



IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

(Production) Testbed Monitoring Tools

Author(s):	Petr Holub, Martin Kuba, Luděk Matyska and Miroslav Ruda
Document Filename:	GridLab-5-D5.7-0007-Tool
Work package:	Testbed
Partner(s):	Masaryk University
Lead Partner:	Masaryk University
Config ID:	GridLab-5-D5.7-0007-1.0
Document classification:	DELIVERABLE

Abstract: This document describes the production tools used for monitoring the infrastructure and services of the GridLab project testbed. In some aspects it is an updated version of the previous related deliverable D5.6 "(Prototype) Testbed Monitoring Tools".





Contents

1	General overview	2
2	Test-Based Monitoring	3
2.1	Machine oriented tests	4
2.2	Database of historical records	6
2.3	GridLab web services tests	6
2.4	Web services unit tests	7
2.5	Matrix tests	7
2.6	Nightly builds of GridLab software	7
2.7	Deployment	8
2.8	Test dependencies	8
3	Monitoring Based on Monitoring Worms	8
3.1	Architecture	8
3.2	Incorporated Tests	10
4	Testbed Status Information Service	10
5	Summary	10

1 General overview

Permanent and continuous infrastructure monitoring is a necessary part of the management of any Grid. The Grid—as any heterogeneous distributed system without tight central administration—is an unreliable environment where the knowledge of state of its individual components and services is essential for production level quality perceived by applications and users.

The GridLab testbed (Grid) status monitoring are based both on the traditional monitoring architecture using the *pull* model and the novel approach, identifiable with the *push* model. In the former model, a set of programs—individual tests¹—is periodically run on a testing server and centrally collects individual components' states as reported by these programs. The later model is based on *monitoring worm* that migrates through the monitored infrastructure and reports back information about detected problems. Being a high level construct, the worm can detect problems arising from complex interaction between individual components—problems usually undetectable by the simple pull model.

A specific instance are *passive* tests, that use instrumented middleware to produce monitoring information while serving users' and applications' requests. They generally belong to the *push* model, but their primary advantage lies in negligible overhead, as their results are just by-products of the actual use of the Grid. We foresee increasing use of these tests when Grids will become used regularly, currently they must be complemented by active tests to cover periods of low Grid use. Also, the unavailability of appropriately instrumented middleware (together with related security issues) makes the active tests still more attractive. The next version of our production testing tools will include also passive tests based on instrumented GAT middleware, developed also as part of the GridLab project.

We implemented several collections of tests:

- Collection of *machine oriented* tests, i. e., services associated with individual machines are tested. The machine oriented tests are run in regular hourly intervals, with the possibility of unscheduled “on demand” tests on a particular machine initiated by anyone with a valid certificate (the certificate must be accepted by tested machine).
- Collection of *GridLab webservices oriented* tests, i. e., services developed by other GridLab workpackages, implemented as webservices with single or several instances in the whole testbed, are tested. If unit test is available for the webservice, we run also *unit tests* to verify their integrity and functionality. These tests are run also in regular hourly interval.
- *Matrix tests*, i. e., tests whether a particular service can perform an operation on any combination of machines in the testbed. The matrix tests are not run regularly, as they can produce too high load on the tested services. Currently, these tests are run as part of a pre-production test suite when a large demonstration or production-like runs are prepared. Gradually, the matrix tests should complement passive tests, but more precise strategy will be developed when more experience with passive tests and their interaction with active tests will be understood.
- Rather special category of the tests are *nightly builds*, that serve as a feedback to developers about quality and portability of code they are writing and also to eliminate problems as early as possible. The nightly build uses snapshot of code from GridLab CVS, which is the central repository for all software developed within the GridLab project, and performs it compilation on all sites or just some selected subset of sites in the GridLab testbed. Problems both with the code and infrastructure may be found this way.

¹For sake of clarity, we use the word “test” consistently as an active probe only running in pull mode.

To address issues of scalability and obtrusiveness of the pull model tests we have designed and implemented *monitoring worm*, that migrates through the testbed infrastructure as an ordinary job. Multiple worms could travel through the monitored infrastructure in parallel to achieve better scalability and robustness. The worms are taken care by *shepherds*, that supervise a set of worms, take action if a worm got stuck or disappears within the infrastructure and collect information from individual worms, The worm performs status checks on each mode it visits and reports discovered problems to its shepherd. The shepherd digests the information of its worms and passes this digest to the higher level of monitoring infrastructure (currently, due to rather limited size of the GridLab testbed, the information is passed directly to the central status monitoring service). The worm also sends heart-beat information to its shepherd which can spawn new worm in case that the old one dies or becomes unavailable. The total number of worms travelling through the Grid is controled through shepherds.

