

Benchmarking the OGSA-DAI Middleware

*William Hoarau¹ and Sébastien Tixeuil¹
Nuno Rodrigues², Décio Sousa², and Luis Silva²*

*¹LRI - CNRS UMR 8623 & INRIA Grand Large,
Universit Paris Sud XI, France
Email: {hoarau,tixeuil}@lri.fr*

*²Departamento Engenharia Informtica, Univ. Coimbra,
Polo II, 3030-Coimbra, Portugal
Email: luis@dei.uc.pt*



CoreGRID Technical Report
Number TR-0060
October 5, 2006

Institute on System Architecture

CoreGRID - Network of Excellence
URL: <http://www.coregrid.net>

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Benchmarking the OGSA-DAI Middleware

William Hoarau¹ and Sébastien Tixeuil¹
Nuno Rodrigues², Décio Sousa², and Luis Silva²

¹LRI - CNRS UMR 8623 & INRIA Grand Large,
Universit Paris Sud XI, France
Email: {hoarau,tixeuil}@lri.fr

²Departamento Engenharia Informtica, Univ. Coimbra,
Polo II, 3030-Coimbra, Portugal
Email: luis@dei.uc.pt

CoreGRID TR-0060

October 5, 2006

Abstract

One important contribution to the community that is developing Grid middleware is the definition and implementation of benchmarks and tools to assess the performance and dependability of Grid applications and the corresponding middleware. In this paper, we present an experimental study that was conducted with OGSA-DAI, a popular package of middleware that provides access to remote data resources through a unified Web-service front-end. The results show that OGSA-DAI is quite stable and performed quite well in scalability tests, executed on Grid5000. However, we also demonstrate that OGSA-DAI WSI is currently using a SOAP container (Apache Axis1.2.1) that suffers from severe memory leaks. We show how the default configuration of OGSA-DAI is not affected by that problem, but a small change in the configuration of a Web-service may lead to very unreliable execution of OGSA-DAI.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1 Introduction

Grid middleware and grid services are still in its infancy. Until they reach the maturity curve it is important to devise some benchmarks and tools to assess the performance, dependability and security level. These benchmarks and tools should be easily applied to different grid services implementations that follow standard protocols. They should also present some convincing results to the scientists and developers that are working with Grid middleware that was developed in R&D projects. One of the most popular Grid middleware packages is OGSA-DAI, a package that allows the remote access to data-resources (files, relational and XML databases) through a standard front-end based on Web-services specification. In this paper we present an experimental study of the performance and dependability level of OGSA-DAI, by making use of a benchmarking tool, called QUAKE, and running the experiments in the Grid5000 infrastructure. The experiments allowed us to collect some interesting results to the community that is using OGSA-DAI, the team of developers behind it and to the researchers that are studying the dependability metrics of Grid middleware, as is the case of our two groups: INRIA Grand-Large and the University of Coimbra.

2 Related Works

The idea of dependability benchmarking is now a hot-topic of research and there are already several publications in the literature. The components of a dependability benchmark have been defined in [1].

In [2] is proposed a dependability benchmark for transactional systems (DBench-OLTP). Another dependability benchmark for transactional systems is proposed in [3]. In [20] the authors present a proposal for the classification of dependability in transactional database systems [4].

A dependability benchmark for operating systems was proposed by [5]. That benchmark was targeted for the study of the operating system robustness in the scenario of faulty applications. Another study about the behavior of the operating system in the presence of software faults in OS components was presented in [6].

The research presented in [7] addresses the impact of human errors in system dependability. In [8] is presented a methodology to evaluate human-assisted failure-recovery tools and processes in server systems. Another work was presented in [9] that focus on the availability benchmarking of computer systems. Research work at Sun Microsystems defined a high-level framework targeted to availability benchmarking [10].

At IBM, the Autonomic Computing initiative is also developing benchmarks to quantify the autonomic capability of a system [11]. In that paper they have discussed the requirements of benchmarks to assess the self-* properties of a system and they proposed a set of metrics for evaluation. In [12] is presented a further discussion about benchmarking the autonomic capabilities of a system. In [13] is presented an approach to conduct benchmarking of the configuration complexity. A benchmark for assessing the self-healing capabilities of a potential autonomic system was presented in [14].

In [15] the authors present a dependability benchmark for Web-Servers. This tool used the experimental setup, the workload and the performance measures specified in the SPECWeb99 performance benchmark.

The dependability benchmark tool that is presented in this paper is targeted to Grid and Web-Services. It has been used to assess the dependability level of SOAP-servers [16], Web-service specifications [17] and tools of Grid middleware.

In this paper, we will present a benchmarking study that was conducted with OGSA-DAI middleware.

3 OGSA-DAI Overview

OGSA-DAI [18] is a middleware platform that allows data resources, such as relational or XML databases, to be accessed as Web-services. The software includes a collection of components for querying, transforming and delivering data in different ways, and a simple toolkit for developing client applications. In a short sentence, OGSA-DAI provides a way for users to Grid-enable their data resources.

The diagram in Figure 1 represents the global architecture of the OGSA-DAI platform. The data layer represents the data resources that can be exposed via OGSA-DAI. Data resources can be relational databases, XML databases and files. The requests to OGSA-DAI web services have a uniform format irrespective of the data resource exposed by the service.

The business logic layer represents the core of OGSA-DAI and consists of components known as data service resources which are directly connected to the data resources. Each data service resource is related with one data re-

