Computing and Informatics, Vol. 31, 2012, 173–187

WEB INTERFACE FOR GENERIC GRID JOBS, WEB4GRID

Antònia Tugores, Pere Colet

IFISC, Instituto de Física Interdisciplinar y Sistemas Complejos (CSIC-UIB)
Campus Universitat de les Illes Balears
E-07122, Palma de Mallorca, Spain
e-mail: {antonia, pere}@ifisc.uib-csic.es

Abstract. For a long time grid has been used practically only by large projects which can afford to have fine-tuned sophisticated interfaces for the researchers involved on these projects. Grid has the potential to be a key instrument for a wide variety of scientific topics which require to perform many calculations, for example to explore complex system dynamics as function of parameters. The reduced size of the research groups and the diversity of problems makes unsuitable to develop specific interfaces in most of the instances while the cumbersome grid shell commands are a barrier that requires a significant amount of determination to overcome. Web4Grid interface is intended to be a user-friendly web where grid users can submit their jobs and recover the results on an automatic way. Besides the interface provides the capability to monitor the existing jobs and to check the (local) grid status (load, number of free cores available, ...). Web4Grid interface does not require specific grid usage training nor any knowledge about the middleware behind it.

Keywords: Grid, non classic user communities, portal, web-interface, gLite, EGI

Mathematics Subject Classification 2000: 68M14: Distributed Systems

1 INTRODUCTION

Enabling Grid for E-sciencE (EGEE) [1] was a series of projects that began in 2004 to construct a multi-science computing grid infrastructure for the European Research Area, allowing researchers to share computing resources. The aims of

this projects were to build a secure, reliable and robust grid infrastructure, to develop a middleware solution specifically intended to be used by many different scientific disciplines and to attract a wide range of users. They produced the gLite [2, 3] middleware, engineered using components from some sources like the Globus Toolkit [4]. gLite services have been adopted by a large number of computing centers around the world. Since 2010, the pan-European Grid Infrastructure (EGI) [5] in collaboration with the National Grid Initiatives (NGIs) guarantees the long term availability of the generic e-infrastructure created by the EGEE project.

User communities that use gLite are grouped into Virtual Organizations, and before accessing the grid, users must have a X509 certificate and be a member of a Virtual Organization. To access the grid users have first to log into a User Interface (UI). The UI is a computer with a X509 certification which runs specific gLite components, and in which the user certificate is installed. Once it has log in the user can have access to the grid functionalities.

To run a job, first the user has to create a proxy certificate consisting of a new certificate signed by the end user and a new private-public key pair. Then a file specifying the job characteristics such as the executable program, the parameters, input and output files, etc. has to be created. This file is written using the Job Description Language (JDL). Once the JDL descriptor file has been created, the user can submit the job to the available Workload Management System (step 3 in Figure 1) by using specific gLite commands at the User Interface. The set of available WMS is determined by the Virtual Organization. Once the job has been accepted by the WMS, the WMS assigns it to the most suitable Computing Element (step 4 in Figure 1). A Computer Element is a set of computing resources localized at a site like a cluster or a computing farm. Finally the job is executed in an idle computing resource or Worker Node (step 5 in Figure 1) and users can query the status from the UI to the WMS. The certifications allow for this process to take place without the need for the user to have an account in any computer beside the UI.

gLite has two different ways to handle the input and output files. Small input files (whose size is smaller than 100MB) can be just referenced when submitting the job. They are uploaded to the WMS and made available to the Worker Nodes. Larger files require a more cumbersome procedure consisting in uploading the files to a Storage Element through a LCG File Catalog (LFC) which acts as an interface providing human readable identifiers (step 2 in Figure 1).

Similarly, when the execution is completed, small outputs are sent directly to the WMS while large files have to be updated to the Storage Element through the LFC (step 7 in Figure 1). Finally the user, depending on the outputs size, has to download the outputs from the WMS (step 10 in Figure 1) or from the SE (step 12 in Figure 1) to the UI using specific commands that require the job identifier or the name of the output file.