Results of all tests are collected, written to a relational database to keep historical records and displayed on the GridLab Administrative portal. A dynamic portlet integrated with GridLab GridSphere allows various views of both historical and the latest results. The test results are accessible either directly at [1] or through appropriate navigation on the GridLab web pages (starting from the GridLab main page <http://www.gridlab.org>).

2 Test-Based Monitoring

The GridLab test design is based on the following principles:

- modular design with pluggable tests
- multilanguage support for test coding, for individual tests implemented in any language
- scalability, able to support hundreds of machines/services
- limited parallelism (using thread pools)
- soft state (strict timing of individual tests, including proper test hang-out treatment)
- Globus independence (adherence to standards)
- history maintenance (writes results to a permanent storage and keeps also old records)
- easy configuration, with respect to machines, tests and users (their credentials)
- use of test dependencies (do not run tests that must fail).

The testing framework (Fig. 1) fulfilling all these requirements was developed in Java and is currently used to test the GridLab testbed status. It uses a configuration file which specifies (in XML format) the list of machines to be tested, list of tests to be performed, size of the thread pool and where to get user credentials. The scheduled tests could run under service credential, the unscheduled run either under the same credential (when initiated by administrator) or under the user's own credential (this is especially important if a problem with particular credential or its acceptance is expected).

Currently, the tests are run under the credential of one of the principal GridLab testbed administrators, for the following reasons:

1. Some sites do not accept other than personal certificates for users running applications (i. e., user must be a person, not an abstract entity).

2. Use of personal certificate ensures that the infrastructure behavior is exactly the same as for any other user (the service certificate may, in general, generate a slightly different behavior).

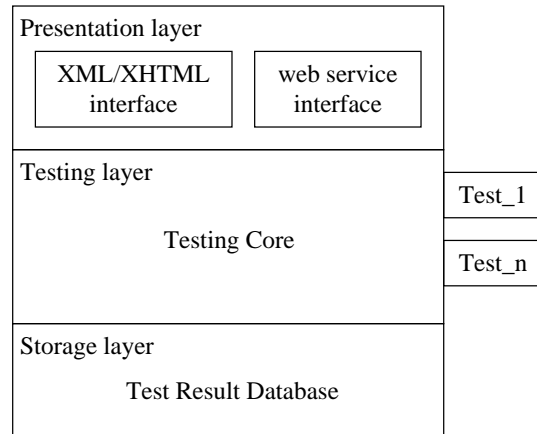


Figure 1: Architecture of GridLab Testbed Status Monitoring system.

While written in Java, the system supports multiple programming languages—currently shell scripts are used where a remote job must be submitted, otherwise Java is used. We use Java CoG for implementation of Globus clients. Small wrapper programs (in C) take care of timeouts and hang-up jobs, taking also care of all clean-up when a particular test does not finish correctly. Output from tests is written to a RDBMS, making history extraction an easy job. They are also written in XML. The most recent results are converted into XHTML using a default XSLT stylesheet, to provide a matrix representation.

2.1 Machine oriented tests

The GridLab testbed currently runs Globus 2 services², which means that all machines on the testbed must provide the following functionality:

- GRIS—tests whether each machine is running Grid Resource Information Service.
The test connects to each machine on port 2135, performs two unauthenticated LDAP queries with branch point `Mds-Vo-name=local,o=grid`. The first query is with scope `ONELEVEL_SCOPE` and filter `(objectClass=*)`, getting information about CPU and OS. The second query is with scope `SUBTREE_SCOPE` and filter `(objectClass=MdsServiceGram)`, searching for available jobmanagers. The test is implemented in Java using standard LDAP provider.
- GSI-FTP—tests whether each machine is running Grid File Transfer Protocol server.
The test connects to each machine to port 2811, performs GSI authentication to the Grid FTP server and then disconnects. The test is implemented in Java using Globus Java CoG Kit 1.1 classes.

²The GridLab testbed currently runs Globus 3.2-prewebservices instead of 2.2.4 used previously. However, the service set remains the same.

- Gatekeeper — tests whether Globus Gatekeeper (job submission service, providing access to one or more *jobmanagers*) is running on each machine.

The test tries to authenticate to a gatekeeper by calling `Gram.ping()` method. The test is implemented in Java using Globus Java CoG Kit 1.1 classes.