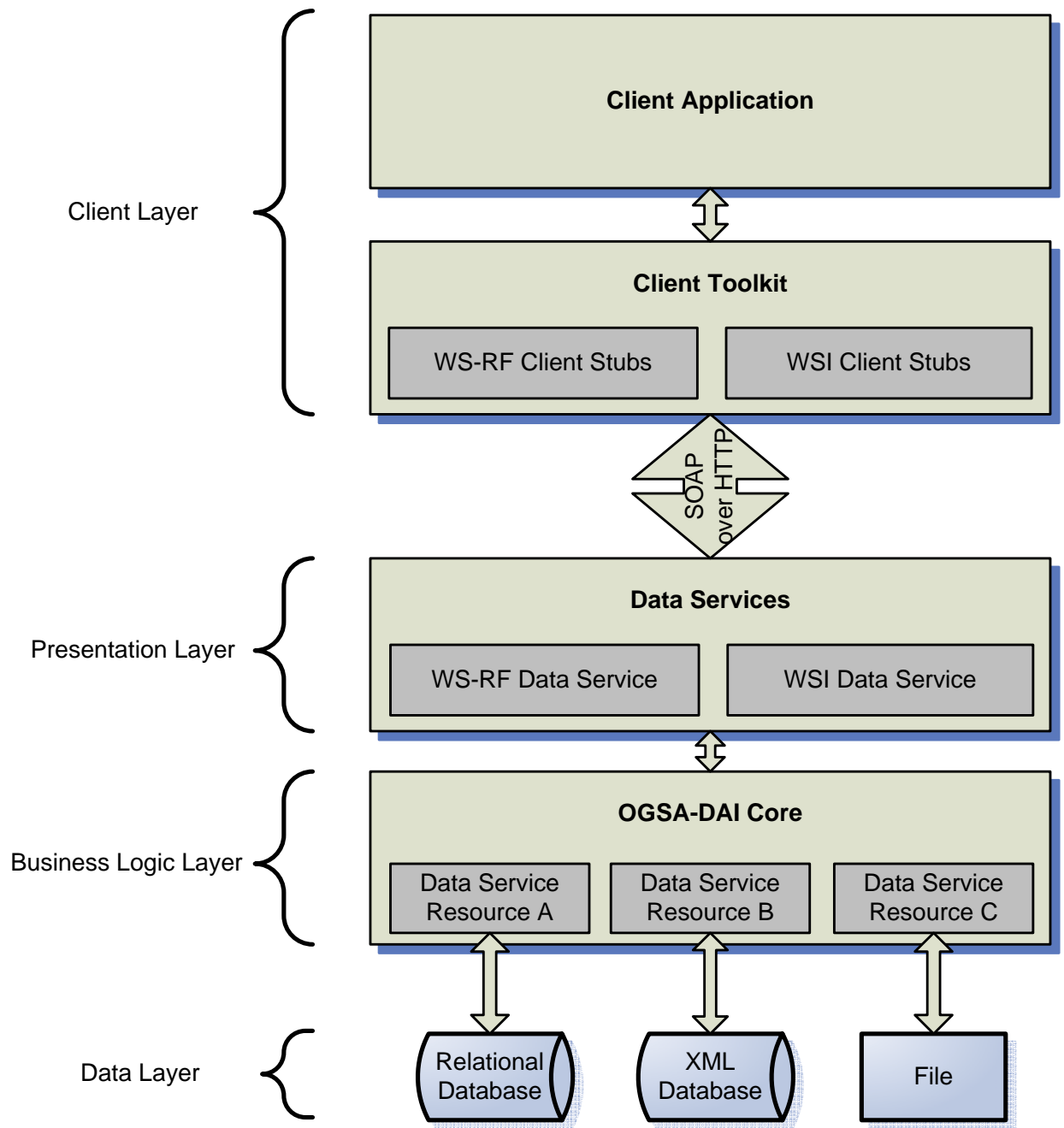


Figure 1: OGSA-DAI Architecture

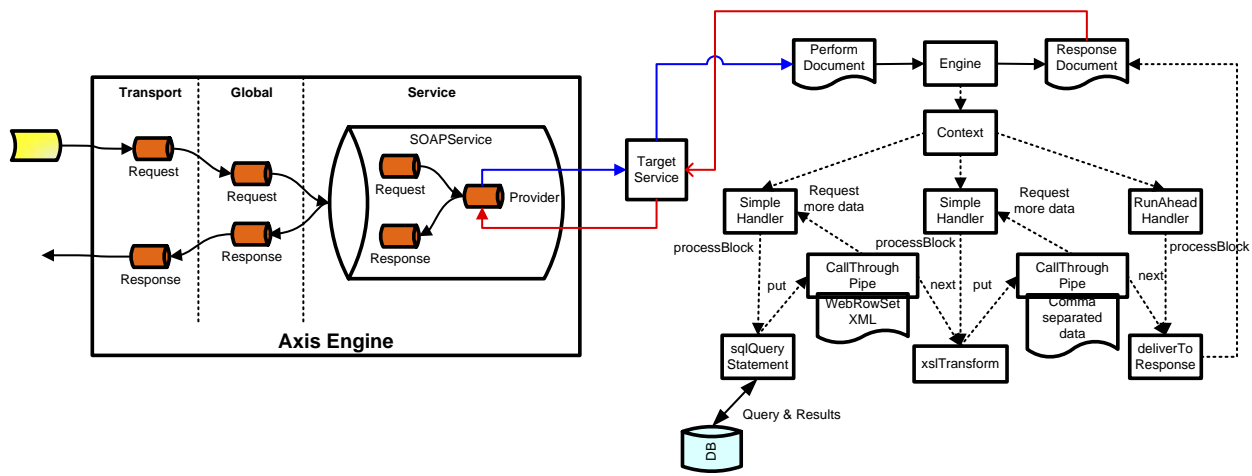


Figure 2: OGSA-DAI Internals

source. The data service resources are responsible for session management, processing the documents which describe the actions that a data service resource should take on behalf of the client, generate response documents that contain the results from a database query, conduct the streaming data in and out of data service resources to and from the clients.

The presentation layer encapsulates the functionality required to expose data service resources using web service interfaces. The Web-service front-end can run over two types of specifications: Web Services Interoperability (WSI) [19] and Web Services Resource Framework (WSRF) [20]. The WSI version runs over Jakarta Tomcat [21] and Axis [22] while the WSRF version runs with the Globus Toolkit [23]. The client layer permits that every client can interact, in a transparent way, with a data resource via a corresponding Web-service.

The front-end of OGSA-DAI is a set of Web-services that in the case of WSI require a SOAP container to handle the incoming requests and translate them to the internal OGSA-DAI engine. Figure 2 presents the main internal modules of OGSA-DAI implementation. While the detailed description of the OGSA-DAI internal is out-of-scope of this paper (more information can be found in [18]) the interesting aspect to take into account is the Web-service that handles the transport layer that uses SOAP messages. Our benchmarking tool can be easily applied to OGSA-DAI since it makes use of standard Web-service specifications.

