Application Repository Based Evaluation of the EDGI Infrastructure

Adam Visegradi*, Sandor Acs*, Jozsef Kovacs*, Gabor Terstyanszky**
* MTA SZTAKI, Budapest, Hungary
{a.visegradi, acs, smith}@sztaki.hu

** University of Westminster, London, UK
G.Z.Terstyanszky@westminster.ac.uk

Abstract - The infrastructure set up by the EDGI EU FP7 project contains Desktop Grid (DG) sites (BOINC or XtremWeb) performing the execution of jobs coming from gLite, ARC or Unicore type service grids. The infrastructure contains an Application Repository (AR) as a central service storing all relevant information for applications. This AR is also the key to the gateways of the Desktop Grid sites, since enabling the execution of a given application on a DG site can be performed through the AR. The entire infrastructure has a monitoring system developed to collect statistical information about job execution. However, validating the information in the AR and testing job execution against the DG site was still missing. In order to evaluate the operation of the EDGI infrastructure, a new service has been designed and prototyped. This service collects all relevant information and submits jobs to Desktop Grids to gather data about their current state. This data can then be used by monitoring agents. A reporting webpage for the administrators is implemented. We will also show how reporting can be integrated with the Nagios system.

I. Introduction

A. The EDGI Infrastructure

The FP7 European Desktop Grid Initiative (EDGI) [1] project has created an infrastructure which integrates Desktop Grids (DG) with Service Grids (SG) in order to support European Grid Initiative (EGI) and National Grid Initiative (NGI) user communities to run applications which require large number of CPUs and cores. EDGI went beyond existing Distributed Computing Infrastructures (DCI) that typically incorporate cluster Grids and supercomputer Grids. It extended these grids with public and private Desktop Grids and Clouds. The project integrates software components of different cloud middleware (OpenNebula and OpenStack), Service Grid middleware (ARC, gLite, Unicore), and Desktop Grid middleware (BOINC[4] and XWHEP[7]) into SG→DG→Cloud platform for service provision. The EDGI infrastructure connects Service Grids with private, public and volunteer DG resources through the SG→DG bridge [2]. The project also provides access to Clouds from Desktop Grids via the DG→Cloud bridge to get additional resources for DG systems if the applications have QoS requirements that could not be satisfied by the available DG resources. EDGI deployed the production EDGI infrastructure that integrates ARC-, gLite- and Unicore-based Service Grids with Desktop Grids based on the EDGI bridge middleware. This production infrastructure also enables the dynamic and on-demand extensions of the connected Desktop Grids with Cloud resources. As a result, e-scientists can benefit of the flexible and versatile eco-system provided by the EDGI project (Fig. 1).

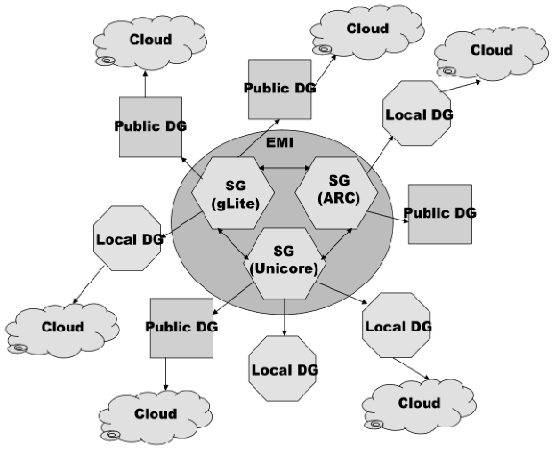


Figure 1. The EDGI Eco-system

The EDGI Infrastructure (Fig. 2) contains the EDGI Portal, the EDGI Application Repository, the SG→DG bridge and the Desktop and Service Grid resources. There are five major user types of the EDGI Infrastructure: E-scientists, Application Developers, Application Validators, Desktop Grid Administrators and Repository Administrator. The EDGI Portal is the GUI to submit and monitor applications, and retrieve and display results. The EDGI Application Repository stores the non-validated (private) and validated (public) applications. E-scientists can browse and search the repository in order to find applications they want to execute.