- GSI-SSH — tests whether each machine is running Secure Shell (SSH) daemon with Grid Security Infrastructure (GSI) extensions.

The test runs command `gsissh -p 2222 \ $MACHINE echo "TESTSTRING"` and searches for "TESTSTRING" in the output. The test is implemented by calling native `gsissh` client.

On top of these, a GIIS test is also performed (only one GIIS server per a Grid is required). This test performs one unauthenticated LDAP query on `ldap://mds.gridlab.org:2135/Mds-Vo-name=gridlab,o=grid` with scope `ONELEVEL_SCOPE` and filter `(objectClass=*)`. Gets machine name from each returned Distinguished Name (DN) by extracting the string between first equal sign and first comma in the DN. It is done this way to overcome misconfiguration of attribute `Mds-Host-Name` which sometimes doesn't contain fully qualified domain name, but just a short name of the machine. The test is implemented in Java using standard LDAP provider.

This basic set of tests was originally defined as part of the TeraGrid project [2]. We added the following tests to the the original TeraGrid collection:

- Accepted CAs — tests whether each machine has certificates and policy files of all required Certification Authorities.

It extracts all filenames from the CA tarball and submits a shell script to each remote machine to test existence of the same files in the directory `/etc/grid-security/certificates/`. This test is run only if Gatekeeper test was succesful. The test is implemented using shell scripting and native `globusrun` client.

- gridmapfile — tests whether each machine has required accounts in grid-mapfile.

The test gets all Distinguished Names (DN) from master grid-mapfile, reads grid-mapfile from each machine by submitting `/bin/cat /etc/grid-security/grid-mapfile`, extracts DNs from it and compares them to DNs from master grid-mapfile. This test is run only if Gatekeeper test was succesful. The test is implemented using shell scripting and native `globusrun` client.

- Installed software — tests whether required software is installed.

It submits a complex shell script to each remote machine, which first sources `/etc/gridlab.conf` file to set environment variables and then tries to find required software packages [3]. This test is run only if Gatekeeper test was succesful. The test is implemented using shell scripting and native `globusrun` client.

- MDS extensions — tests whether MDS schema extensions developed by the GridLab WP-10 were installed into GRIS on each machine.

The test connects to GRIS and performs an unauthenticated LDAP query which looks up object `GridLab-Mds-WebServices-Group=webservices,Mds-Host-hn=$MACHINE,Mds-Vo-name=local,o=grid, scope OBJECT_SCOPE`. Test succeeds when the object exists, fails otherwise. The test is implemented in Java using standard LDAP provider.

- MDS webservice — tests whether MDS WebService developed by the GridLab WP-10 is running on each machine.

It makes a SOAP call to operation `getServiceDescription()` in namespace `urn:mds` in webservice running on port 21000. The test is implemented in Java using Apache Axis and Globus 3 classes.

- mercury2— tests whether monitoring software, developed by the GridLab’s WP–11, is properly installed and running.

It runs command

```
monclient -n host.os.boottime -p host=$MACHINE -v monp://$MACHINE
```

for each machine. The test is implemented using native monitoring client.

- MPI tests— several tests, check whether MPI-enabled jobs, written in C or Fortran, can be successfully compiled and run on all jobmanagers
- Delphoi— tests functionality of the Delphoi adaptive service.

The test is implemented using native adaptive client and it runs command `control-pythia check -host $MACHINE` for each machine.

Detailed up-to-date description of all tests can be found on WP–5 web pages [4].

2.2 Database of historical records

All machine oriented tests’ results are stored in a database. This allows disconnecting the presentation layer, as the web pages with the test results are generated from the database and independently from the actual tests. This allows an easy deployment of even complex filters and continuous presentation of results (they are stored into the database as they are accepted, so even partial results can be presented). The database can serve as a data source not only for the web/portal presentation, but the results can be made available independently (e. g., through LDAP or SOAP)—we have implemented the web service interface as described in Sec 4.

Currently the results are displayed in two ways—either as the latest results of all tests on all machines, giving an overall overview of the testbed status, or as a history of a particular test on a particular machine, giving overview of its behavior in time. More possible ways of viewing the historical data are planned, like history of all tests for a given machine or history of all machines for a given test.

