides a complete monitoring of the task assignment and job execution and no manual intervention is required to follow each step and to manage the eventual resubmission of failed tasks. Usually we see two kind of failed jobs: the first kind of failures includes cases of jobs that were killed by the queue manager, while the second one is related to internal problem of the job (some operation failed). In the first case the wrapper can not update the MDB, while in the second case the wrapper script updates the MDB increasing the number in the "FAILURE counter". In order to deal with the first kind of failure, tasks which are found in a "RUNNING" state by more then a fixed amount of time are considered failed and automatically resubmitted without increasing the failure counter. To optimize the input and output operations and to avoid bottlenecks and failures if/when needed the files available on the File Exchange server could be transferred on the grid Storage Element. In this case, two procedures were set up to randomly choose the storage element (SE) source of the job input file as well as the SE where to store the job output. In this way if one SE fails temporarily it is possible to continue running the task and store the output.

4.3 Software Architecture of the JST

To allow robustness and seamless interaction across module that handle new and old functionality of JST, the application was rewritten using the MVC design pattern (Model View Control) and three distinct layers were defined:

- View
- Business logic
- Access to data

In order to implement those layer we developed a series of libraries, opportunely referenced and integrated with each other. These libraries are:

- Service library, in FrontEnd package, contains the methods available to the service
- XmlEntityLayer is devoted to serialize/deserialize the XML response format of the service (Map of the responses in XML)
- EntityLayer contains the schema of the database
- BusinessLayer contains the business logic for the management of the MDB and methods of service
- DataLayer provides classes and methods for accessing data in the database
- Core contains various utilities and functionality
- MultiThread contains management and scheduling methods of the thread for the execution of the Job