At the moment OGSA-DAI middleware is used in several important Grid projects [24], including: AstroGrid, BIoDA, Biogrid, BioSimGrid, Bridges, caGrid, COBrA-CT, Data Mining Grid, D-Grid, eDiaMoND, ePCRn, ESSE, FirstDIG, GEDDM, GeneGrid, GEODE, GEON, GridMiner, InteliGrid, INWA, ISPIDER, IU-RGRbench, LEAD, MCS, myGrid, N2Grid, OntoGrid, ODD-Genes, OGSA-WebDB, Provenance, SIMDAT, Secure Data Grid, SPIDR, UNIDART and VOTES.

This list is clear representative of the importance of OGSA-DAI and the relevance of this particular benchmarking study.

4 QUAKE: A Benchmarking Tool for Grid Services

QUAKE [16] is a dependability benchmark tool to assess the performance and dependability level of Grid and Web-Services. It is composed by a set of software components, as presented in Figure 3.

The main components are the Benchmark Management System (BMS) and the System-Under-Test (SUT). The SUT consists of a SOAP-server running some Web/Grid service. The application under test is not limited to a SOAP-based application: in fact, the benchmark infrastructure can also be used with other examples of client-server applications that use other different middleware technologies.

There are several client machines that invoke requests in the server using SOAP requests. The Benchmark Management System (BMS) is a collection of software tools that allows the automatic execution of the benchmark. It includes a module for the definition of the benchmark, a set of procedures and rules, definition of the workload that will be

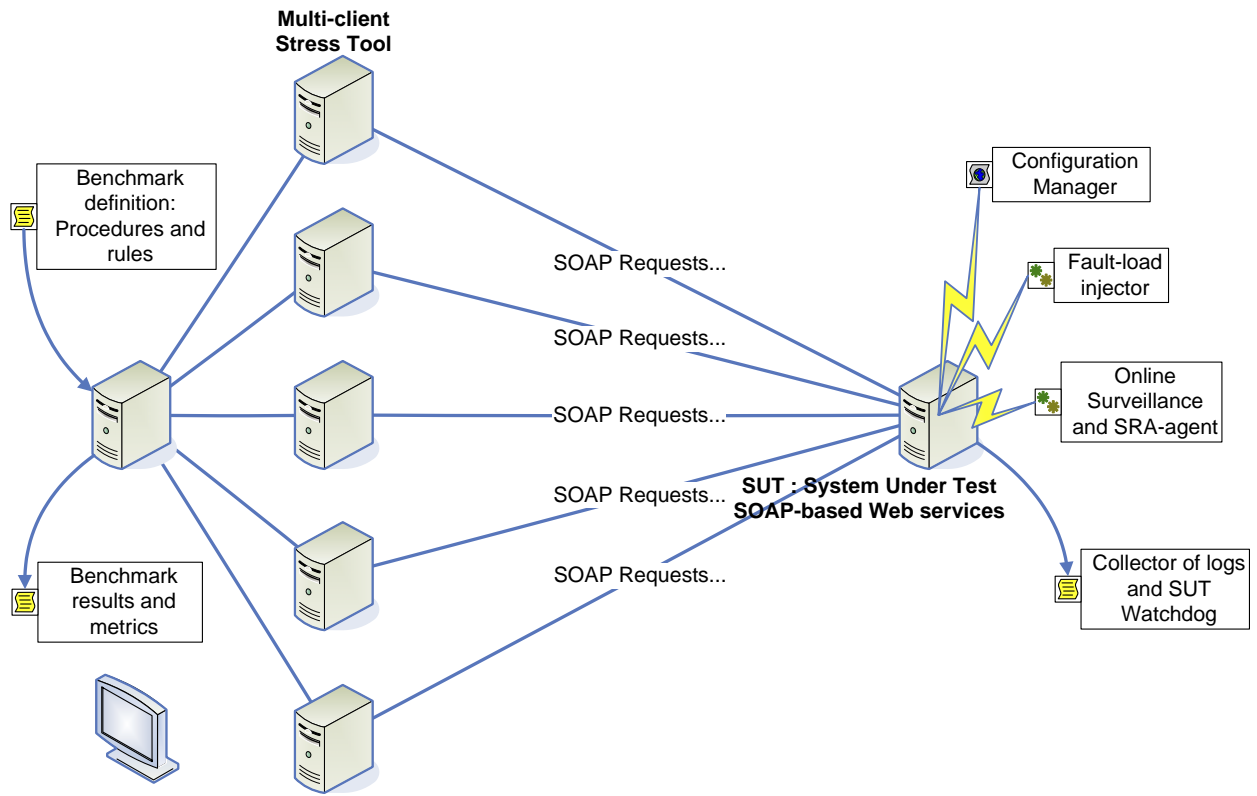


Figure 3: QUAKE Architecture

produced in the SUT, a module that collects all the benchmark results and produces some results that are expressed as a set of dependability metrics. The BMS system may activate a set of clients (running in separate machines) that inject the defined workload in the SUT by making requests to the Web Service. The execution of the client machines is timely synchronized and all the partial results collected by each individual client are merged into a global set of results that generated the final assessment of the dependability metrics. The BMS system includes a reporting tool that presents the final results in a readable and graphic format.

The results generated by each benchmark run are expressed as throughput-over-time, the total turnaround time of the execution, the average latency, the functionality of the services, the occurrence of failures, the characterization of those failures (crash, hang, zombie-server), the correctness of the final results and the failure scenarios that are observed at the client machines (explicit error messages or timeouts).

QUAKE includes a list of different workloads that can be chosen at the beginning of each experiment: poisson, normal, burst and spike. In this study, we just used the bust distribution which generates the maximum workload.

From the side of the SUT system, there are four modules that also make part of the QUAKE benchmark tool: a stress-load injector, a configuration manager, a collector of logs with the benchmark results and a watchdog of the SUT system. The stress-load injector is an optional module in this framework. It is introduced by an external program that runs in the same server (SUT) and run for the same system resources, consuming one or several of the following operating systems resources: (a) Consumption of memory using a ramp-up distribution; (b) Creation of a large number of threads; (c) Extensive usage of file-descriptors; (d) Consumption of database connections. The configuration manager helps in the definition of the configuration parameters of the SUT middleware. It is absolutely true that the configuration parameters may have a considerable impact in the robustness of the SUT system. By changing those parameters in different runs of the benchmark it allow us to assess the impact of those parameters in the results expressed as dependability metrics. The SUT system should also be installed with a module to collect raw data from the benchmark execution. This log data will be then sent to the BMS server that will merge and compare with the data collected from the client machines. The final module is a SUT-Watchdog that detects when a SUT system crashes or hangs when the benchmark is executing. When a crash or hang is detected the watchdog generates a restart of the SUT system and associated applications, thereby allowing an automatic execution of the benchmark runs without user intervention.