2.3 GridLab web services tests

With the continuous move towards the service-based model as represented by OGSA, the Grid status monitoring cannot be restricted to services associated with particular machines only. Currently, eight web services are already deployed on the GridLab testbed, and their status is also regularly monitored: Scenario Broker, Adaptive service, Metadata service, Replica Catalog, Data Movement Service, Data Browsing service, Authorization service, and Testbed Status Information Service (Sec. 4). All these web services are accessible over the HTTPG (HTTP over GSI) protocol. Each service has a method (function, operation) `getServiceDescription()` which returns a string. For each service, a simple C program, using gSOAP tool, is used to call this method and report on unavailability or unexpected output. These programs are invoked from a regularly dispatched shell script, which also creates a web page using the same color notation as the general status monitoring described above. However, currently the service testing is not integrated into the general monitoring framework.

An alternative implementation in Java (using portlets and GridSphere portal framework developed by GridLab’s WP–4 for presentation) was developed for running unscheduled “on demand”

tests. The dynamic nature of Java, as opposed to the static C gSOAP tool, enables testing of any webservice, accessible over HTTP or HTTPG protocol, if it has a suitable method (function, operation) to be called.

2.4 Web services unit tests

For those web services, where unit tests are available, we run also these to verify their functionality with respect to their specifications. Currently, the unit tests are used for:

- Replica Catalogue

The replica catalogue unit test is performed using native client called `replica-integrity-test` provided by WP-8 as a part of replica management software.

- Testbed Status Information Service

The unit test calls the three web service methods for Testbed Status Information Service (described in Sec. 4) and checks whether all the methods work without failure. The test is implemented using C and GSI enabled gSOAP.

2.5 Matrix tests

The major drawback of the pull model is that all machine/service functionality is checked from a central place. However, increased number of Grid applications involves inter-machine (and inter-service) operations. Examples are “Data Movement service” which needs to be able to copy files from any machine to any machine, or “GRMS service” which needs to be able to submit a job to any machine. A new set of tests checking whether a given machine can perform a given operation on a combination of machines has been developed. Results of a single test of a single service provide a *matrix*, with one, two, or possibly more dimensions.

Some preliminary implementation of these tests show that the number of machine combinations (obviously growing fast with number of machines) to be tested doesn’t allow regular (hourly, daily) testing. Matrix tests are currently used only irregularly, and a better strategy is being developed.

Also the ways how results of matrix tests are going to be presented to human users are investigated, because even simple case of history records for several tests with two-dimensional results produce a 4-dimensional matrix, which is difficult to present to users in an easily understandable way.

It is also possible to display results of “third parties” matrix-like tests. This approach is currently used to work with the Deplhoi system, a network monitoring developed also as part of the GridLab project (the Adaptive components workpackage WP-7).

2.6 Nightly builds of GridLab software

To support GridLab developers in their effort to write reliable and highly portable code, we have deployed nightly builds of selected software components that are run on all nodes or their selected subset in the GridLab testbed. For each build, the software is downloaded (“checked-out”) from the central GridLab CVS source code repository and built on selected nodes from scratch. Either the whole building process output or just errors are mailed to the developers. Currently, three GridLab applications use regular nightly builds:

- GAT (Grid Application Toolkit)

- Mercury2 (GridLab monitoring)
- Cactus (Cactus Project [5])

Another application that uses nightly building and unit testing strategy is the GridLab GridSphere. However, it is not tested and deployed widely on the testbed and we use a dedicated machine for it—i. e., the test functionality is used in this case for a different purpose than to monitor testbed infrastructure.

2.7 Deployment

Machine oriented and GridLab webservice tests are run every hour and the most recent results as well as historical records are available on the GridLab Administrative portal. An e-mail message is sent to appropriate administrator when some machine oriented tests fail. Individual tests can also be run unscheduled, on request from a site administrator or a user with a valid credential.

2.8 Test dependencies

There are dependencies among the tests, and it would be wasting of resources to run tests that must fail because of the problem already found by the previous test. For example, there is no point in trying to run an MPI job on a machine, if authorization against its Gatekeeper cannot be made, no jobmanagers are reported by its GRIS or the executable for the job cannot be compiled. So the configuration XML file for machine oriented tests can define dependency of a test on successful results of a set of other tests.

3 Monitoring Based on Monitoring Worms

The pull model based grid status monitoring with active tests poses additional load on the tested infrastructure and also doesn't scale very well [6]. However, relying solely on passive tests based on instrumented applications and middleware layers is also not good strategy unless the monitored grid is heavily used so that sufficient amount of data is regularly received. Therefore we have designed and implemented a monitoring architecture based on *worms* that travel through the grid and monitor its status. The worms can use both active tests and passive monitoring based on both instrumentation of the worm itself and middleware layers it uses.