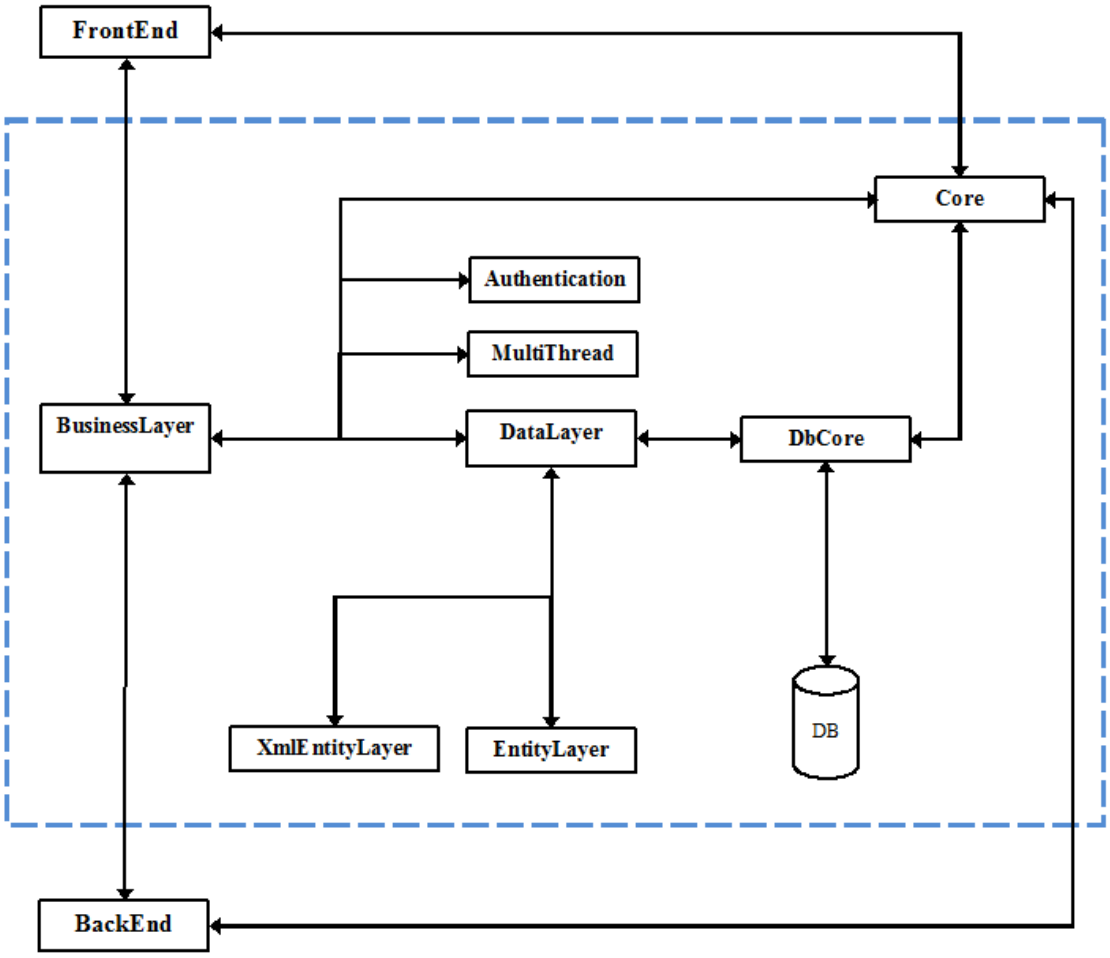

Figure 2: Libraries Schema

Figure 2 is shown how each library interact which each other. In particular FrontEnd and BackEnd interact directly with the libraries BusinessLayer and Core. BusinessLayer implements the logic of system and the management of the database, so the BusinessLayer is the component that calls opportunely the other libraries.

4.4 BackEnd implementation features

The application called BackEnd, written in Java, it is a daemon, and could be constantly running on many nodes contemporarly in order to provide better scalability. It is implemented exploiting a MultiThread strategy and it is interfaced to the MDB to retrieve the task to be submitted to the Grid, to a Local Farm or to be executed on dedicated servers. Each of the task is flagged to be executed over a grid environment, or on a local batch farm, or to be executed on the server itself. The BackEnd deamon could be configured by a site admin and could cover all the three functionalities. Indeed it could execute on the same server a give amount of jobs, or could submit the request for executing job over the grid or local computing infrastructures.

4.5 FileExchange Server

In order to fulfil the community requirement in terms of data management we have chosen a solution that is able to provide an easy and user friendly interface for transfer file of several GB size or huge number of small files in a complex directory structure. All this requirements are satisfied by mean of an Apache based WebDAV server. Indeed, this solution has several specific characteristics that could be useful:

- Flexible authentication
- Could be mounted as a file-system
  - Allowing drag&drop operation
  - Allowing direct opening of the files
- Supported in the most widely used Operating System
  - Windows
  - Mac Os
  - Linux
  - Android
  - iOS
- Could support lock of files
- Could provide Name Space operation
- Could support user driven metadata information management (PUT and GET)

The files could also be available via http/https link, both to the end users and to the application. This means that there is no need to transfer the temporary output from the computing infrastructure to the client, but an application, running under a web services, could exploit the output of another application directly from the server instead of requiring staging the files to/from the desktop of the users. This will greatly improve the overall performance in running multi-step workflow.

Security issues on Data Management

At the moment the project is trying to promote its services to external users, so the framework should provide functionalities to allow seamless interaction with the storage services within the Taverna Workflow, both in terms of input upload and output download. Furthermore, given that all users share the same account without password we add the requirement of avoiding the overwriting the input files uploaded from the users, and avoiding the possibility for a users to read output files of others users and avoiding overflowingof the storage server by a single user. The security in the exchange of files is guaranteed using two different systems in output and input. In input the user are given access to File Exchange Server, a WebDav server without user and password identification. The account allow only uploading but no reading. Further the account is sized limited and periodically erased. To avoid that two users will submit the files with the same name during a same period of usage, in the account the overwriting is disabled. In output both the users

or further job that need to access results can use an address to access an HTTP read-only folder. To ensure security the name of the last directory in the address is randomly generated. The web services delivered the address only if the user interrogate the Web Services with the correct FLAG value given during the job submission transaction.

4.6 FrontEnd implementation features

The application called FrontEnd was written in Java using the components Jersey; in particular the framework JAX-RS and Java 6 SDK. The FrontEnd is a RESTful web services: those service are widely used in the workflow managers environment, but can also be easily exploited with simpler clients, like unix command line tools and standard web browsers.

Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and the possibility to easily modify the application or reuse part of its code. This is important to provide usable and robust services to the end users. In the REST architectural style, data and functionality are considered resources, and these resources are accessed using Uniform Resource Identifiers (URIs), typically links on the web.

The resources could be accessed or modified by using a set of simple, well-defined operations. The REST architectural style constrains a client-server architecture, and it is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources using a standardized interface and protocol. These principles encourage RESTful applications to be simple and lightweight in order to provide high performance. One of the advantages of the use of REST paradigm is also the independence from the clients ensuring a high portability and scalability.

Security implementation

The user interact with the computing infrastructure only by means of the FrontEnd, sending short string within predefined REST command. The security within the REST command is maintain by the fact that the string are used to compose the argument for predefined application, avoiding the risk of shell injection. Indeed, if required, the same methods could be implemented exploiting HTTPS protocols in order to guarantee the privacy of the data provided to REST service.