The need to learn JDL language, the artificial separation between data storage and job execution, and the cumbersome and sometimes hard to memorize gLite

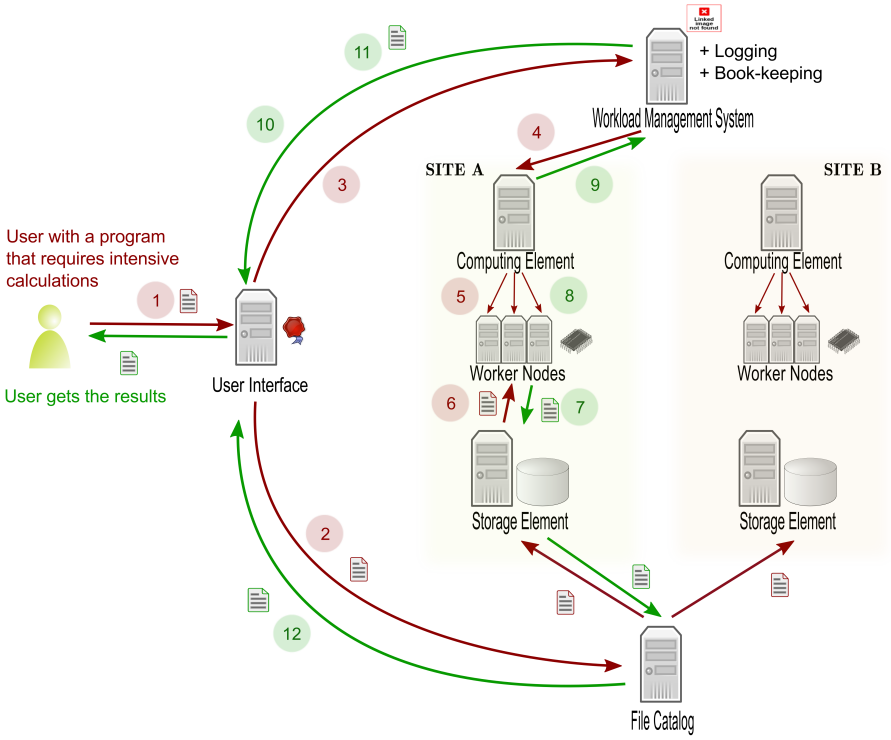

Fig. 1. gLite job workflow.

1. The user logs into the User Interface.
2. The user uploads large input files to an SE.
3. The user submits the job to the WMS.
4. The WMS sends the job to the most suitable CE.
5. The job is redirected to an idle Worker Node.
6. The WN executes the job (if needed, the user application downloads the inputs from the SE).
7. If needed, the user application uploads the outputs to the SE.
8. The WN notifies its CE that the job has finished.
9. The CE notifies the end of the job to the WMS.
10. The user queries the WMS through the UI and notices that the job status is DONE.
11. The user retrieves the outputs from the WMS through the UI.
12. The user retrieves the outputs from the SE through the UI.

commands are some of the nuisances scientists find when using gLite user interface. Using portal technology in sciencific applications such as described in [6] and [7] would solve this issue and users would have a web point of access to the available computable resources.

The objective of this paper is to present a user friendly environment to use grid we have developed. A Finite States Machine manages the job work flow and hides its complexity. A set of scripts, that implement the Finite State Machine, allow users to run their programs, get the results without any additional interaction with the grid as well as monitor the jobs. On the top of that, a web application has been developed to avoid the command line and get a graphical easy-to-use interface. Both the web application and the high level command line interface are executed in the User Interface where the user has its files as well as the certificates. The paper is organized as follows: Section 2 summarizes existing graphical grid interfaces. Section 3 describes the main script. Auxiliary scripts are described in Section 4. Section 5 is devoted to the web application. Section 6 analyses the usage and finally, concluding remarks are given in Section 7.

2 CURRENT INTERFACES

Different user interfaces have been developed to allow easy access to grid. We sumarize here the characteristics of most important ones.

Within the *e-NMR* project, [8, 9], some of the nuclear magnetic resonance applications have been adapted to the grid. Besides, some user friendly web portals, specific for each migrated application, have been also created (Figure 2a). To use these portals, users with a personal X509 certificate loaded on their web browser and registered with the enmr.eu Virtual Organization are enabled to register with username/password to the application portal and execute jobs. The most important issue of this kind of portals is the bijection between portals and applications. That does not allow researchers to use their own applications with the grid without technical help.

