

Dependable Grid Services: A Case Study with OGSA-DAI

Javier Alonso, Jordi Torres

`{alonso, torres}@ac.upc.edu`

*Technical Univ. of Catalonia - Barcelona Supercomputing Center
Barcelona - Spain*

Luis Moura Silva

`{luis}@dei.uc.pt`

*CISUC - Univ. of Coimbra
Coimbra - Portugal*



CoreGRID Technical Report
Number TR-0148
July 23, 2008

Institute on Architectural issues: scalability, dependability,
adaptability

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

Dependable Grid Services: A Case Study with OGSA-DAI

Javier Alonso, Jordi Torres
{alonso, torres}@ac.upc.edu
Technical Univ. of Catalonia - Barcelona Supercomputing Center
Barcelona - Spain

Luis Moura Silva
{luis}@dei.uc.pt
CISUC - Univ. of Coimbra
Coimbra - Portugal

CoreGRID TR-0148

July 23, 2008

Abstract

Grid middleware usually makes use of several software modules that due to their complexity and development approach may have some latent bugs and leaks. These bugs can cause visible performance failures and undesired service crashes. To cope with this sort of transient failures we present a proactive software rejuvenation approach that exploits the use of virtualization middleware. To prove the effectiveness of our mechanism we decided to apply it to OGSA-DAI, a sound example of a middleware that has been widely used in several Grid-related projects. OGSA-DAI makes use of Tomcat/Axis as the SOAP container and Axis v1.2.1 suffers from memory leaks. When it is not configured properly these leaks will result in a crash of the OGSA-DAI server. In this paper, we explain the application of our rejuvenation scheme in this particular example and we show that it is easy to get a software-based approach to improve the availability of a Grid Service even when one of the underlying layers suffers from clear symptoms of software aging. Our ultimate goal is to give a contribution for the techniques and concepts that can be used to achieve dependable Grid services.

1 Introduction

Grid computing understands the usage of large scale and heterogeneous resources in geographically dispersed sites. The target applications for Grid are usually long-running. In those systems the overall MTBF (Mean-time-between-failures) can be even lower than the total execution of a single application. It is thus mandatory that Grid middleware should have some effective support for fault-tolerance and mechanisms for high-availability, otherwise Grids cannot be used.

Most of the failures that happen in nowadays systems are just transient failures that happen from time to time and can be potentially solved by some automatic action of fault-tolerance. One scenario that is also expectable in complex software systems, such as any Grid middleware, is the occurrence of software aging. Aging is usually observed as a progressive degradation through time, which can lead to system crashes or undesirable hang ups. This phenomena is particularly troublesome in long-running applications, which can be found in Grid environments. It has also been observed in telecommunication systems [1], web-servers [2], enterprise clusters [3], OLTP systems [4] and spacecraft systems [5].

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

The most natural procedure to combat software aging is the technique of software rejuvenation [6]. Two basic approaches for rejuvenation have been proposed in the literature: time-based and proactive rejuvenation. Time-based rejuvenation techniques are widely used today in some real production systems, such as web-servers [7]. Proactive rejuvenation techniques have been studied in [3, 4] [8, 9] and it is widely understood that this technique of rejuvenation provides better results than the previous one, resulting in higher availability and lower costs. In our previous work [17], we have presented a self-recovery technique that makes use of virtualization. The main goal of the mechanism is to avoid the occurrence of software aging by applying clean planned restarts and replicated execution of application-services in different virtual machines that will be running on top of the same physical server.

The main reasons for using virtualization middleware [10, 11, 12] in our solution is to offer the chance for server consolidation and to support the compatibility at the level of binary code: no re-compilation or dynamic re-linking is necessary to port legacy application from physical machines to virtualized machines (VMs).

In this paper, we want to show how useful is our mechanism if we want to apply it to Grid Services. To achieve this goal we present a case-study using a very well-known Grid middleware: OGSA-DAI. We present the default performance behavior and the improvement using our self-recovery mechanism.

The rest of the paper is organized as follows: Section 2 presents an overview about our case-study (OGSA-DAI). Section 3 presents some performance results taken in two different infrastructures. Section 4 describes our virtualized clustering approach to obtain dependable Grid Services. Section 5 presents some new results with the application of our technique into OGSA-DAI and the measured impact. Finally, Section 6 presents the conclusions of this paper.

2 An Overview about our case-study: OGSA-DAI

One of the most popular Grid middleware packages is OGSA-DAI [13], 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. 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 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. This SOAP container is Tomcat/Axis 1.2.1. The detailed description of the OGSA-DAI internal is out-of-scope of this paper. At the moment OGSA-DAI middleware is used in several important Grid projects [14], 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.

3 Performance Study

In this section we present some performance figures of OGSA-DAI (version WSI 2.2) taken with different configurations and workloads to have a better view of its performance.