There are some other tools in the market that can be used for performance benchmarking of SOAP Web-services, namely: SOAtest [25] and TestMaker [26]. However those tools are mainly targeted for testing the functionality of the SOAP applications and to collect some performance figures.

QUAKE has some fundamental differences: it is targeted to study the dependability attributes, it includes a different approach for the workload distributions, a stress-load module to affect the resources of the system-under-test and it is mostly targeted to the collection of dependability metrics.

5 Experimental Results

In this section, we present a benchmarking study we have conducted with OGSA-DAI by using the QUAKE tool in a large-scale grid infrastructure: the Grid5000.

Grid5000 is an experimental platform dedicated to computer science for the study of grid algorithms, and partly founded by the French incentive action "ACI Grid". Grid5000 consists of 14 clusters located in 9 French cities with 40 to 450 processors each, with a total of 1928 processors. Most of the tests were executed on Grid Explorer which is a major component of the Grid5000 platform. It is located at Orsay and consists in 312 "IBM eServer 326m" computers. All computers are dual-processors AMD Opterons running at 2.0 GHz with 2 GB of RAM, and each computer has a 80 GB IDE hard drive and a GigaEthernet network interface card. For the scalability experiments, 3 clusters of Grid5000 were used. We used 200 machines of the previously mentioned Orsay's cluster, 50 machines of the Sophia's cluster and 50 machines of the Lille's cluster. Using this configuration, we were able to scale up to 300 clients during our experiments. A major feature of the Grid5000 project is the ability for the user to deploy its own environment (including operating system kernel and distribution) over all the nodes that are dedicated to the experiment. We deployed a Debian Linux operating system, with a kernel 2.6.13-5, including Java 1.5.0, Tomcat 5.0.28, Axis 1.2.1 and OGSA-DAI WSI 2.2.

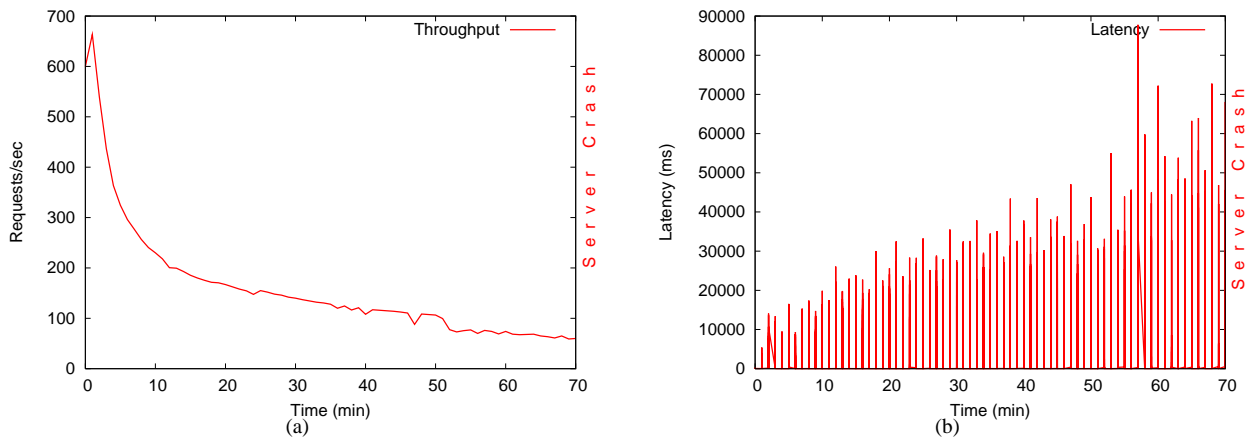


Figure 4: Results with Tomcat/Axis

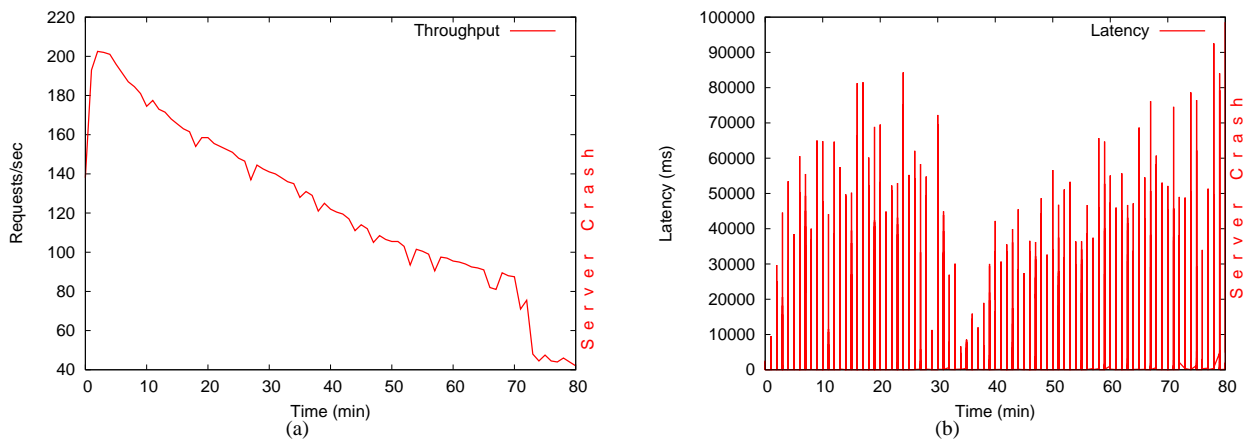


Figure 5: Second experiments with Tomcat/Axis (database synthetic application)

5.1 Benchmarking Tomcat+Axis

As OGSA-DAI is a middleware on top of Tomcat and Axis, the first study was to measure the dependability level of a simple Web-service application deployed with Tomcat/Axis running in a large-scale cluster of Grid5000. The test was executed with 40 clients and 100,000 requests per client. We used a synthetic application that executes a very simple computation. The versions of the WSI container were: Tomcat 5.0.28 and Axis 1.2.1. The results are presented in Figure 4.

In Figure 4(a) we can see that the throughput is decreasing over time until a certain point where the server crashes: at this point only 665,078 requests were executed, instead of the 4,000,000 that were expected. In Figure 4(b) we can see the latency increasing over time until the point where there was a crash (approximately 70.5 minutes of test running).