A more flexible portal is *Ganga* [10, 11]. It was first developed to meet the needs of the ATLAS and LHCb for a grid user interface. As a result of its modularity, it has been extended and customised to meet the needs of different user communities like Geant4 regression tests and image classification for web-based searches. The number of commands to submit jobs and retrieve results has been reduced, but the relation between portal and application is still an issue. All jobs must specify the software to be run (application), the processing system (backend) to be used and the input and output datasets. Optionally, a job may also define functions (splitters and mergers) for dividing a job into subjobs that can be processed in parallel, and for combining the resultant outputs. Although there is a graphical interface related to Ganga, the command line interface is more used.

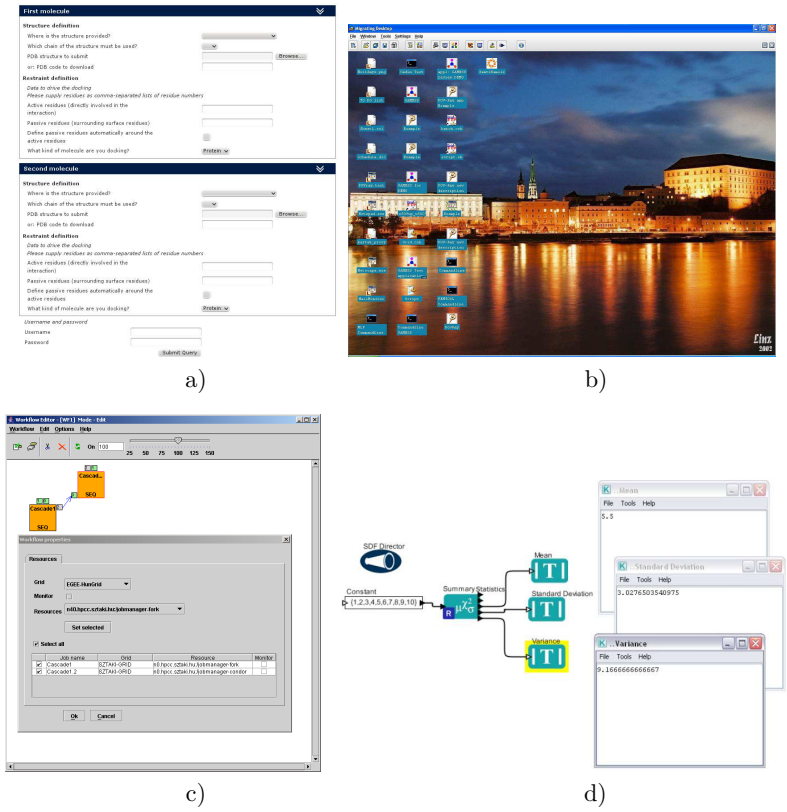

Fig. 2. a) Easy interface to the HADDOCK docking program. b) Default main Migrating Desktop window. c) P-GRADE job editor. d) Kepler simple workflow definition.

The most complete graphical user interface is *Migrating Desktop*, [12, 13]. It is similar to a window-based operating system (Figure 2 b)) that hides the complexity of grid middleware and makes access to grid resources transparent. Migrating Desktop supports batch and interactive jobs handling sequential and parallel applications, but the interface is cumbersome and requires training.

Finally, the *Parallel Grid Run-time and Application Development Environment Portal* (P-GRADE Portal), [14, 15] (Figure 2 c) and *Kepler* [16, 17] (Figure 2 d) are graphical environment that cover every stage of grid application. They are oriented to directed acyclic graphs (DAG) where each node has a computing resource and a job to be launched on that resource. This interface allows users to use jobs outputs as inputs for other jobs without human interaction. This sophisticated interface requires prior training on DAGs definition. Besides it is not suitable to simple job execution.

3 USER FRIENDLY COMMAND LINE INTERFACE. MAIN COMMAND.

Unlike usual command line interfaces, a user friendly one should not have a large number of commands nor too many parameters, and simplicity is a must. This starting point has led to a single command call invoked at the beginning to manage the whole job work flow even when the Storage Element is used; this aim is achieved with an straightforward finite state machine (FSM), a database to safely keep all the data up-to-date, a daemon in charge of detecting ending of jobs, an unified home directory between the User Interface and the users' desktops, and the well known Python flexibility.