B. The EDGI Application Repository

The security models of Desktop and Service Grids are significantly different. While Service Grids trust the users and identify them by unique certificates, Desktop Grid systems trust the applications instead. As a result, only trusted and validated applications can run on DG systems. If users of a Service Grid infrastructure want to utilize

Desktop Grid resources then the applications should be trusted and pre-deployed on the supporting DG systems. This requires the application validation and uploading them into an application repository where both users and DG system administrators can access them. Moreover, the bridging mechanism from an SG to a DG system should always check that the application submitted via the bridge available and validated in the repository and identical to the submitted executable. The EDGI project designed and implemented the EDGI Application Repository, which is based on the EDGeS Application Repository (EDGeS AR), to support the full life-cycle of application validation, deployment and usage. As the focus of EDGeS was the creation of the production bridging mechanisms, the EDGeS Application Repository only provides the minimum set of required functionalities to bridge Service Grids to the target DG systems.

The repository is used by six actors, i.e. the six user types given above and the Modified Computing Elements. E-scientists are end-users who want to run applications on the EDGI Infrastructure. They want to search and browse the repository to find validated applications which they want to execute. Application Developers are computer scientists are familiar with infrastructure where the applications are executed. They elaborate these applications to enable e-scientists to run their applications on the EDGI infrastructure. To achieve it they have to be able to develop applications with their implementations and configurations. After implementing and uploading the applications into the repository they mark them as non-validated and they should notify Application Validators. If the developer modifies a validated application, then a new non-validated version is automatically created which should be validated. After significant updates, the developer could depreciate previous application versions after successful validation. Application Validators are also computer scientists who test applications created by application developers. They should find non-validated applications submitted by application developers, download packages and sample inputs and attempt to run the applications. After successful validation they give feedback about the applications and make them available for the Repository Administrator by marking them as validated. Desktop Grid Administrators manage Desktop Grid resources. They can search the repository and download validated application packages for deployment. They can allow users to use available resources by installing applications on the EDGI Infrastructure. Repository Administrator manages the repository, i.e. he/she handles users (registering/deleting and modifying their data).

The EDGI AR stores three types of components: applications, implementations and configurations. Application represents an application i.e. software. It describes the inputs and outputs and explains what the application does. However it does not actually contain any files necessary to run the application itself because there can be different implementations available e.g. for different operating systems. Implementation defines an implementation of an application. It strictly follows the input and output definitions of the application and implements the functionality given in the application description. It contains or references (via e.g. URLs) all the files and also holds other data/metadata necessary to run the application on a given platform. An implementation goes through a validation process and is eventually deployed on a resource. Implementations have a list of sites where they are or can be installed. Configuration describes in which Desktop Grid and/or Service Grid the implementation can be executed. It also stores files such as input and sample files, which are needed to run the applications.

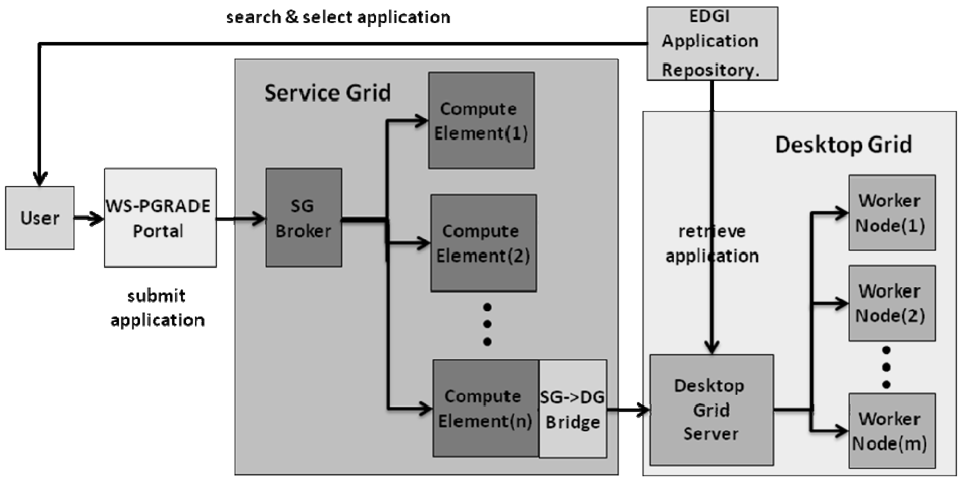


Figure 2. The EDGI Infrastructure

C. Consistency and Availability

An application registered in the Application Repository may be assigned to multiple backend grids. The AR and each of these grids may pertain to different domains of authority; they are managed by different members of the project. The consistency among them is not managed automatically; it must be enforced on a higher level, by project leaders. However, there were not any tools for easy administration.