3.1 Architecture

The architecture uses several layers as shown in Fig. 2: a monitoring worms layer, a layer of shepherds that control the worms and receive information from them, and testbed status monitoring service that is used for permanent storage of monitoring results.

Each worm is spawned by its shepherd using one of available job submission services (GridLab GRMS, Globus GRAM) and this step verifies functionality of the respective job submission service. During the time spent in the information gathering and monitoring, the worm sends results and heart-beat information to its shepherd. After that the worm asks for migration to the next host in the set predefined by its shepherd at the beginning of the worm's existence (Fig. 3). The migration can be done by means of migration-enabled middleware or via shepherd that resubmits the job to the next machine via job submission service again. The life-time of the worm counted by number of migrations is very limited to avoid virus-like behavior.

Each shepherd takes care of its "herd" of worms: starts new worms, restarts a worm when it dies. There are several ways to detect death of the worm:

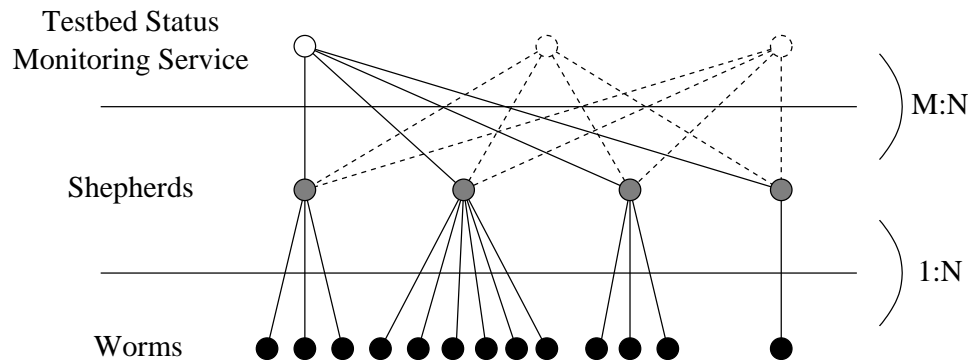


Figure 2: Monitoring architecture based on worms and shepherds.

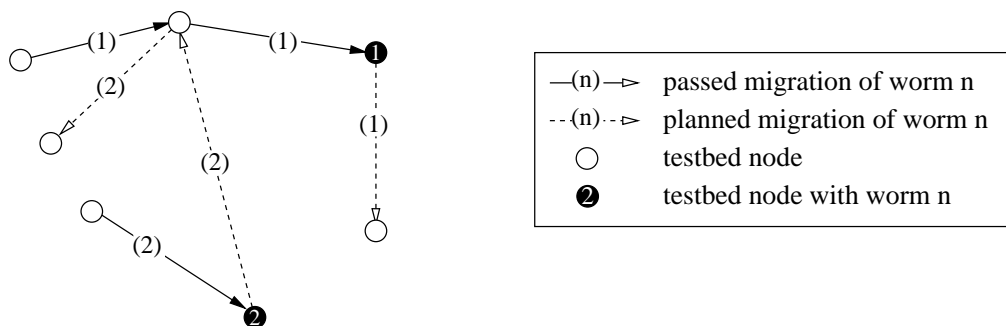


Figure 3: Monitoring worm migration scenario.

- the death is reported directly by the worm itself
- worm's heart-beat information is no longer received by the shepherd
- the shepherd polls job status of the worm's Globus job and thus it can diagnose when the job or even the job manager is lost

The shepherd reports information from the worms to the testbed status monitoring service that maintains persistent database of problems discovered. Each shepherd can be subscribed to multiple testbed status monitoring services to achieve more robust and fault-tolerant behavior, e.g. in case of grid infrastructure disintegration. Although shepherds support subscription to more status monitoring services, in the GridLab testbed that is rather small, there is just a single testbed status monitoring service and all shepherds communicate with this service only. The shepherds and thus also the worms can be run with service certificates for regular service monitoring and also with user's certificates to monitor the grid from the user's perspective in a way similar to traditional on-demand tests.

Whole system of worms and shepherds is implemented in pure ANSI C to achieve maximum portability on heterogenous grid infrastructures. Currently the worm doesn't use GAT but it will do so when a stable release of the GAT is available. The worm is ready to use the following components of the GAT: GRMS Adaptor, file transfer, and potentially also Pipes and Adverts.

3.2 Incorporated Tests

The worm is intended to imitate jobs running on the grid to discover problems before the real jobs fail because of them. The launching process of the worm itself is a complex test of all the services necessary for job submission: mainly GRMS and information services.