The main script, *runGrid*, can be easily invoked by the user with just the name of the executable and the input files. runGrid performs the tasks at the level of the User Interface in a transparent way, not requiring the installation of software in any other grid computer. We assume that:

1. a researcher typically uses the same Virtual Organization;
2. jobs to be run are not parametric nor direct acyclic graphs (DAGs);
3. all the files generated by the job are expected to be returned to the user;
4. it is not known how much time the job will take.

The command runGrid is typically invoked in the form

```
runGrid -a application -p ‘‘param1 param2’’ -i *.dat,dir/inputFile
```

where `application` stands for the name of the executable file, param for the parameters to be passed to the executable (optional) and finally the name of the input data files is given after `-i` (wildcards are accepted).

We make use of long term proxies to allow for easy creation and update of mandatory short term proxies in the UI before they expire, so that long programs can be executed without user intervention.

The FSM associated to runGrid comprises several states. The first stage, INIT, is triggered by the call to runGrid. In this state a new entry is added to the database in charge of the jobs, the command line parameters are checked and a long term proxy is created. Then, the FSM moves from the INIT state to the UPLOADING state where a compressed file containing all the input files is created and uploaded to a suitable Storage Element.

Once the upload has finished the FSM enters the READY state in which some auxiliary files are created:

1. a wrapper to manage the application inputs and outputs within the Worker Node (WN); and
2. the Job Description Language (JDL) file in which the wrapper plays the role of the executable and in which the standard output and error are redirected to specific files.

At this point, the job is ready to be submitted to the grid. Some checks like the availability of Computing Elements for the Virtual Organization are done to ensure everything is alright, and finally the job is submitted. When the Computing Element job status is Running (the job has been sent to an idle Worker Node), the state changes from SUBMITTED to RUNNING and remains in this state until the job execution has finished.

In the Worker Node the application wrapper script is executed. It initially downloads the inputs from the Storage Element and after uncompressing them it launches the user application. When the application finishes, it compresses the outputs, and uploads them to the Storage Element. Finally, the exit code returned to the WMS is the same the user application returned.

When the daemon notices the job has finished the FSM moves to EXECUTED and the outputs are ready to be downloaded. In the DOWNLOADING state, both the WMS outputs and the file stored in the Storage Element are downloaded and uncompressed to a specific directory in the same folder the job has been submitted from. Then, the auxiliary files are cleaned (CLEANING), and finally the FSM moves to the final DONE state.

So far we have described the regular work flow but other states like ERROR or CANCELING exist. When a job has not finished with the correct exit code or there has been any non predictable issue the status changes to ERROR and the work flow goes on but keeps auxiliary files and increases log details.

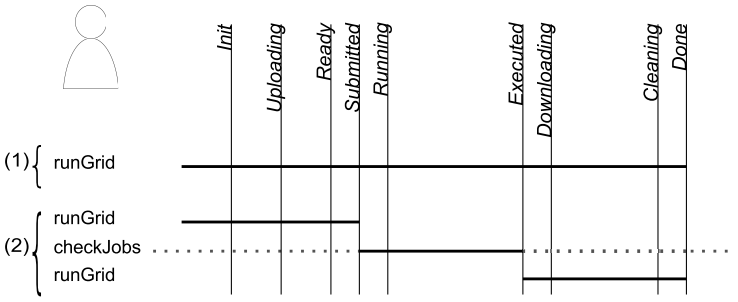

Fig. 3. The first version of runGrid (1) involved just one script to control job management that updated the job status to the local database and while the job was running it periodically checked the status via the WMS to detect the job end. The improved version (2) involved a daemon that periodically checks all the jobs of each user. In this new implementation the main script stops after submitting the job to the grid, and the daemon awakes it when the job has finished, allowing the main script to download the results and clean the data.

The workflow of the main script first version can be seen in Figure 3 (1). An instance of the script kept running until the results had not been downloaded to the User Interface (UI). In the states SUBMITTED and RUNNING the amount of connections to the WMS was too high and we had to restrict the number of parallel connections in order to avoid saturating both the WMS and the UI.

