Grid Infrastructure Monitoring as Reliable Information Service

Petr Holub, Martin Kuba, Luděk Matyska, and Miroslav Ruda

Institute of Computer Science, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
{hopet,makub,ludek,ruda}@ics.muni.cz

Abstract. A short overview of Grid infrastructure status monitoring is given followed by a discussion of key concepts for advanced status monitoring systems: passive information gathering based on direct application instrumentation, indirect one based on service and middleware instrumentation, multidimensional matrix testing, and on-demand active testing using non-dedicated user identities. We also propose an idea of augmenting information provided traditionally using Grid information services by information from the infrastructure status monitoring which gives verified and thus valid information only. The approach is demonstrated using a Testbed Status Monitoring Tool prototype developed for a GridLab project.

1 Introduction

A large-scale heterogeneous Grid is subject to frequent changes and service disruptions of some of the huge number of components that constitute this environment. Grid management requires on-line monitoring of the resources to determine their state and availability. The less dynamic data about the Grid resources are usually published by the resources themselves using information services (e. g. Meta Directory Service from Globus package [2]), while the highly dynamic data are usually published via different means (some implementation of the general Grid Monitoring Architecture, GMA [1]). The information provided by both of these approaches is used by other Grid services for discovery of elements of the environment and the ways these elements can be used.

The general Grid monitoring covers two complementary areas: application monitoring and infrastructure monitoring. Both can be subdivided into performance monitoring and status monitoring. Application performance monitoring, application status monitoring, and infrastructure performance monitoring are out of scope of this paper. We are dealing with Grid infrastructure status monitoring, which will be called *status monitoring* for brevity. Status monitoring covers infrastructure sanity checks, availability and interoperability of computers, other components, as well as web services and Grid services [5].

In this paper we introduce an idea of augmenting the information service by the status monitoring service. The augmentation is performed by verifying the information provided by the information service using the data from the status

monitoring, and allowing validated results only. Such model is ready for testing within the GridLab project where GridLab Testbed Status Service [3] will be used as one of information sources. A description of GridLab Testbed Status Monitoring Tool prototype is also covered by this paper. Roughly, the primary goal for the information service augmentation is a provision of enough data supporting production-quality Grid management; another goal is a verification and correction of static data provided by the "classical" information services.

2 Infrastructure Status Monitoring

Classical infrastructure status monitoring uses so called active tests, run by the monitoring service itself. These tests are fully under the control of the status monitoring service, but they pose additional load on the monitored resources (some sensors must run on the resources, the monitoring data are transmitted over the network, etc.). The alternative is passive monitoring[1] that gathers monitoring data directly from the users' applications running on the Grid (e. g. large data transfers). This passive monitoring poses only negligible additional load on the Grid but it is irregular and of very heterogeneous nature. Therefore, a perfect status monitoring system should complement the passive monitoring with active tests in cases where enough data are not available (e. g. due to longer period of user inactivity).

Both monitoring approaches use sensors as the primary data sources (generators). They can be run either in "push" mode, when sensor continuously sends data to any subscribed monitoring service, or in "pull" mode when the sensor is polled each time the data is required by monitoring service.

Passive monitoring model was based on an idea of an application instrumentation, i. e. a specific code is incorporated into the applications to report monitoring data. With the monolithic application model this means that developers themselves must include the instrumentation calls into their own code. Use of instrumented libraries linked at compile time or even better at run time is a step toward less obtrusive monitoring. With the increased use of service-based or middleware-based applications the instrumentation calls can be hidden within the service code or in the middleware and the application developers or the users do not have to be even aware of the fact that the whole program or its specific run is instrumented. The application based instrumentation (either directly in the application code or within the higher layers of the service code) has one major advantage—it can serve as a source of very complex monitoring data. The successful completion of a high-level application request (e. g. a service or even a set of services) is simultaneously a proof of proper function of all the services used as well as the lower middleware layers. A simple message about successful service call covers a success of many related (lower) services, increasing substantially scalability of the monitoring by allowing the system to omit individual tests that are implicitly successful. On the other hand, a failure can

[1] For the sake of clarity we don't use word "test" in the context of passive information gathering in this paper and we reserve it for active tests.

immediately trigger more detailed status tests of called services, thus leading to very fast failure discovery.