We have developed a component which collects data from individual grids and compares it against the contents of the Application Repository. This information can be used by project managers and members to maintain consistency throughout the EDGI infrastructure.

If an application is consistently registered and deployed, it may still be unusable due to availability problems of the underlying grid. Automatic black-box testing of the infrastructure is necessary to maintain availability. The component that tests the infrastructure can use the consistency information to skip tests which will not succeed anyway.

II. EVALUATING CONSISTENCY

Our goal is to support maintenance of consistency between the Application Repository and each backend Desktop grid in the infrastructure. Backend grids are independent of each other; consistency between two – or generally, over any set – is undefined.

It is important to note, that Clouds are not included in the testing process. Clouds in the EDGI infrastructure are “only” used to provide additional dedicated resources for the various Desktop Grid backends. In our testing mechanism we consider the Desktop Grid as one powerful resource, but not going down to the level of individual resources. The correct operation and accessibility of a Cloud is realized by standard monitoring tools shipped with the cloud middleware.

First, we gather information from the backend grids. A modified computing element accesses the underlying grid through a bridge. Each bridge may be connected to multiple backend grids. Bridges do not support brokering incoming jobs; instead, the client must specify the target grid. For this, each bridge maintains multiple queues, each queue being associated with a single backend grid. Therefore, we can state that a target grid can be identified with a (Bridge, Queue) pair. Each grid may support multiple applications, whose list can be queried from the associated bridge. The structure of the information in a bridge is shown on Fig. 3. Gathering information from all known bridges results in a set containing (Bridge, Queue, AppName) triplets.

Grids to be evaluated and their associated (Bridge, Queue) pairs must be configured manually; there is no component which provides this information. Bridges have a web-service interface; the bridge can be identified by this service’s URL. The list of applications supported by a given Bridge/Queue pair can be queried through its WS-interface. Thus, data is generated in the following way:

1) (Grid name, (Bridge URL, Queue)) associations must be configured.

2) For each grid configured, the list of applications is queried from the bridge.

3) The result is a set of (Grid name, Bridge URL, Queue, AppName) tuples.

After extracting information from grids, we need to acquire information stored in the Application Repository. The structure of the information in the AR – the relevant part, thereof – is shown on Fig. 4. Each application – identified by its (canonical) name – may have multiple Implementations and each Implementation may have multiple GridIDs associated with it. A GridID is a tuple specifying the name of the supporting grid, and optionally, a bridge-specific name (GridAppName) for the application. The same application may have different names in different grids. If the GridID does not specify one, the GridAppName is the same as the canonical name. Normalizing this structure, we get a set of (Application, (Grid, GridAppName)) associations.