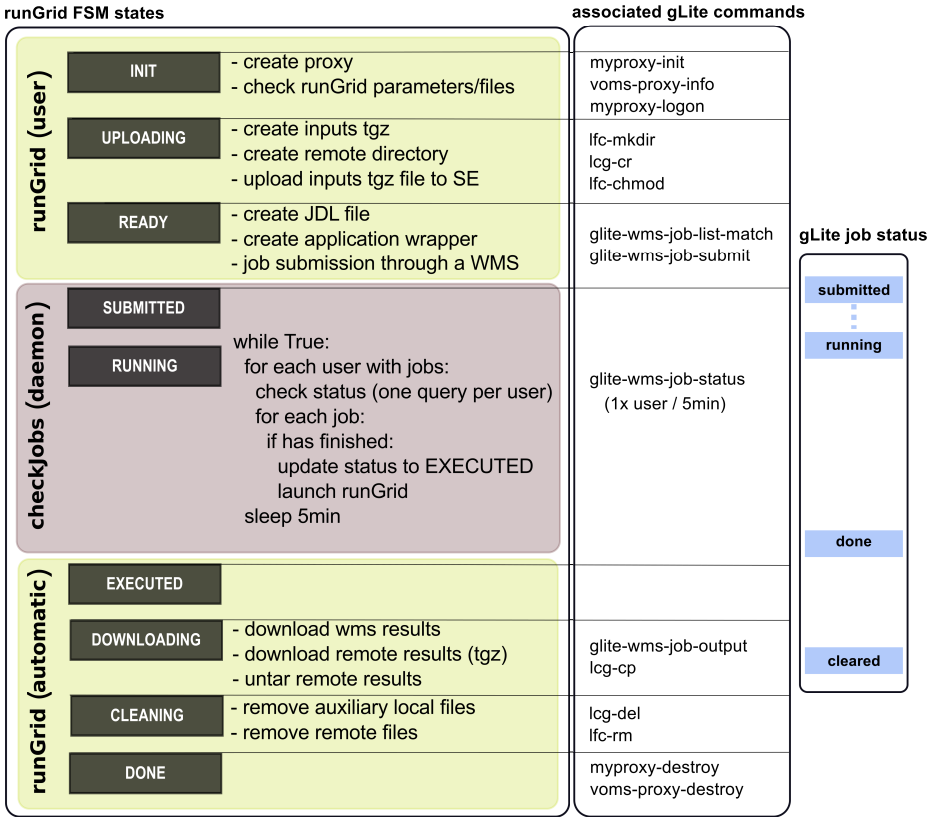

Fig. 4. States of the runGrid command and checkJobs daemon as described in the text. We also indicate the gLite commands associated to each state.

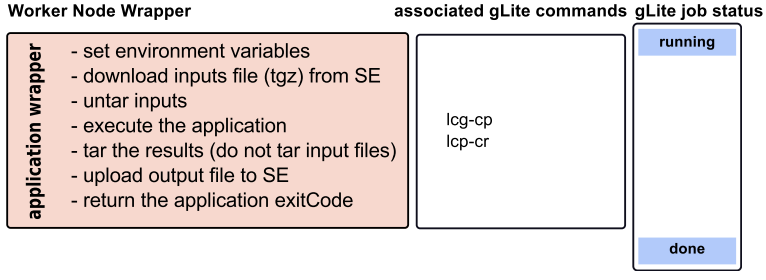

Fig. 5. In the Worker Node the user application is replaced by the application wrapper that downloads the inputs from the Storage Element, executes the user application, compresses the outputs and uploads the file to the Storage Element. Finally, the exit code returned to the WMS is the same the user application returned.

To solve this issue, we have explored several solutions: The gLite Logging and Bookkeeping service (LB) allows the user to be notified on every job status change so in principle we could use it to avoid queries alltogether. We did not get this service to work properly, and we had to continue querying the WMS to check the job status. Thus, a second version was developed (Figure 3 (2)). In this case the main script stops when the job is submitted and a single daemon (checkJobs) checks periodically the status of all the running jobs with an unique query per user to the WMS. This way, the number of queries to the WMS has been drastically reduced. Finally, when the daemon detects that the job status is "Done", it launches runGrid again to continue downloading the outputs and cleaning any auxiliary data.

Advanced users can use additional command line parameters like Worker Nodes memory requirements, specific Virtual Organization or set a particular outputs directory amongst others and configure specific user default values.

4 AUXILIARY COMMANDS

Job submission is not the only important action to be run on the grid. Users should know the status of their jobs and be able to cancel them or look up for job details. For all those secondary, but not less important functionalities some auxiliary scripts have been created.