Traditionally status monitoring has been performed on machine oriented basis: either the machines sent continuously some data, or machine by machine was tested whether all required and advertised services are running. Introduction of widely deployed services within the Open Grid Services Architecture [5] framework leads to the necessity to perform also the service oriented monitoring to check that all services are running properly. Most services run in a machine independent way, i. e. they are not tightly associated with a particular machine and can run in several instances simultaneously or can be easily re-started on any machine from a subset of Grid machines. This means that a service—not a machine—must be tested. Testing of a service means that the service must be found (discovered), connected to and actually tested (run) dynamically, with no stored (cached) information about the placement of the service.

For many services it is important to test availability on N-to-N or M-to-N basis (where M might be subset of N or it might be completely different and independent set) instead of 1-to-N basis. For instance data transfer service may be required to be capable of transferring data from any node to any node (or even a set of nodes). Or a user may be required to be able to login from any node within a selected subset of Grid nodes to any node from different subset. As a first step towards more complex solution, we have introduced matrix tests that perform either M-to-N or N-to-N tests to cover such cases.

Another important aspect of the status monitoring is whether the tests run using some dedicated testing identity (e. g. using dedicated service certificate) or whether the tests are run using ordinary users' identities. While the second option is rather automatic for instrumented checks, it must be ensured that similar way is possible for active checks as well. It is also desirable to be able to run active tests on demand under specific user identity as this provides means for looking for a cause of a problem encountered by particular users only.

3 Monitoring Services as Information Services Augmentation

Information services traditionally advertise information about resources available on the Grid (machines and their capabilities, job submission mechanisms, services etc.). Owners of the resources or directly the resources usually publish such information about the resources into information service without any validation, and therefore obsolete or invalid information can be easily advertised.

Status monitoring can augment traditional information services by taking information published in them, checking validity and then publishing results in form of verified and thus authoritative information about resources using interface common to information services.

Another problem with the Grid information services is that the most common information service—Globus MDS—also shows serious performance bottleneck when too many non-homogeneous information providers are subscribed to it (esp.

with high percentage of "problematic" ones, like information providers behind firewalls or incompatible versions of software clients). Status monitoring service can mitigate these problems by providing "cached" and valid only data from such service.

4 Prototype Implementation

GridLab Testbed Status Monitoring Tool prototype [3] has been designed to test availability and sanity of Grid environment (this system is not used for performance monitoring as this is covered by Mercury [4], which is other part of the GridLab project). Very first version of this tool was based on monitoring tool available from TeraGrid Project [6]. This monitoring tool comprised a single threaded Perl script performing all tests in sequential order, making this solution not scalable. Also adding new tests was not easy and decreased the scalability even further. Based on the experience gained with this and similar tools we created new design of the testbed monitoring tool.

Current prototype has a core written in Java language and uses clean layered architecture shown in Fig. 1. We not only made testing independent of other parts, we also split a storage of results from the presentation, providing very high system flexibility. System is easily configurable using XML language. This architecture is scalable enough for small to medium size Grids, with at most few hundreds of nodes. The system may need to be enhanced with hierarchical distributed setup for larger Grids, or with robust features from peer-to-peer networks if fault tolerance is of a high concern.

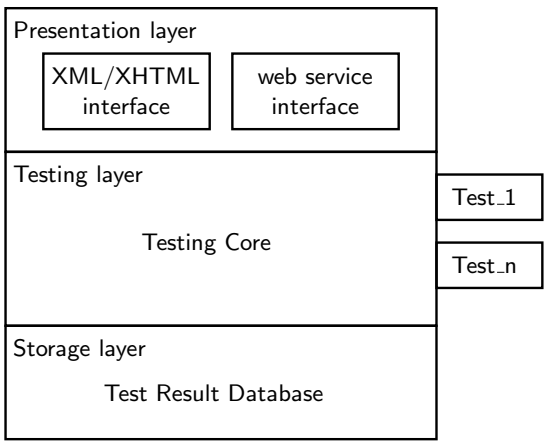

Fig. 1. GridLab testbed status monitoring architecture

Testing layer. The current GridLab testbed status monitoring is still a centralized activity, which means all the tests are initiated from one site and all the results are gathered there. While the sequential run of individual tests is inefficient, the fully parallel run of all tests is also impossible for larger Grid infrastructure (with fifty or more resources in the testbed). The fully parallel run may not only overload the testing machine, it may also pose an unacceptable load on the whole Grid infrastructure. Therefore, we use thread pool of configurable size to perform tests in a limited configurable parallel setup providing us compromise between load on both testing and tested infrastructure and scalability needed.