When the worm runs on a grid node, it checks for sanity of the environment which the real jobs expect. Currently, this is mainly check of valid and up-to-date `/etc/gridlab.conf` file used to set the environment, but more extensive tests can be added.

4 Testbed Status Information Service

Testbed status monitoring results are also accessible via web service interface to allow various grid services to utilize this information. The web service uses SOAP transport protocol with GSI security. However, it doesn't implement all OGSA as we wait for OGSA to stabilize after current shift in Globus team towards the WSRF. The service currently supports following methods:

- `getServiceDescription()`
standard method of all GridLab services for responsiveness tests,
- `getHosts()`
returns all hosts known to be on the testbed,
- `getUsableJobmanagers()`
returns jobmanagers known to be working for MPI and non-MPI jobs.

Currently, the GRMS service utilizes this interface to obtain list of machines that are available for job submission, machines and jobmanagers that support MPI jobs.

5 Summary

The current GridLab testing framework and the tools developed and/or deployed provide sufficient information about actual state of the Grid. The history stored in the database is used to track problems and is especially important as the GridLab operating centre (nor the site administrators) have a full 24x7 service and night and weekend problems are taken care of only on the next business day.

The information gathered by the monitoring tools is used to augment the information services and to check its validity [6]. We plan to extend this use to provide an independent and in the same time highly reliable information source for a restricted set information sources.

The test results are available in several forms—in computer-processable forms as static XML data or dynamic SOAP webservice, and in human-readable form as public web pages with easily understandable color boxes (green for success, red for failure, some other colors for timeouts and other exceptional results).

The webservices used in GridLab use GSI-secured transport, but are not OGSA compliant in the sense of portType inheritance, lifecycle, event notifications and service data introspection. The recent decision (announced on GlobusWorld in January 2004) of the Globus project to abandon OGSI (Open Grid Services Infrastructure) which was the foundation of OGSA and move towards WSRF (Web Service Resources Framework), which is not defined yet and is being developed together with Globus 4 (will be released in January 2005), left OGSA as highly moving target not stabilized for implementations. The GridLab webservices will be moved to OGSA when the evolution of WSRF permits it.

The testing worm takes care of complex testing, based on the push model of monitoring data gathering. The worm, being in fact just a specific application, tests a complex interaction of individual services and components that comprise the whole Grid. The hierarchical structure of worms and shepherds allows high scalability of this approach, keeping in the same time a reasonably low overhead on the grid resources. The scalability of this approach will be tested on some of the foreseen large scale demos.

Matrix tests are a specific case of the complex tests, testing specific set of services, but across the whole Grid. The matrix tests, while useful, pose too high load to be used in a regular way. We consider them as a complement to the passive tests coming from the middleware instrumentation. The current set of infrastructure monitoring tools still does not include passive tests based on middleware instrumentation. We are, however, working with the GAT developers to add new service interfaces to the middleware—these interfaces will provide information that will be collected by the specific version of the worm, processed and sent via shepherds to the central monitoring service. This way the use of the instrumented middleware will have no negative effect on the applications (no direct interaction with a remote service will be necessary).

As already pointed out in Sec. 3 and more thoroughly discussed in [6], the status monitoring based on (active) tests is suboptimal since it poses additional load on the infrastructure tested. Less intrusive strategy is the passive monitoring that relies on information sent by applications and instrumented middleware running on the grid. This is not yet included in the current set of monitoring tools, but we are working with the GAT developers to add new service interfaces to the middleware. These interfaces will provide information that will be collected by the specific version of the worm, processed by it and sent via shepherds to the central monitoring service. This way the use of the instrumented middleware will have no negative effect on the applications (no direct interaction with a remote service will be necessary). And worms can also start specific local tests if there is insufficient information collected by the instrumented middleware (e. g., because of low application load).

References

- [1] <http://www.gridlab.org/WorkPackages/wp-5/testbed/status.html>
- [2] <http://www.ncsa.uiuc.edu/~jbasney/teragrid-setup-test.html>
- [3] <http://www.gridlab.org/WorkPackages/wp-5/admin/packages.html>
- [4] <http://www.gridlab.org/WorkPackages/wp-5/testbed/testsdescription.html>
- [5] Cactus Project, <http://www.cactuscode.org/>
- [6] P. Holub, M. Kuba, L. Matyska, and M. Ruda: “Grid infrastructure monitoring as reliable information service.” In *The 2nd European Across Grids Conference*, Nicosia, Cyprus, January 2004.