The above mentioned *checkJobs* daemon has been essential to reduce the User Interface and the Workload Management System load, and more effort will be dedicated to solve the issues with the LB system and receive the job status notifications instead of pushing the system. This daemon queries the WMS in order to detect job finalization, and when detected, updates the FSM state to EXECUTED and launches the main script to continue the job workflow.

Another basic functionality apart from submitting jobs is canceling them. For that purpose, *cancelJobs* script allows users to delete all or some of the owned jobs. When canceling a job, the FSM moves to CANCELING to cancel the submission if needed, and then continues with CLEANING state removing any local or remote auxiliary data related to the job.

Last but not least, users should be able to check their jobs status, and *status* presents a table containing basic or detailed information (depending on the parameters) of the owned jobs and launched from that User Interface.

5 WEB4GRID

Web4Grid is a web interface that runs on the top of the scripts (Figure 6 a)), allowing users to submit jobs from every computer with a web browser without logging into a User Interface.

This user-friendly web interface has been developed with one of most known open source web application frameworks, Django [18]. Django's goal is the creation of complex database-driven websites. Besides, Django provides a large degree of mo-

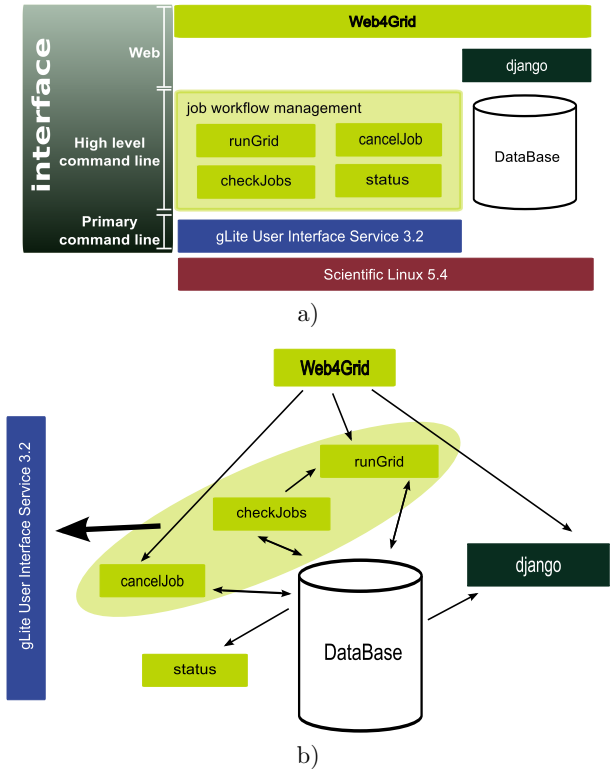

Fig. 6. a) User Interface layered software. The basic layers include the operating system and the gLite-UI packages which provide the low level command line interface. On top there are the scripts described in Section 4. Web4grid runs on the uppermost level. b) Interdependencies among the components: The command line scripts runGrid, checkJobs and cancelJob read and write to the database and access to the UI Services. Status and Django just read information from the database. Finally Web4grid interacts with the scripts to launch and cancel jobs and displays the status using the information provided by Django.

dularity which has turned out to be essential for our purposes; Web4Grid comprises the following modules: authentication, grid management, sessions and pagination. The authentication module currently used is LDAP [19]. We are considering the possibility to implement another authentication module to use the certificate stored in the browser as web and grid authentication and do not require researchers to physically store the certificate in a particular User Interface. This way, researchers from anywhere could access to any User Interface.

After authenticating, already submitted jobs are monitored through the web interface allowing the user, with just one click, to view a detailed description of each job, submit a new job, cancel one or more jobs and check the general grid status

(busy and free cores) allocated to each Virtual Organization the user belongs to (see Figure 7 a)).

Job submission options fit *runGrid* basic command line: users just select the application to be run and the related input files contained in their home folder. Finally they set the command line parameters (Figure 7 b)) and the job is ready to be submitted. As all the actions are launched through the web interface and are managed by the improved command line interface described above, all other essential information (VO, SE, CE, ...) is defined in the user profile and can be edited and updated any time the user is logged in.