The testing layer is exposed through a language independent interface which makes it possible to implement new tests in virtually any programming language. For example current tests based on Globus 2 protocols have been written either using Java CoG or using small C wrapper for executing binaries written in other languages. The wrapper takes care of timeouts for hung-up jobs and clean-up for incorrectly finished ones, making the monitoring tool prototype resistant to failures. Especially hang-ups caused by firewalls incorrectly discarding packets showed to be a constant problem in Grid environment.

Test dependencies have been implemented that allow skipping of tests for which some mandatory prerequisite has failed. This decreases the unnecessary load on the tested Grid infrastructure. A language for general description of more complex dependencies is under development. We plan to use the same approach for the passive monitoring based on service and application instrumentation (the description of dependencies is crucial in such environment).

The architecture also supports a user triggered on-demand tests. These tests run under users' own identities and provide valuable data for problem and bug tracking.

Storage layer. The regular tests are run periodically and their results are stored by the storage layer. As the history must be made available for inspection, the data are stored in a database. While currently a PostgreSQL database is used, any other relational database can be easily substituted since JDBC database interface is employed.

Presentation layer. The presentation layer supports both static and dynamic web pages creation. For static results presentation, the test results are converted to XML and then transformed to a static XHTML page using XSLT processing. The XHTML page mimics the original TeraGrid tests layout and provides good general overview of the current infrastructure status in the form of a full matrix of colored boxes.

For dynamic results presentation integrated in a Grid portal, GridSphere [7] portlet based interface with multi-lingual capabilities has also been implemented. This interface supports browsing through status data history.

Test results are available via web service interface as well which allows for using this monitoring service as an information service by other services on the Grid. This specific web service uses SOAP transport protocol with GSI security

and the implementation is moving towards OGSA compliance (now lacking few of the required Grid service interfaces).

4.1 Incorporated Tests

While all currently implemented tests use active pull model, our general status monitoring framework supports easy integration of passive monitoring as well. Active tests were chosen because of faster and easier implementation compared to passive monitoring and especially applications, services and middleware instrumentation. At this stage of the development, the "production" application runs are still a minority on the GridLab testbed, which means that passive monitoring can not yet be a major source of monitoring information. All tests are run on regular scheduled basis with possible activation on demand by users using their own identities.

Simple tests. A test from the simple test category produces a scalar value for each tested machine. The prototype currently incorporates all tests available in the original TeraGrid software and adds also several new tests:

- Globus-2 tests: GRIS, GSI-FTP, Gatekeeper, GSI-SSH, and GIIS,
- availability of MPI C and MPI Fortran compilers,
- job manager tests: tests all job managers advertised in information services, whether they can run both normal and MPI jobs,
- GridLab specific tests: check on accepted CAs (whether compliant to GridLab requirements), check whether required software is installed and really working (C, C++, CVS, F90, GNU make, Perl, Java), check whether `grid-mapfile` contains all required users, check GridLab Mercury [4], and GridLab MDS Extensions and MDS web service [8].

Except for GIIS which is tested once per GIIS server (in the case of GridLab only once for the whole testbed since there is only one GIIS server in the testbed), all other tests run on per machine basis.

The simple tests on GridLab testbed currently take about 15 minutes to test all of 17 services on all 19 machines using 6 concurrent threads. The time is spent mostly in waiting for response due to delays in network communication and in waiting for timeouts, because the widespread use of firewalls leaves no way to distinguish a slow responding service from unavailable service other than waiting for a timeout. The only notable CPU load is due to authentication and encryption in GSI communication.