Supported Methods

InsertJobs: this is the method that allow the user to create a multiple jobs, for the same application, into the database, each job is identified by a different argument. Arguments are split using ";"

```
http://localhost:8080/INFN.Grid.FrontEnd/services/QueryJob/
    InsertJobs?NAME={nameValue}&arguments={http://webtest.ba.infn.
    it/vicario/FinalFusariumDB_2.nex ArgOne; http://webtest.ba.
    infn.it/vicario/FinalFusariumDB_1.nex ArgTwo;}
```

This method returns the xml type: XmlInsertJobs with information on to the tasks added in the table TaskList.

SelectJobs: this is the method that allow the user to get the status of a list of task of a given job into the database, identified by the "FLAG" value:

```
http://localhost:8080/INFN.Grid.FrontEnd/services/QueryJob/
    SelectJobs?FLAG={FlagValue}
```

This method returns the xml type: XmlSelectJobs with information relating to the jobs searched in the table TaskList.

Here is an example of the structure of the XML that the FrontEnd service report to the user:

```
XmlSelectJobs:
        <Jobs>
                <Job>
                        <Arguments>Arg4</Arguments>
                        <Comment>local</Comment>
                        <CPUs>1</CPUs>
                        <Flag>001</Flag>
                        <Id>4</Id>
                        <LastCheck>2012-02-14 10:54:58.0</
                            LastCheck>
                        <Name>blast4</Name>
                        <Output/>
                        <Provenance/>
                        <Status>done</Status>
                </Job>
        </Jobs>

        XmlInsertJobs:
        <Job>
                <Name>NOME</Name>
                <Flag>001</Flag>
                <JobsID>
                        <JobId>12</JobId>
                        <JobId>13</JobId>
                </JobsID>
        </Job>
```

5. Interaction between REST service and Taverna Workflow Manager

The interaction between Taverna (Hull et al. 2006) workflow manager and the FrontEnd of the submission service is possible using the Taverna plugin that is able to exploit the REST web service. In order to use it is only needed to provide the URL where the service is located, the parameters that are needed in order to submit and check the status of the jobs. The parameters needed are:

- The name of the application
- The arguments

The service as already described in the previous section, could be used to run each specific task of a very complex workflow. Indeed, the Taverna could manage whithin the same workflow, services provided by this framework together with service already available in other infrastructure or *ad-hoc* developed service within Taverna itself.

This provide an extreme flexibility in building powerful workflows, as this give the possibility to the user to run over a grid infrastructure only the CPU intensive part of the work while the lightweight part could relay on any kind of infrastructures.

The main problem in interaction between this framework and the Taverna workflow manager, is that while the latter is devoted to interact with synchronous services, the framework that we build is basically asynchronous. In order to solve this issue it is needed to build a loop into Taverna workflow manager that allow the end user to wait for the execution of the task and to check the status of the submitted tasks until they get done.

An example of this workflow is shown in the Figure 3. As soon as we include more application on our computing infrastructure we will go on publishing those example workflow on standard workflow catalogues like MyExperiment (De Roure, Goble, and Stevens 2009), while the included webservices will be published in BioCatalogue (Bhagat et al. 2010).

6. Test and results

We already test this solution in order to prove the scalability, the reliability and the functionalities that we described before. We tested the FrontEnd with huge amount of insert, in order to prove that it was reliable and fast enough to serve the request from the users. The test showed that also after 1 million of inserts the FrontEnd do not suffer of memory leak or similar problems. We also tested the behaviour with multiple and concurrent access to the same server: using up to 100 concurrent clients we do not registered any kind of problem or error, and the server is also not overloaded from the resulting activities. From the point of view of the back-end and database server, we have experience in running with more than 1 million of task into a single table, without experiencing any issues. We have used this system for long running challenges on the European Grid Infrastructure (EGI) lasting for more than one month of running.

7. Conclusion and future works

The aim of this work is to provide a seamless interface to powerful computing resources, such as Grid Distributed infrastructures based on EMI middleware, or big batch farms to end-users that are used to carry-on their analysis only using simple and high-level tools like workflows managers. The key idea is to provide a stateless web services interfaces based on REST technologies in order to make transparent to the end users all the complexity related to such a job submission. In this way the researcher could exploit the grid infrastructures as he/she do with any available web services. In the next month we will work on notification (i.e email) of completion of task. This is particularly important for phylogenetic inference in which a given task of a job could last several days. It is is useful then to avoid that the user selected client/workflow engine continue to query the REST for such long amount of time. So the notification could be used to hint the user or the workflow engine to query the REST service after a long period of waiting. We plan to add a method in the FrontEnd