As Web4Grid is working on the top of the command line scripts, outputs are stored in the application directory to allow users to manage parameterization results easily, but if they prefer to check the results locally, they can download them through the web interface.

a)

b)

Fig. 7. a) Job status and job details can be easily checked. b) Job submission through Web4Grid. After selecting the folder the application is placed in, the user is asked to select the application and then the input files and set the command line parameters.

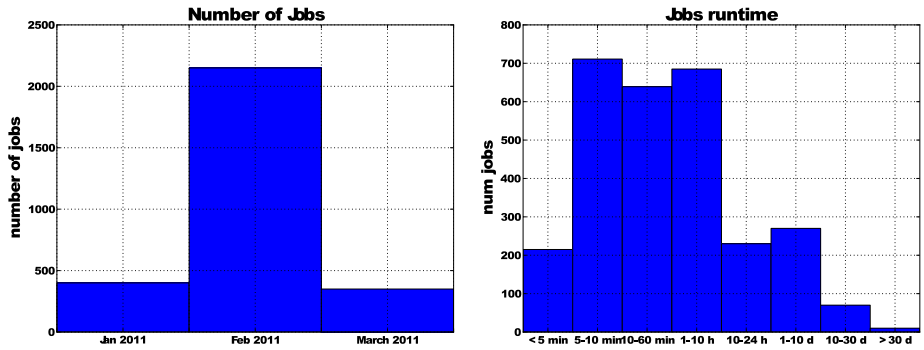

Fig. 8. The first image shows the total number of jobs ran with the high level command line interface. In the second image, the time the jobs lasted can be seen.

6 IMPLEMENTATION AND USER'S EXPERIENCE

We have implemented runGrid and Web4Grid at the IFISC User Interface. IFISC users are physicists working on a broad range of topics that within the context of complex systems include quantum transport, quantum and nonlinear optics, photonics, fluid dynamics, biological physics, nonlinear phenomena in ecology and physiology, complex networks and collective phenomena in social systems [20]. Numerical simulations are heavily used to model the dynamics of these systems. Users typically wrote down their own codes which use different algorithms ranging from pseudoespectral methods for spatio-temporal dynamics to correlations calculations and neighbourhood determination in on-line social networks. These programs were executed on a computer cluster [21] running MOSIX [22, 23] which was a quite user-friendly environment where users could submit their jobs to be distributed to the cluster. Home directories are fully integrated so that users have the same home directory in desktops and in the cluster. This allows users to edit the program in the desktop, compile and run in the cluster and analyse and visualize the data in the desktop or in a multiprocessor server.

When IFISC joined Grid-CSIC [24], we created runGrid with the aim to provide an environment which was user friendly and flexible enough so that it suits to all the users' programs. To facilitate the migration to grid, a seminar explaining basic ideas on grid computing and how to use the runGrid script was given. The runGrid script was initially deployed on June 2010. Following the users' suggestions we made several improvements in the script such as allowing for the use wildcards to list the input files or returning all the output files to the same directory where runGrid is invoked rather than to a subdirectory. We also increase the script robustness and fixed some initial bugs.

Although users could use both raw gLite command line interface and the high level one, all of them used runGrid. A total of 70 different applications have been

executed and the total number of jobs submitted using the scripts since June has been approximately 4500 (last months data is shown in Figure 8). As can be seen in Figure 8, in one-third of these jobs, long term proxies were a must because jobs lasted more than 12 hours.

The amount of applications executed and the large difference between them lead to different requirements. While some programs needed a few MB of memory, other expect at least 16 GB. Some applications were submitted to the grid just once, but other more than one thousand of times. Despite the variability of programs, we have not detected any major issues or handicaps using the high level command line interface.

7 CONCLUSIONS AND FURTHER WORK

We have developed a web application that would allow researchers to run generic grid jobs and monitor the grid status from any computer or mobile device provided it has a web browser. We aim at promoting the use of grid in scientific environments well beyond the traditional ones.

Besides we have also developed a high level script that allows for an easy job submission from the command line. The runGrid script is particularly useful for users that want to submit many programs using scripts. While there is some provision for job parametrization within gLite, we have not incorporated it in the runGrid script. The practice shows that different users have different needs for job parametrization and implementing all of that in a single script would lead to a cumbersome interface. Therefore it is more flexible that each user constructs scripts suitable to its needs. Still, runGrid provides a simple command to be called from these user scripts which takes care of all the burden of the grid submission and output files downloading allowing the user script to be simple and clean. We are currently working on an implementation of Web4Grid that would allow for these user scripts to be uploaded and executed from the web interface.