Service tests. With OGSA model that is generally seen as the next generation model for the Grid environment, Grid services become cornerstones of Grid infrastructure. Therefore service oriented tests are appropriate solution for monitoring infrastructure based on this paradigm. Services may run on various Grid nodes and the important issue is whether service is running correctly and not whether the service runs on one particular machine. Another fact supporting

approach different from machine oriented tests is that different machines will run different subsets of services and eventually the matrix of host and services may become quite sparse.

All the services produced by GridLab are persistent GSI-secured web services. It means they are accessible using HTTPG protocol (HTTP over GSI). Invocation of the web service methods by the testing tool is implemented using either Java CoG or C program using gSOAP tool [9] with GSI plug-in. The services support Grid security features, however to full OGSA compatibility they lack `portType` inheritance, because most of them are implemented in C, and there is no C server-side implementation of OGSA available yet.

In the first stage the service status monitoring checks whether the service is responsive. We could not rely on all services's API being inherited from a single `portType`, so we require that each GridLab service must provide an operation called `getServiceDescription()` returning a string containing service description. The responsiveness of a service is checked by calling this operation.

We have developed first stage tests for the following GridLab web services: GRMS, Adaptive service, Meta-data service, Replica Catalog, Data Movement, Data Browsing, Authorization, Message Box Service, and Testbed Status.

Second stage is aimed at verifying whether service is operational and performs as expected. Actual tests differ largely from service to service. Up to now second stage tests for Data Movement service and GRMS have been implemented.

The service tests on GridLab testbed currently take just several seconds to test all 9 services.

Matrix tests. Up to now two matrix tests have been implemented for GridLab infrastructure: Data Movement service test and GSI-SSH tests. The first one checks correct operation of Data Movement service between all pairs of nodes in an N-to-N fashion thus forming two-dimensional matrix of results. The test can be also easily extended to third dimension accommodating the possibility that data transfer can be initiated from a third node, i. e. node that is neither source nor target of the data being transferred. This example demonstrates problem with extreme load growth imposed on underlying infrastructure when complex active measurements and tests are put in use.

The GSI-SSH test checks whether it is possible to login from one node to another node. The test can work in either full N-to-N or in M-to-N fashion since only a selected subset of Grid nodes can be allowed to initiate SSH connection to the rest of the Grid.

While the matrix tests are not scalable, they provide invaluable information about the "real" Grid status. The current history of use of the data movement test had shown that it is almost perfect source of monitoring information about node mutual interoperability (as opposed to the 1-to-N centralized tests which check just interoperability between the testing machine and each node). The matrix tests reflect much better the actual situation users encounter when using a Grid and are able to find very specific and subtle problems (e. g. various incompatible firewall configurations). These tests have also character of complex tests that are similar to high level application tests (see Sec. 2), which means

that if the test passes correctly all lower layers and services are verified as well. If failure of such test is experienced, specific lower level tests can be immediately triggered to identify precise source of the problem. For example Data Movement matrix tests will not run without firewalls set up correctly, `grid-mapfile` installed properly etc. This complex property allows to omit a lot of other tests thus compensating the scalability issue to some extent.

The matrix tests on GridLab testbed currently take about 2 hours to test a full matrix of 17×17 data transfers among all 17 machines, most of the time is again spent waiting for timeouts caused by firewalls.

We expect that most of the inter-node tests required by full matrix setup could be replaced by the passive monitoring information when the Grid is used for "real" production (the applications will become actual data sources). This will add the necessary scalability to this kind of tests.

5 Future Work

Current status monitoring prototype tool mostly implements active tests in bottom-up fashion, i. e. testing starts from low level tests and proceeds to higher levels only if lower level prerequisites are successfully tested. For future work we are targeting opposite approach in which tests of lower level services and layers will be triggered only when higher level test fails to allow more precise identification of source of problems. This approach will be enabled by employing high degree of passive monitoring based on instrumentation of applications, services, and various middleware layers resulting in lower load on Grid infrastructure induced by monitoring itself. Heavier use of push mode sensors goes hand in hand with deployment of passive monitoring model which results in far more scalable and inobtrusive monitoring solution.