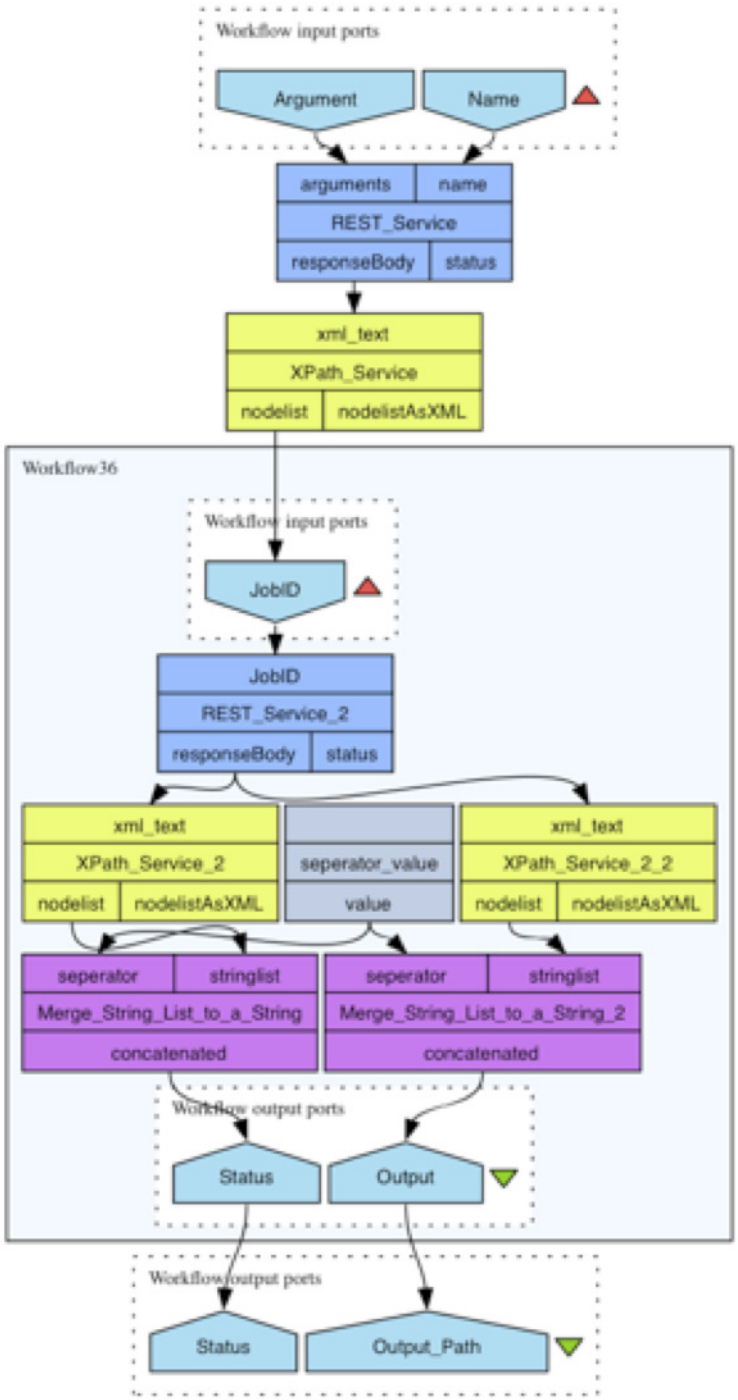

Figure 3: An example of a Taverna workflow that could exploit the Grid Job Submission Framework described

that will allow to access the monitoring job table using both Taverna Workflow manager and more simple clients. This will allow users to access more detailed statistics of their job, and diagnose cause of failure or delay of their submission.

8. Acknowledgements

This work is funded by BioVel FP7 project (grant 283359)

References

Bhagat, Jiten et al. (July 2010). "BioCatalogue: a universal catalogue of web services for the life sciences." In: *Nucleic acids research* 38.Web Server issue, W689–94. ISSN: 1362-4962. URL: http://nar.oxfordjournals.org/cgi/content/abstract/38/suppl_2/W689.

De Roure, D., C. Goble, and R. Stevens (2009). "The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows." In: *Future Generation Computer Systems* 25, pp. 561–567. URL: doi:10.1016/j.future.2008.06.010.

Gil, Yolanda et al. (Dec. 2007). "Examining the Challenges of Scientific Workflows". In: *IEEE Computer* 40.12, pp. 24–32. ISSN: 0018-9162. DOI: 10.1109/MC.2007.421. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4404805.

Hull, Duncan et al. (2006). "Taverna: a tool for building and running workflows of services." In: *Nucleic Acids Research* 34.Web Server issue, pp. 729–732. ISSN: 1362-4962. URL: http://view.ncbi.nlm.nih.gov/pubmed/16845108.

Team, BioVeL. *BioVeL*. URL: www.biovel.eu.

Tulipano, Angelica et al. (Dec. 2011). "GRID distribution supports clustering validation of large mixed microarray data sets". en. In: *EMBnet.journal* 17.1, pp. 18–25. ISSN: 1023-4144. URL: http://journal.embnet.org/index.php/embnetjournal/article/view/205/481.