This first study was conducted with a very synthetic application. Then, we decide implement a Web-Service on Axis that was working as a front-end to access a database: it receives a SQL query and returns the resulting rows. The results are also taken with 40 clients and are presented in Figure 5. We can see the results are quite similar to the first experiment. The throughput decreases over time until the server has a crash after 80 minutes of execution.

This was the same behavior we have observed in [16]. In that paper we presented a detailed study of the reliability of SOAP and we have demonstrated that Axis1.3 suffers from severe memory leaks. Some sort of corrective

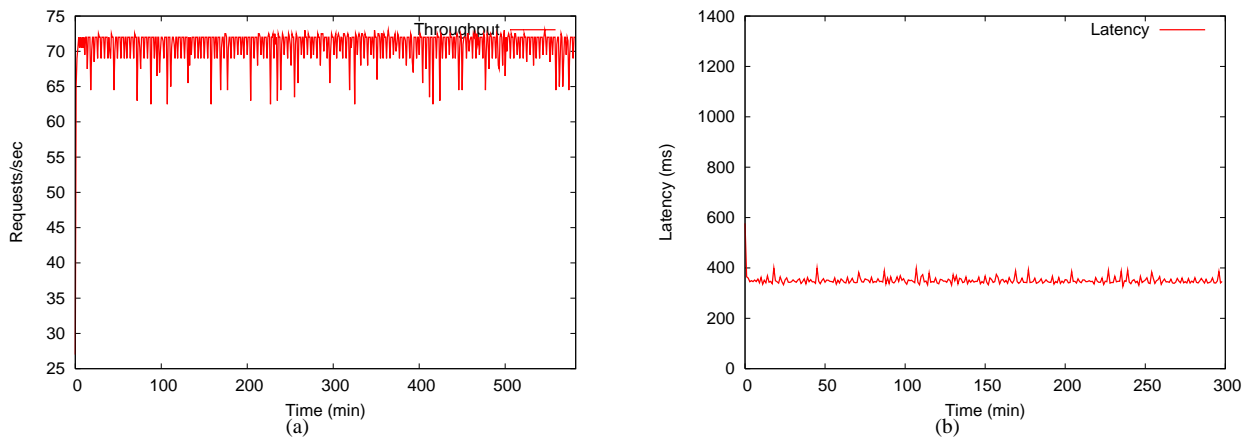


Figure 6: Results with OGSA-DAI (WSI)

mechanism is mandatory to avoid the failure of the applications that are using that SOAP implementation.

It was that particular study that drove our curiosity and our concern to assess the dependability level of OGSA-DAI: if Axis 1.2 and 1.3 suffer from severe memory leaks and OGSA-DAI makes use of Axis 1.2.1 what would be resulting reliability of the applications that make use of OGSA-DAI data grid services?

Results are presented in the next subsection.

5.2 Benchmarking OGSA-DAI

Since OGSA-DAI uses the Axis1.2.1 platform to deploy its own Web-services it is legitimate for one to assume that OGSA-DAI should be prone to the severe effects of the memory leaks from Axis. To find evidences, we conducted several tests to the OGSA-DAI platform in a large-scale cluster of Grid5000.

The first experiment was conducted with 25 nodes, each one executing 100,000 requests. Figure 6 presents the observed throughput and latency.

As can be clearly seen the performance of OGSA-DAI remained quite stable during all the experiment that was conducted with a maximum burst distribution. The throughput was fairly stable with an average value of 71.43 requests/sec. The latency had an average value of 349.1 ms.

We repeated this benchmark but increasing the number of clients to see the impact of scale on the OGSA-DAI middleware. Different experiments were executed with 25, 100, 200 and 300 clients. The results are presented in Figure 7.

We can see that the throughput is the same for every experiment and the latency increases linearly. The server can perfectly handle 300 simultaneous clients requesting accesses to data-services using a burst distribution.

It is clear that OGSA-DAI has some congestion-flow control mechanism that sets up the throughput to a fixed value despite the number of simultaneous clients. Although this could explain the impact of scale in the server, it does not explain why the memory leaks present in Axis do not manifest in the OGSA-DAI environment.

We then started to look to some of the OGSA-DAI configuration parameters, and in particular to the main parameters that establish some congestion flow mechanism:

- **maximum simultaneous request:** define how many requests can be processed simultaneously;
- **request queue length:** this is the queue where the request are stored before being processed.

For all the previous tests the maximum number of simultaneous requests was the same than the number of clients: for example, with 100 clients, the maximum simultaneous request was set up to 100. The queue length was always 20.

We ran the scalability test again but with a value of half to the maximum simultaneous requests and a larger queue to see the impact of these parameters on performance. The results we got were basically the ones presented in Figure 6. So, apparently, these parameters have no influence on the performance, at least at this scale.