3.1 Experiments in Grid'5000

In [16], the authors present a benchmarking study of OGSA-DAI. That study has been made with the use of QUAKE [9], a dependability benchmarking tool that was developed to evaluate the performability and dependability figures of Grid and Web-Services. That study was conducted on Grid'5000, an experimental platform dedicated to computer science for the study of Grid algorithms, and partially founded by the French incentive action "ACI Grid". Grid'5000 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 Grid'5000.

One of the experiments was executed with the default configuration of OGSA-DAI. It was conducted with 25 nodes, each one executing 100,000 requests. All nodes were dual-processors AMD Opterons running at 2.0GHz with 2GB of RAM, and each computer has a 80 GB IDE hard drive and a GigaEthernet network interface card. The server was running with 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.

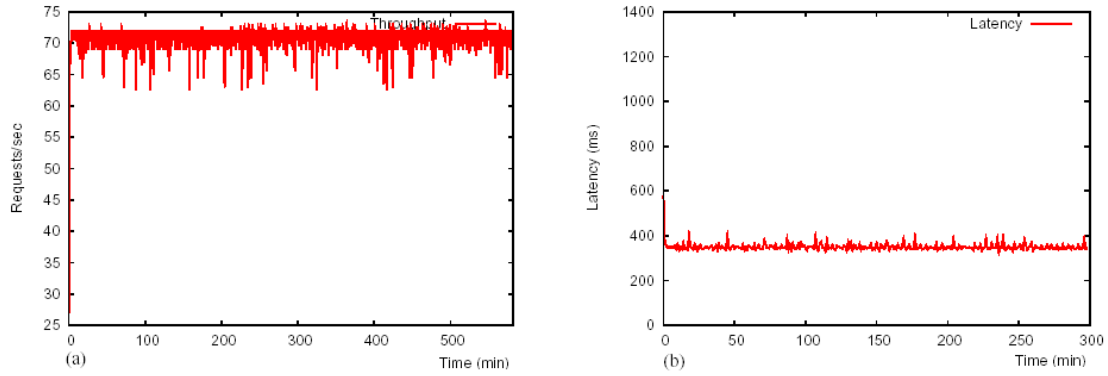


Figure 1: Results with OGSA-DAI (WSI) application scope

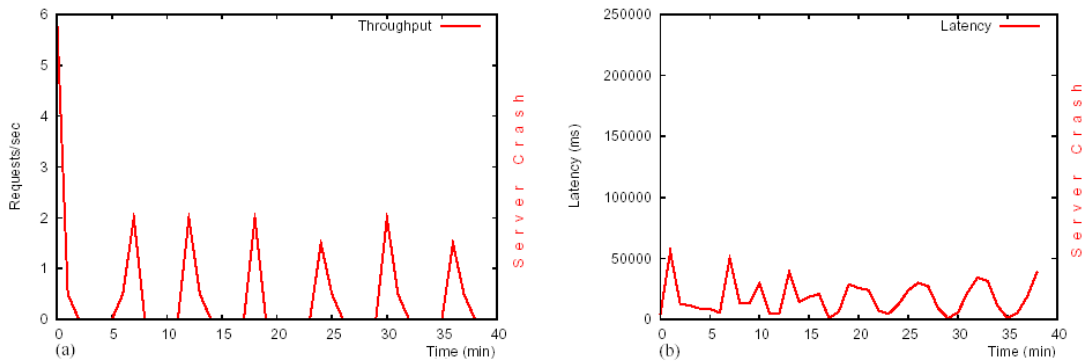


Figure 2: Results with OGSA-DAI (WSI) session scope

The results are presented in Figure 1. This experiment shows that the default configuration of OGSA-DAI presents a quite stable level of performance. In that configuration the OGSA-DAI Web-services are instantiated with application scope. This means the service is instantiated only once and is shared by every request of every different client. This is not the usual way of deploying Web-Services unless the service is completely stateless or provides global data that should be shared by all the clients. It is more common to instantiate the services as session scope. This way it is possible to avoid sharing between different users and it is possible to correlate different requests from the same user. However, if we configure OGSA-DAI with session scope it is very prone to the occurrence of software aging; not because there is any bug on OGSA-DAI, but because it makes use of Apache/Axis (v1.2) that is known to be an unreliable SOAP router due to the existence of internal memory leaks.

The results with session scope are presented in Figure 2. This result was taken with a burst workload and we can observe a very unstable level of performance leading to frequent suffered hangs up and crashes of the OGSA-DAI server. Right to be truth, this only happens when the web-services are configured with scope session. In application scope the internal leaks of Axis are not triggered and the problem is not spotted in the performance figures of OGSA-DAI.

3.2 Experiments with a Local Cluster

In this sub-section we present some similar results that were collected in a small cluster in the University of Coimbra. In these experiments we used a cluster of 5 machines: 3 running the client benchmark application, one for the Database Server (Tania) and another server for the Grid-services front-end (Katrina). All the machines were interconnected with a 100Mbps Ethernet switch. The detailed description of the machines is presented in Table 1.