We plan to extend the Grid Application Toolkit [10] (the specific middleware layer connecting transparently applications with lower layers of Grid infrastructure) which is developed within the GridLab project with instrumented interfaces that will allow use of applications as monitoring data providers. In the same time we plan to use this instrumented layer to develop a monitoring worm that will "travel" autonomously through he Grid, gathering the monitoring information from used middleware components and nodes and sending this information (probably in a digested form when no error is to be reported) to some central site. The travel of the worm will be accomplished in close collaboration with all the middleware components (resource discovery, resource brokerage, job submission service etc.), thus testing extensively the Grid environment as a whole. The combination of active tests and passive monitoring with the data provided by the (regular, random or user triggered) worm reports should cover the whole Grid with a minimal obtrusive overhead. Understanding the interactions of these monitoring components will be subject of our future study.

We also want to build a database of typical problems occurring in the Grid environment. It will be used to produce better explanation of problems detected, thus improving understanding of the test results by end users.

6 Conclusion

A Grid infrastructure status monitoring system is an essential component of any Grid that aims to provide a usable working environment. Such a system is also a core of the Grid management, including information provision and validation for resource management on the Grid. The status monitoring system being developed within the EU GridLab project is one of the most comprehensive status monitoring systems currently deployed on a really large scale heterogeneous Grid. As not only individual components, but also emerging Grid services are permanently monitored, it represents a preliminary version of an OGSA compliant status monitoring system. Another advantage of this system is its use as information service augmentation and verification tool, providing a guarantee for a reliable information provided by a general information service (like the MDS).

The ability to define test dependencies in the monitoring system decreases monitoring overhead on the Grid infrastructure through elimination of the tests known in advance to fail. As a complement to this approach, we introduced some very high level tests whose successful completion signalizes that all the lower layers are working and eliminates necessity of individual tests. An example of such higher level tests that is already used on the GridLab testbed is the Data Movement service test. Even the N^2 complexity of this test is not prohibitive as it can potentially replace a large bunch of simpler, but also obtrusive tests. These will be needed only for a targeted inspection when the higher level tests fail.

The passive, service instrumentation based monitoring is another part of the whole Grid monitoring system. While not discussed in this paper to much extent, they may eventually replace most of the active (monitoring system triggered) tests and thus keeping the overhead of the Grid monitoring to the acceptable level even in very large Grids.

Acknowledgement. This work is supported by the European Commission, grant IST–2001–32133 (GridLab).

References

1. Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V., Wolski, R.: A Grid Monitoring Architecture. GGF Technical Report GFD-I.7, January 2002. `http://www.gridforum.org/Documents/GFD/GFD-I.7.pdf`
2. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In Proceedings of the 10^{th} IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
3. Holub, P., Kuba, M., Matyska, L., Ruda, M.: GridLab Testbed Monitoring – Prototype Tool. Deliverable 5.6, GridLab Project (IST–2001–32133), 2003. `http://www.gridlab.org/Resources/Deliverables/D5.6.pdf`
4. Balaton, Z., Gombás, G.: Resource and Job Monitoring. In the Grid. Proc. of the Euro-Par 2003 International Conference, Klagenfurt, 2003. 404–411

5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002. http://www.globus.org/research/papers.html#OGSA
6. Basney, J., Greenseid, J.: NCSA TestGrid Project: Grid Status Test. http://grid.ncsa.uiuc.edu/test/grid-status-test/
7. Novotny, J., Russell, M., Wehrens, O.: GridSphere: A Portal Framework for Building Collaborations. 1st International Workshop on Middleware for Grid Computing, Rio de Janeiro, June 15 2003.
8. Aloisio, G., Cafaro, M., Epicoco, I., Lezzi, D., Mirto, M., Mocavero, S., Pati, S.: First GridLabMDS Release. Deliverable 10.3, GridLab Project (IST–2001–32133), 2002. http://www.gridlab.org/Resources/Deliverables/D10.3c.pdf
9. van Engelen, R. A., Gallivan, K. A.: The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. In the proceedings of IEEE CCGrid Conference 2002.
10. Allen, G., Davis, K., Dolkas, K. N., Doulamis, N. D., Goodale, T., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Radke, T., Russell, M., Seidel, E., Shalf, J., Taylor, I.: Enabling Applications on the Grid: A GridLab Overview. International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications, August 2003.