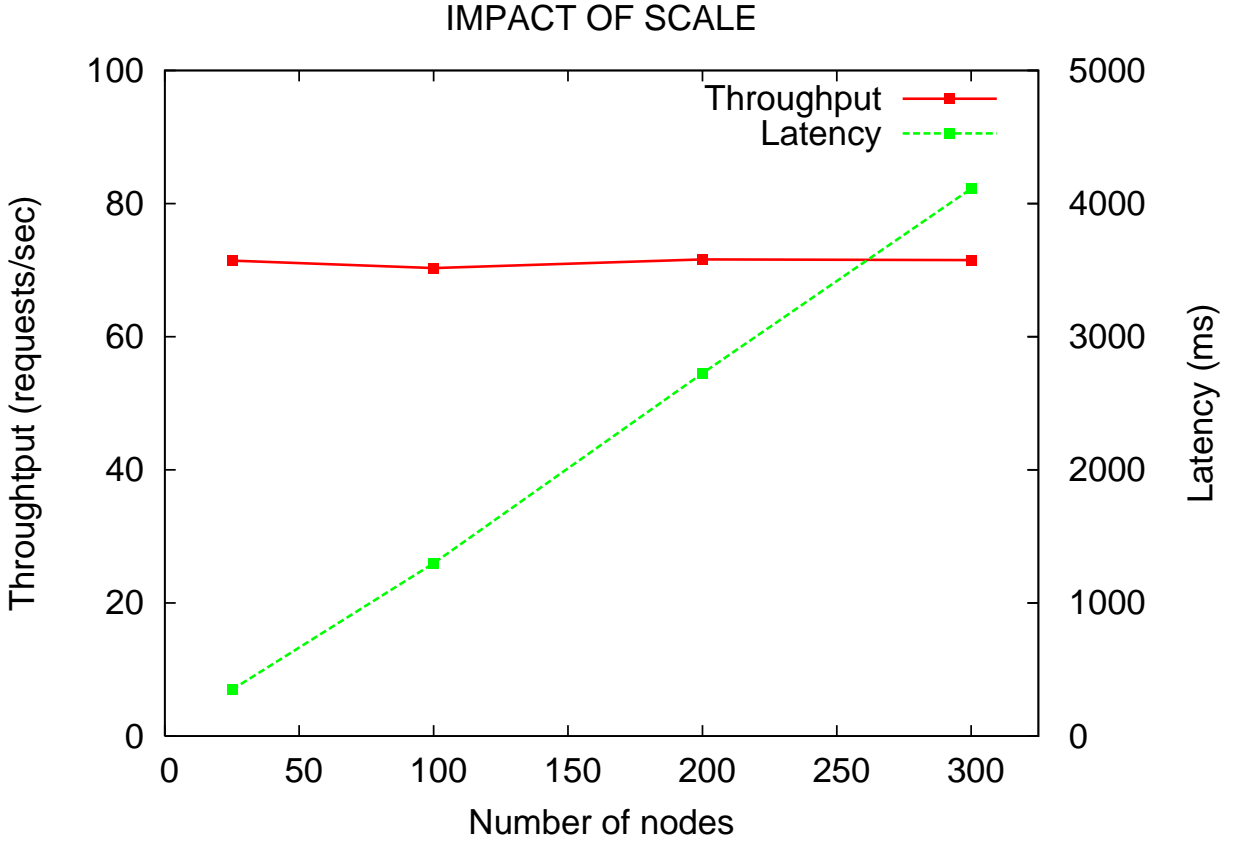


Figure 7: Impact of scale on OGSA-DAI (WSI)

IMPACT OF TRESHOLD

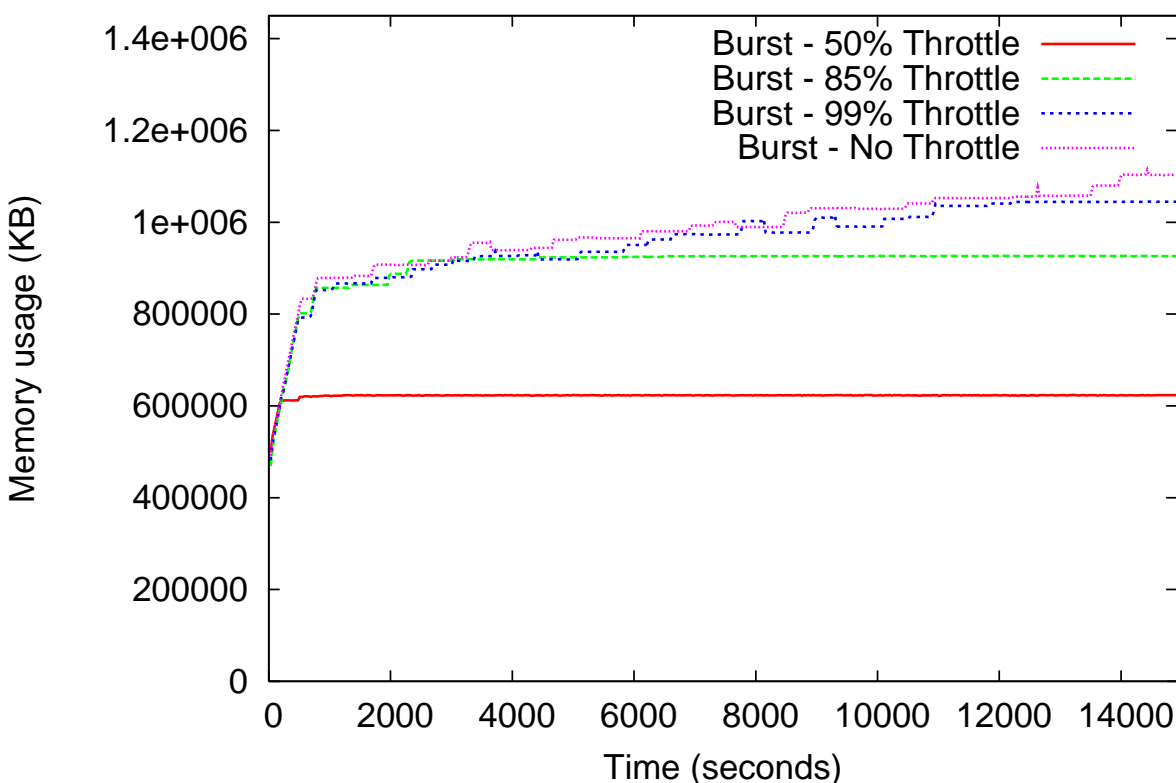


Figure 8: Impact of the threshold-value on memory consumption (OGSA-DAI)

After this first set of results, we still had no answers to our fundamental question: how OGSA-DAI can be so stable if it uses Axis1.2.1 that suffers from memory leaks?

We conducted some further experiments and some code-inspections to the OGSA-DAI implementation, trying to understand its inner details. One thing that recalled our attention was a method that is called before executing an operation that consumes memory. This method calculates the memory needed for that operation to take place, and then checks if the total consumed memory is above a predefined threshold. If it is not, then the operation can be executed. Otherwise the middleware makes a set of explicit calls to the JVM garbage collector to free up some memory.

The default threshold for memory consumption is 85% of the JVM Heap, so all tests we ran previously used this value. We conducted some other tests by changing the threshold value: 50%, 85%, 99% and with the mechanism turned off. The idea was to see if this was the main mechanism responsible for the stability of OGSA-DAI server. The results are presented in Figure 8.

This mechanism forces a more frequent usage of the garbage collection mechanism but it is not a mechanism *per se* that provides the robustness to the OGSA-DAI middleware. When we turned off this mechanism we saw some small instability in the memory consumption but the server never crashed.

Finally, there was one last thing that caught our attention: the fact that the Web-service deployed with OGSA-DAI was using a scope set to application.

This means the WS application is only instantiated once being then shared with every request it receives and by every different client.

This is not the usual way of deploying Web-Services unless the Web-service is completely stateless or provides global data that should be shared by all the clients.