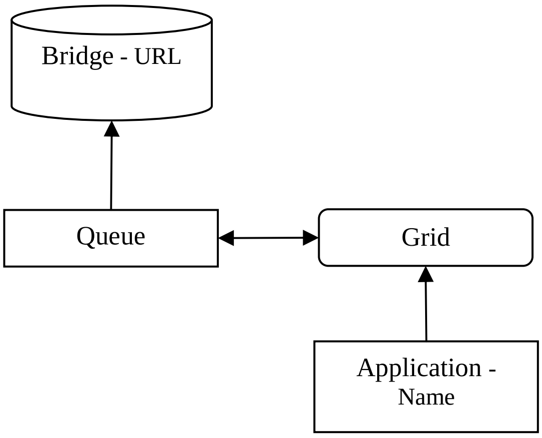


Figure 3. Bridge Information

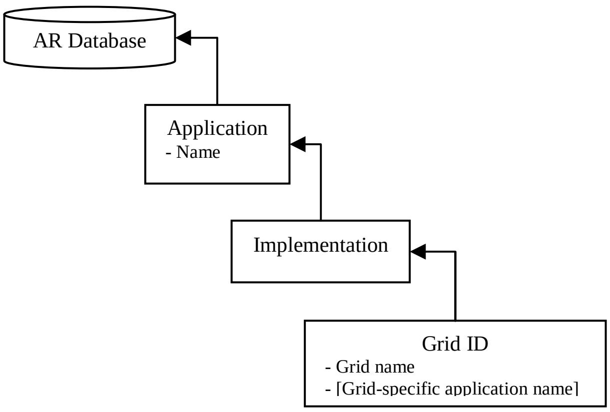


Figure 4. AR Information

We have to integrate these database schemas. The flattened schemas resulting from bridge and AR queries and the associations between them are shown on Fig. 5. Assuming that the two sets of information are consistent, they can be combined in a single dataset. However, sometimes AR and grid information is inconsistent due to infrastructural or administrative errors. The following inconsistencies and problems can be identified:

1) The AR is referencing a bridge that is unknown (i.e. not configured for evaluation).

2) The AR is referencing a bridge that is unavailable.

3) The AR is referencing an application which is not registered in the associated grid.

4) A bridge reports an application which is not referenced in the AR.

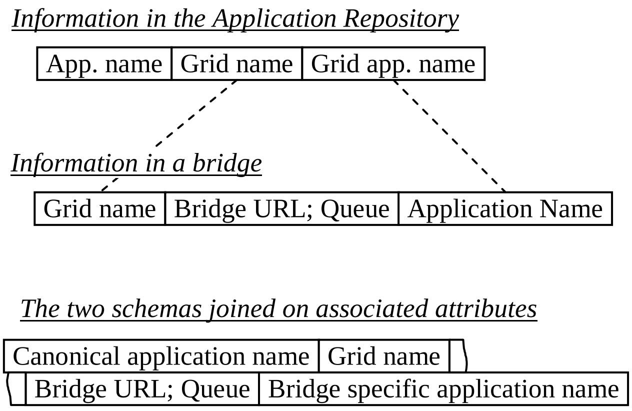


Figure 5. AR-Grid Information Associations

Identifying these problems and presenting them in a user-friendly way helps administrators and project leaders maintaining the infrastructure:

1) If the AR references a grid unknown to the testing system, the administrator can check the AR for a typo or may configure the new grid in the testing system.

2) System administrators can check firewall settings and service state if the bridge is reported to be unreachable.

3) If the AR references an application which is missing from the grid, if the grid administrator forgot to register that application, the test system can prompt him to correct the error.

4) Usually, it's not a problem, if an application is registered in a grid but is not registered in the AR; it may be the grid owners' own application. However, seeing this, the project leader may ask the grid administrators to make this application public by registering it in the AR, so other users can benefit from it.

Since installing our prototype system, the consistency and availability of the EDGI infrastructure has improved greatly. This system made it much easier to maintain the infrastructure and to coordinate participants of the project. Furthermore, the consistency information can be used to optimize the infrastructure testing by skipping tests which would not possibly succeed. A possible presentation of the results can be seen on Fig.6. The rows of the table represent applications, the columns represent grids. Each cell shows where that specific application is registered: AR/DG/AR+DG. Unreachable bridges, unknown grids and inconsistencies are highlighted.

III. Testing the Infrastructure

In the EDGI infrastructure, multiple smaller infrastructures are being integrated. While monitoring backend grids individually [6] is necessary, assessing availability of the whole integrated system cannot be simply done by testing its components. To measure the availability of the EDGI infrastructure, we have developed a simple component, which submits pre-configured test-jobs to the system periodically. This component does nothing else; interpreting the results is the responsibility of a higher layer.

Before submitting test-jobs, the result of the consistency analysis is checked. If a certain test-job is expected to fail because of a misconfiguration in the system or infrastructure outage, the test can be omitted. If the job may finish successfully, it is submitted and the results are recorded.