Users that need more advanced requirements can use more elaborated interfaces such as already existing ones.

To spread the use of this kind of portals, next steps should probably be the use of this kind of applications in large research centers or even having some portals per National Grid Initiative users with certificate could log in and submit their jobs.

Acknowledgements

Financial support from CSIC through project GRID-CSIC (Ref. 200450E494) and from from MICINN (Spain) and FEDER (EU) through project FIS2007-60327 (FISICOS) is acknowledged.

REFERENCES

[1] Enabling Grids for E-sciencE. http://www.eu-egee.org.

[2] gLite Grid Computing Middleware. http://glite.cern.ch.

[3] Kunszt, P. et al.: Data Storage, Access and Catalogs in gLite. Local To Global Data Interoperability – Challenges And Technologies, 2005, pp. 166–170.

[4] Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. Network and Parallel Computing Proceedings, Vol. 3779, 2005, pp. 2–13.

[5] European Grid Infrastructure. http://www.egi.eu.

[6] Wege, C.: Portal Server Technology. Journal of IEEE Internet Computing, Vol. 6, 2002, No. 3, pp. 73–77.

[7] Thomas, M. P. et al.: Grid Portal Architectures for Scientific Applications. Journal of Physics Conference Series, Vol. 16, 2005, pp. 596–600.

[8] Loureiro-Ferreira, N. et al.: e-NMR gLite Grid Enabled Infrastructure. IBERGRID: 4th Iberian Grid Infrastructure Conference Proceedings 2010, pp. 368–380.

[9] e-NMR project. http://www.e-nmr.eu.

[10] Moscicki, J. T. et al.: GANGA: A Tool for Computational-Task Management and Easy Access to Grid Resources. Journal of Computer Physics Communications, Vol. 180, 2009, No. 11, pp. 2303–2316.

[11] GANGA. http://ganga.web.cern.ch.

[12] Kupczyk, M. et al.: "Applications on Demand" as the Exploitation of the Migrating Desktop. Journal of Future Generation Computer Systems, Vol. 21, 2005, No. 1, pp. 37–44.

[13] Migrating Desktop. http://desktop.psnc.pl.

[14] Kacsuk, P.: P-GRADE Portal Family for Grid Infrastructures. Journal of Concurrency And Computation-Practice & Experience, Vol. 23, 2011, No. 3, Specuial Issue on SI, pp. 235–2454.

[15] P-GRADE. http://desktop.psnc.pl.

[16] Ludascher, B. et al.: Scientific Workflow Management and the Kepler System. Journal of Concurrency and Computation-Practice & Experience, Vol. 18, 2006, No. 10, pp. 1039–1065.

[17] Kepler. http://www.gridworkow.org/snips/gridworkow/space/Kepler.

[18] The Django Book. http://www.djangobook.com/en/2.0/.

[19] Koutsonikola, V.—VAKALI, A.: LDAP: Framework, Practices, and Trends. Journal of IEEE Internet Computing, Vol. 8, 2004, No. 5, pp. 66–72.

[20] IFISC research. http://i_sc.uib-csic.es/research/.

[21] Nuredduna. http://ifisc.uib-csic/nuredduna.

[22] Barak, A.—La'Adan, O.: The MOSIX Multicomputer Operating System for High Performance Cluster Computing. Journal of Future Generation Computer Systems, Vol. 13, 1998, No. 4-5, pp. 361–372.

[23] MOSIX. http://www.mosix.org.

[24] Grid-CSIC. http://www.grid.csic.es.

Antònia Tugores is an IT specialist at IFISC (CSIC-UIB) in Palma de Mallorca, Spain. She graduated in mathematics in 2003 at the Universitat de les Illes Balears. She worked developing graphical software in 2006. In 2008 she moved to GrisSystems, a leading commercial grid middleware company where she was involved in research projects such as NextGRID, @neurIST, BEinGRID and Espaa Virtual. Since March 2010 she works at IFISC in the project Grid-CSIC. Her main interests are grid and cloud computing and their applications in different areas.