It is quite usual that a Web-service application needs to store information about a particular session so it can correlate future requests to previous ones made by the same user. These kinds of applications often use scope set to

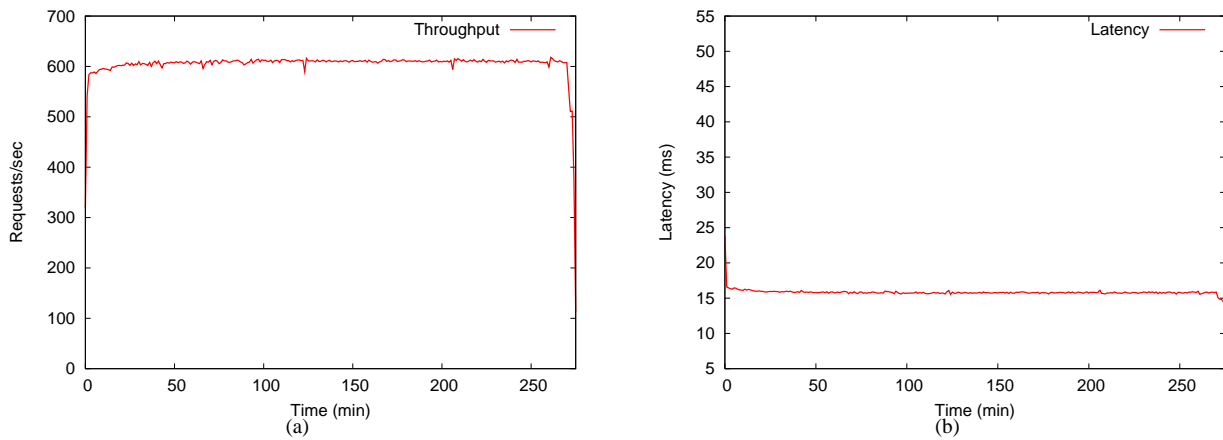


Figure 9: Experiment with Tomcat/Axis (application scope Web-service)

session so that they can manage the information about the clients in a session-based model.

As stated in section 3, OGSA-DAI makes use of Web-Services as a front-end layer to the data resources. It really does not need to correlate different requests and the Web-service is completely stateless; it just needs to create an interface which will be the same to every request despite the client where it comes from. This clearly justifies the use of the application scope in the deployed Web-Services.

But this raises the question: could this be the answer to the stable behavior of OGSA-DAI?

To understand the impact of the scope of the Web-services in the robustness of the applications we conducted two different tests:

1. one test with a WS deployed over Axis using an application scope;
2. another test using OGSA-DAI but the scope was changed to session.

In both tests we used the QUAKE tool, with 10 simultaneous clients that were programmed to make 10 million requests into the server in the overall.

Figure 9 shows the results of the synthetic application running on Axis1.2.1 with an application scope Web-service. Figure 10 represent the results of the OGSA-DAI with a session equal to scope.

In Figure 9, we can see that Axis1.2.1 does not suffer from memory leaks if the Web-service is deployed with scope set to application. On the contrary, we can see in Figure 10 that OGSA-DAI runs very unstable if the scope of the Web-service is set to session: the application with OGSA-DAI crashes after 42.5 minutes of test execution and was only able to fulfill 2020 requests. The throughput was very unstable and reached very low values. These results are quite interesting and finally explain why the default configuration of OGSA-DAI was so stable.

The reader should compare the results from Figure 10 with Figures 6 and 7: in these two Figures we could see that OGSA-DAI was very stable and provided a sustained throughput of 70 req/sec, even when we increased the number of clients to 300. In Figure 10 we can see that the throughput is very unstable and very low in comparison (average value of 0.78 reqs/sec). Similar observations to the latency of requests: in Figure 7 the results taken with 300 clients have shown an average latency of 4000 msec. In Figure 10 we can see an average latency of 11.842 msec (almost 12 seconds) with only 10 simultaneous clients.

All these results are due to the fact the default configuration of OGSA-DAI sets the scope of the Web-service to application. In this case OGSA-DAI does not trigger the memory leaks of Axis1.2.1. If the scope is set to session, the OGSA-DAI will trigger the severe memory leaks of Axis1.2.1 and the resulting reliability will be a major point of concern.

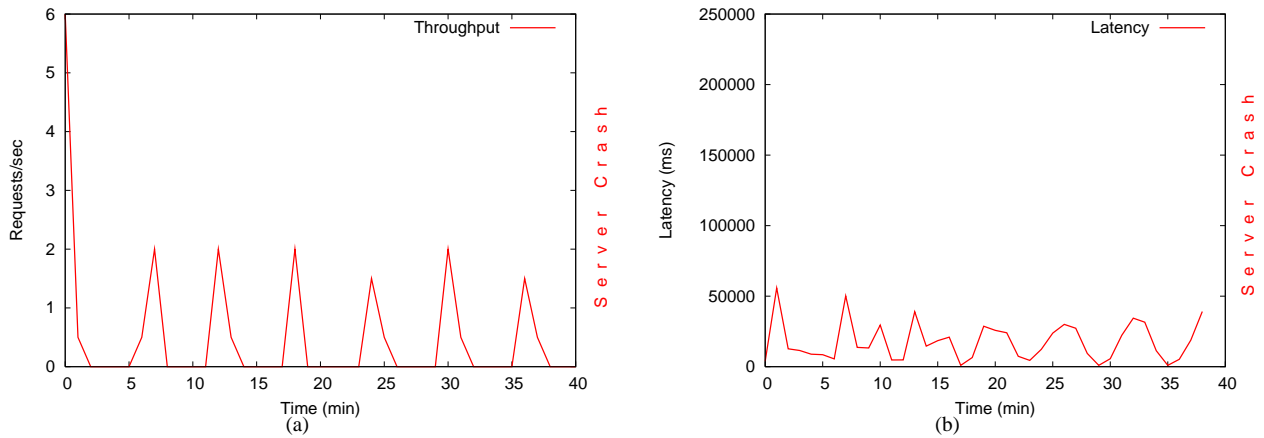


Figure 10: Experiment with OGSA-DAI (session scope Web-service)

6 Conclusions

In this paper we have presented the first results from a benchmarking study of OGSA-DAI. The good news from these initial results are the fact that OGSA-DAI is quite stable and performs considerably well in scenarios of scalability and workload testing, as was the case of that experiment with 300 clients in the Grid5000. These are good news for all those Grid Projects that are making use of OGSA-DAI and it is a very positive acknowledgement of the good work that has been doing by the OGSA-DAI team.