For each (Grid, Application) pair a test case can be defined. If a given application in a given grid is found to be consistently registered, both in the AR and the grid, the associated test can be executed. Practically, this means that each item of the joint schema (see Fig. 5.) may be associated with a test case. Each test is a black box for the testing system. For input, the joint information is provided. The output is expected to be a status code and – in case of an error – a message. Also, the test must have a timeout defined for that application. The testing system is entirely indifferent of the mechanism implemented in a test. It may be a gLite submission script or a direct submission to the bridge; the commands in the script may even be executed remotely on a UI machine. In the prototype, we implemented direct bridge submission from remote UI machine.

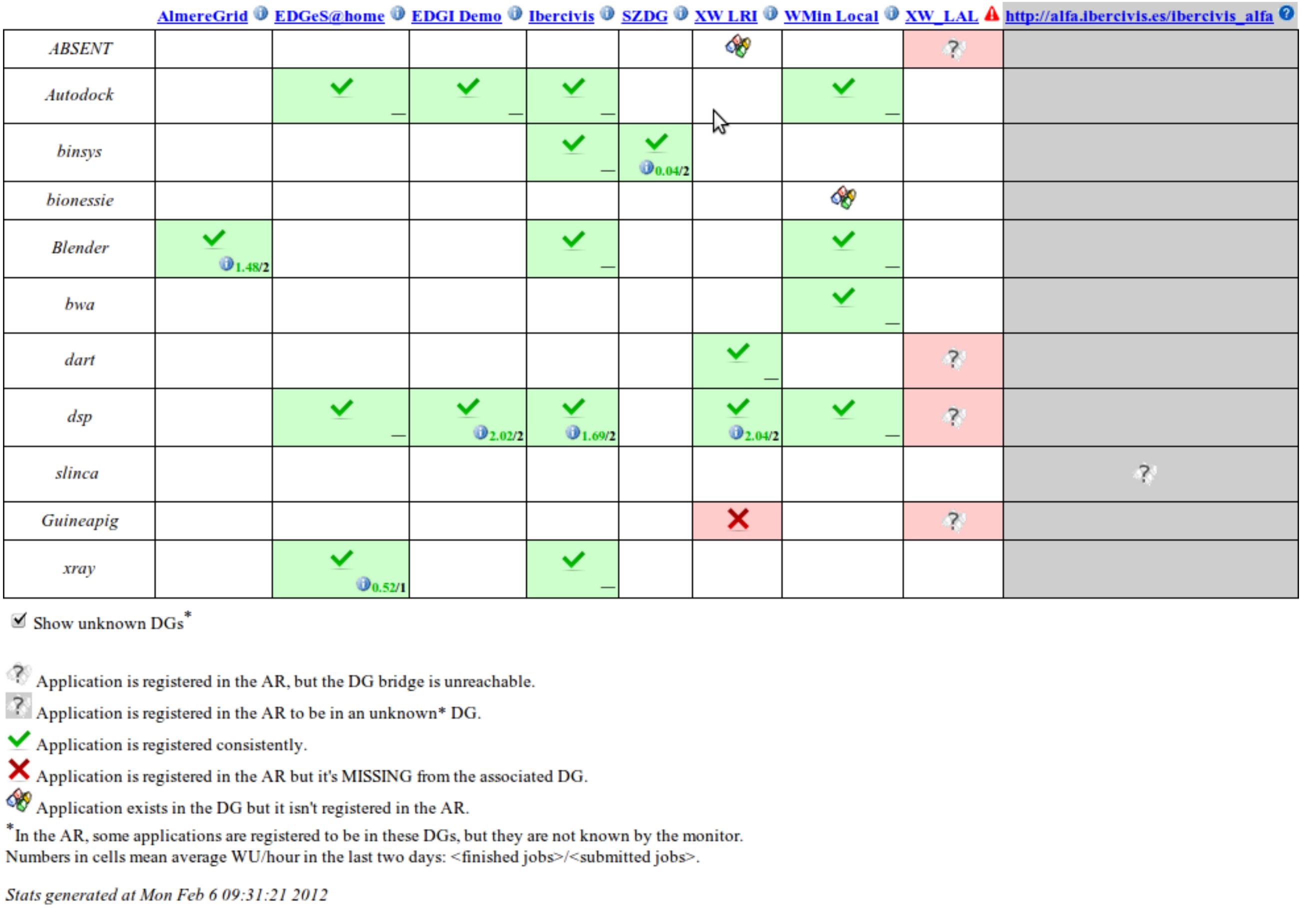
The test system will submit new tests periodically, and query the state of pending tests from time to time. In our prototype system, we submit two tests for each application every hour and query pending tests every thirty minutes. If a test does not finish before the defined timeout, the test is cancelled and reported to be failed.

1) Displaying the Results

Interpreting the results is not trivial; particularly in case of volunteer desktop grids. Volunteer desktop grids need much higher timeout for jobs than institutional desktop grids. While an institutional DG can finish jobs in a short time (i.e. within an hour), a volunteer DG may need almost a month or even more to provide results. For example, in case of the EDGIDemo DG, each hour two tests are submitted, and most of the time the results arrive before the next test submission. In contrast, the SZDG [8] volunteer DG has a thirty day timeout for jobs, which – depending on the load of the DG – usually proves to be too little.

High timeout for tests means high lag of the information provided by the testing system. Suppose we start testing a grid in which we have a 30 day timeout. In the worst-case scenario, we need to wait a month before we can decide, whether a particular test has failed. This means that whenever the infrastructure fails, the fact of failure can only be established 30 days after the actual event. Lowering the timeout would result in higher failure rates, as tests would be less likely to finish in time. Another way is to monitor the current output of the infrastructure. That is, because we provide a continuous load on the infrastructure, we can expect continuous output rate if we assume a uniform lead time of jobs. The problem here is the opposite of the previous. Whenever the infrastructure fails, we are notified almost immediately. However, when the system has recovered from a failure and testing has begun, we need to wait until the first tests start to finish. In the aspect of maintenance, this is a much better choice; even so because smaller outages will not affect the results at all. However, if we want to provide availability information to users, this method has the disadvantage of misinforming them while the system is "bootstrapping"; showing them the infrastructure is unreliable, while it is actually up and running.

In the prototype, we used the latter method. We show for each tested application, how many test jobs have finished in an hour on average through a certain timeframe. This is still problematic since the moving average will flatten the series a little, but with a well-chosen timeframe size, this information can still be useful. Also, volunteer and institutional DGs will produce series with different characteristics. In our experience, submitting two tests each hour, an institutional DG will finish around two jobs by hour on average, while a volunteer will finish around 0.04. The numbers produced by volunteer DGs can be explained with their (very) high load. Usually, the load in a volunteer DG is *kept* high, and because the workunits are processed in a FIFO fashion, the test jobs will not be executed before the test system cancels them due to timeout. We conjecture that introducing job-priorities and submitting test jobs with high priority would correct this anomaly.



	AlmereGrid	EDGeS@home	EDGI Demo	Ibercivis	SZDG	XW LRI	WMin Local	XW_LAL	http://alfa.ibercivis.es/ibercivis_alfa
ABSENT								?	
Autodock		✓	✓	✓			✓		
binsys				✓	✓ 0.04/2				
bionessie									
Blender	✓ 1.48/2			✓			✓		
bwa							✓		
dart						✓		?	
dsp		✓	✓ 2.02/2	✓ 1.69/2		✓ 2.04/2	✓	?	
slinca									?
Guineapig						✗		?	
xray		✓ 0.52/1		✓					

Show unknown DGs*

? Application is registered in the AR, but the DG bridge is unreachable.
? Application is registered in the AR to be in an unknown* DG.
✓ Application is registered consistently.
✗ Application is registered in the AR but it's MISSING from the associated DG.
Application exists in the DG but it isn't registered in the AR.
*In the AR, some applications are registered to be in these DGs, but they are not known by the monitor.
Numbers in cells mean average WU/hour in the last two days: <finished jobs>/<submitted jobs>.