On the other hand, we have proved that the stability of OGSA-DAI is due to the fact it makes of application scope Web-services, which fortunately do not trigger the memory leaks of Axis1.2.1. If by any reason, a programmer needs to implement sessions for the different clients and have to change the scope of the Web-service to session, then there will be a critical problem in terms of reliability of the OGSA-DAI middleware.

This is an interesting lesson that should be taken into account by the team that is developing OGSA-DAI middleware.

In this paper, we have mainly proved that having a dependability benchmarking tool for Web-services and grid-enabled application is crucial for the community. By using our QUAKE tool we were able to detect and understand the reasons for the stability of OGSA-DAI but also the potential leaks that may turn it a very unreliable middleware package, if it triggers the memory leaks of the underlying Axis implementation.

Good news for the community is the fact that Axis 2 has already been released and this new version solves most of the problems of the previous versions.

However, there are still a lot of software packages (like OGSA-DAI) that make use of the Axis1.2 or 1.3 versions and thereby may potentially suffer from serious memory leaks if they use session scope Web-services.

In the future we plan to continue our work on dependability benchmarking for Web-services and we are studying the potential integration of QUAKE with FAIL-FCI [27].

Acknowledgements

This research work is carried out in part under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). We would like to thank the OGSA-DAI team for their collaboration in this experimental study.

References

- [1] P.Koopman, H.Madeira. "Dependability Benchmarking & Prediction: A Grand Challenge Technology Problem", Proc. 1st IEEE Int. Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems; Phoenix,

- [2] M. Vieira and H. Madeira, "A Dependability Benchmark for OLTP Application Environments", Proc. 29th Int. Conf. on Very Large Data Bases (VLDB-03), Berlin, 2003.
- [3] K. Buchacker and O. Tschaeché, "TPC Benchmark-c version 5.2 Dependability Benchmark Extensions", <http://www.fau-machine.org/papers/tpcc-depend.pdf>, 2003
- [4] D. Wilson, B. Murphy and L. Spainhower. "Progress on Defining Standardized Classes for Comparing the Dependability of Computer Systems", Proc. DSN 2002, Workshop on Dependability Benchmarking, Washington, D.C., USA, 2002.
- [5] A. Kalakech, K. Kanoun, Y. Crouzet and A. Arlat. "Benchmarking the Dependability of Windows NT, 2000 and XP", Proc. Int. Conf. on Dependable Systems and Networks (DSN 2004), Florence, Italy, 2004.
- [6] J. Duraes, H. Madeira, "Characterization of Operating Systems Behaviour in the Presence of Faulty Drivers Through Software Fault Emulation", in Proc. 2002 Pacific Rim Int. Symposium Dependable Computing (PRDC-2002), pp. 201-209, Tsukuba, Japan, 2002.
- [7] A. Brown, L. Chung, and D. Patterson. "Including the Human Factor in Dependability Benchmarks", Proc. of the 2002 DSN Workshop on Dependability Benchmarking, Washington, D.C., June 2002.
- [8] A. Brown, L. Chung, W. Kakes, C. Ling, D. A. Patterson, "Dependability Benchmarking of Human-Assisted Recovery Processes", Dependable Computing and Communications, DSN 2004, Florence, Italy, June, 2004
- [9] A. Brown and D. Patterson, "Towards Availability Benchmarks: A Case Study of Software RAID Systems", Proc. 2000 USENIX Annual Technical Conference, San Diego, June 2000
- [10] J. Zhu, J. Mauro, I. Pramanick. "R3 - A Framework for Availability Benchmarking", Proc. Int. Conf. on Dependable Systems and Networks (DSN 2003), USA, 2003.
- [11] S. Lightstone, J. Hellerstein, W. Tetzlaff, P. Janson, E. Lassetre, C. Norton, B. Rajaraman and L. Spainhower. "Towards Benchmarking Autonomic Computing Maturity", 1st IEEE Conf. on Industrial Automatics (INDIN-2003), Canada, August 2003.
- [12] A. Brown, J. Hellerstein, M. Hogstrom, T. Lau, S. Lightstone, P. Shum, M. P. Yost, "Benchmarking Autonomic Capabilities: Promises and Pitfalls", Proc. Int. Conf. on Autonomic Computing (ICAC'04), 2004
- [13] A. Brown and J. Hellerstein, "An Approach to Benchmarking Configuration Complexity", Proc. of the 11th ACM SIGOPS European Workshop, Leuven, Belgium, September 2004
- [14] A. Brown, C. Redlin. "Measuring the Effectiveness of Self-Healing Autonomic Systems", Proc. 2nd Int. Conf. on Autonomic Computing (ICAC'05), 2005
- [15] J. Dures, M. Vieira and H. Madeira. "Dependability Benchmarking of Web-Servers", Proc. 23rd International Conference, SAFECOMP 2004, Potsdam, Germany, September 2004. Lecture Notes in Computer Science, Volume 3219/2004
- [16] L. Silva, H. Madeira and J. G. Silva "Software Aging and Rejuvenation in a SOAP-based Server", IEEE-NCA: Network Computing and Applications, Cambridge USA, July 2006
- [17] D. Sousa, N. Rodrigues and L. Silva "How Reliable is WS-Reliable Messaging: An Experimental Study with Apache Sandesha", Submitted for publication, 2006.
- [18] OGSA-DAI: <http://www.ogsadai.org.uk/>
- [19] Web Services Interoperability (WS-I) <http://www.ws-i.org>
- [20] WSRF, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
- [21] Jakarta Tomcat: <http://jakarta.apache.org/tomcat>

- [22] Apache Axis : <http://ws.apache.org/axis/>
- [23] Globus toolkit: <http://www.globus.org/toolkit/>
- [24] Projects that use OGSA-DAI: <http://www.ogsadai.org.uk/about/projects.php>
- [25] Parasoft SOAtest: <http://www.parasoft.com>
- [26] PushToTest TestMaker: <http://www.pushtotest.com>
- [27] William Hoarau, Sbastien Tixeuil, and Fabien Vauchelles. "Easy fault injection and stress testing with FAIL-FCP". Technical Report 1421, Laboratoire de Recherche en Informatique, Universit Paris Sud, October 2005