Stats generated at Mon Feb 6 09:31:21 2012

Figure 6. Web Page Based on Consistency and Test Data

Because of the nature of the data, we have not created a system which interprets the data; instead we show the processed data on a web page, and let the administrators draw conclusions from it. A web page is shown on Fig.6. Each tested application has a number pair in its cell, where the first number is the hourly average number of finished tests from the last two days, and the second is the number of tests submitted each hour.

2) Using Test Results in Nagios

Nagios [5] is a powerful, scalable and flexible monitoring framework that enables to identify IT infrastructure problems. It is free software, licensed under the terms of the GNU General Public License version 2. Nagios already have lots of plugins for checking host resources (disk usage, processor load, etc.) and network services (HTTP, ICMP, FTP, SSH, etc.) because of its flexibility. Plugins can be written in many shell and program languages. We designed and implemented a plugin which can extend our monitoring system with alerting and alternate visualization back end. This plugin checks data generated by the test system periodically. It investigates the number of finished and failed jobs per day and sends alarms via e-mail for the administrators when expected conditions are not met.

IV. Conclusion

The EDGI infrastructure was designed to promote access to high capacity desktop grid resources. While we have most certainly reached this goal, the monitoring of the system was still unreliable. We have designed an automatic testing system, which

- Monitors consistency among the components of the EDGI infrastructure
- Creates availability data by regular submission of test-jobs to each grid

This information can be used in various ways. We have implemented two applications using this data: a web page, which helps project administrators to manage project participants; and a Nagios plugin which alerts system administrators on infrastructure failures.

Although automatic interpretation of the data is far from trivial, the UI provided by the system is sufficient for an administrator to maintain the EDGI infrastructure. Since the prototype system has been deployed, many anomalies and errors have been corrected; for example typos and misconfigurations in the Application Repository, missing applications from participating desktop grids, or—in case of most newly joined desktop grids—unopened firewalls. Also, the system allows us to notice outages in the infrastructure with small delay, thus these problems can be addressed quickly.

V. Future Work

Although our prototype system proved to be useful as-is, there are several things we can further improve.

The Application Repository may act as a grid information repository. Storing (Grid, Bridge URL, Queue) associations in the AR – in a central location, that is – would decrease the possibility of inconsistencies in the infrastructure. Also, manual configuration of the testing system would be unnecessary.

We will investigate the possibilities of high-priority job submission. We conjecture that doing so would make volunteer and institutional desktop grid results comparable; therefore, we could interpret the data automatically, creating an even more reliable monitoring system.

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 261556 (EDGI)

References

[1] EDGI project, [Online]. Available: http://edgi-project.eu/

[2] 3G Bridge, [Online]. Available: http://sourceforge.net/projects/edges-3g-bridge/

[3] Z. Farkas, P. Kacsuk, Z. Balaton, and G. Gombas, Interoperability of BOINC and EGEE, Future Generation Computer Systems, vol. 26, no. 8, pp. 1092 1103, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/B6V06-5046M26-8/2/b33ce4a0dc54204e34cdff1c90c577eb

[4] D. Anderson, Boinc: A system for public resource computing and storage. Proceedings of the 5th IEEE/ACM International GRID Workshop, pp. 1 7, 2004.

[5] Nagios, [Online]. http://www.nagios.org/

[6] F. Araujo, D. Santiago, D. Ferreira, J. Farinha, L. M. Silva, P. Domingues, E. Urbah, O. Lodygensky, A. Marosi, G. Gombas, Z. Balaton, Z. Farkas, and P. Kacsuk, Monitoring the edges project infrastructure, in 3rd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid 2009), Rome, Italy, May 2009.

[7] Fedak, G. et al.: XtremWeb: A Generic Global Computing Platform. In: Proceedings of 1st IEEE International Symposium on Cluster Computing and the Grid CCGRID 2001, Special Session Global Computing on Personal Devices, pp. 582 587, IEEE Press, May (2001)

[8] Kacsuk, P., Kovács, J., Farkas, Z., Marosi, A., Gombás, G., Balaton, Z.: SZTAKI Desktop Grid (SZDG): A Flexible and Scalable Desktop Grid System. J Grid Comput.: Special Issue: Volunteer Computing and Desktop Grids 7(4), 439 461 (2009). http://dx.doi.org/10.1007/s10723-009-9139-y