In Figure 3 we present the latency of the OGSA-DAI server when applying different constant workloads of 1 request every 10, 20 and 30 seconds every client. In Figure 4 is presented the latency with a burst workload. It can

be seen that in both cases the latency is completely unstable. Figure 5 presents the throughput that is measured when applying a burst mode.

| | Katrina | | Tania | | Clients Machines | |
|-------------------------|-------------------------------------|----------------------------------|-----------------------------------|-----------------------|--------------------|------------|
| CPU | Dual Opteron (2000MHz) | AMD64 | Dual Core Opteron (1800MHz) | AMD64 165 | Intel (1000MHz) | Celeron |
| Memory | 4GB | | 2GB | | 512MB | |
| Hard Disk | 160GB (SATA2) | | 160GB (SATA2) | | | |
| Swap Space | 8GB | | 4GB | | 1024MB | |
| Operating Sys-tem | Linux 0.25-smp | 2.6.16.21- | Linux 0.25-smp | 2.6.16.21- | Linux netboot | 2.6.15-p3- |
| Java JDK | 1.5.0_06, Server VM | 64-bit | 1.5.0_06-b05 | Stan- dard Edition | | |
| Tomcat JVM heap size | 1024MB | | | | | |
| Other Software | Apache 5.0.28, and WSI 2.2 | Tomcat Axis 1.2.1 OGSA-DAI | MySQL 5.0.18 | | | |

Table 1: Detailed Machines Description

Figure 6 presents the memory usage of the Tomcat JVM. It can be seen that the server starts to make a fast use of memory that is not made free by the Java garbage collector. When the server starts to run in the memory limits it starts to lose requests, as presented in Figure 7. If we want to use OGSA-DAI in some real applications that would require a configuration of session scope in the web-services front-end then this behavior is totally unacceptable.

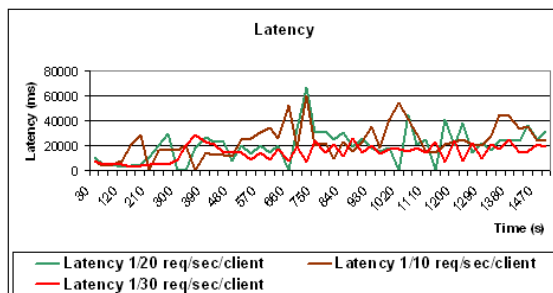


Figure 3: Latency with constant workloads

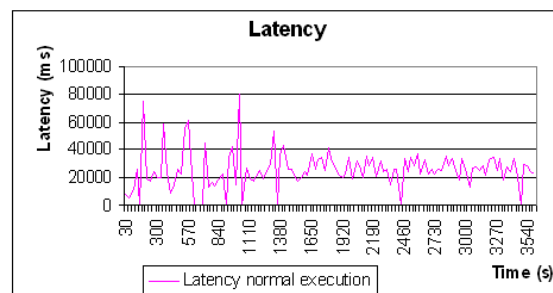


Figure 4: Latency with Burst workload

OGSA-DAI WSI 2.2 is using of an unreliable SOAP router: Axis v1.2. Since this SOAP module suffers from severe memory leaks it can clearly undermine the reliability level of any Grid deployment based on OGSA-DAI.

The best solution is definitely to re-engineer the OGSA-DAI WSI implementation by making a new version or by using Axis v2, a more reliable implementation of SOAP from the Apache group. For the time-being we decided to select the current WSI version of OGSA-DAI as a case-study. We applied our rejuvenation mechanism to this Grid middleware and we observed if it would work effectively without changing any piece of the Grid software. Next section we presented a summary description of our mechanism.

4 A Rejuvenation Scheme by using Virtualized Clustering

Our approach has been designed to use over any server or service. It is just necessary to install a virtualization layer and install some software modules. We have adopted XEN [15] virtualized middleware in our experiments, but we may have used any virtualization middleware. On top of our virtualization layer we create 3 virtual-machines: one

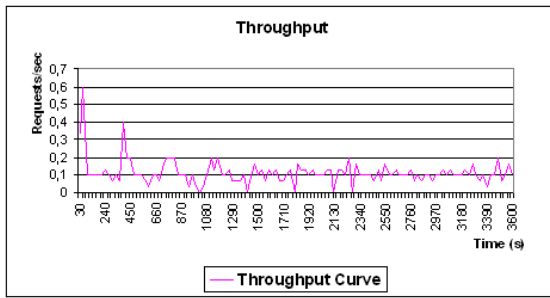


Figure 5: Throughput performance

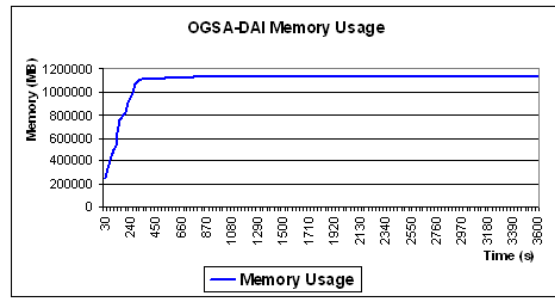


Figure 6: OGSA-DAI Memory consumption

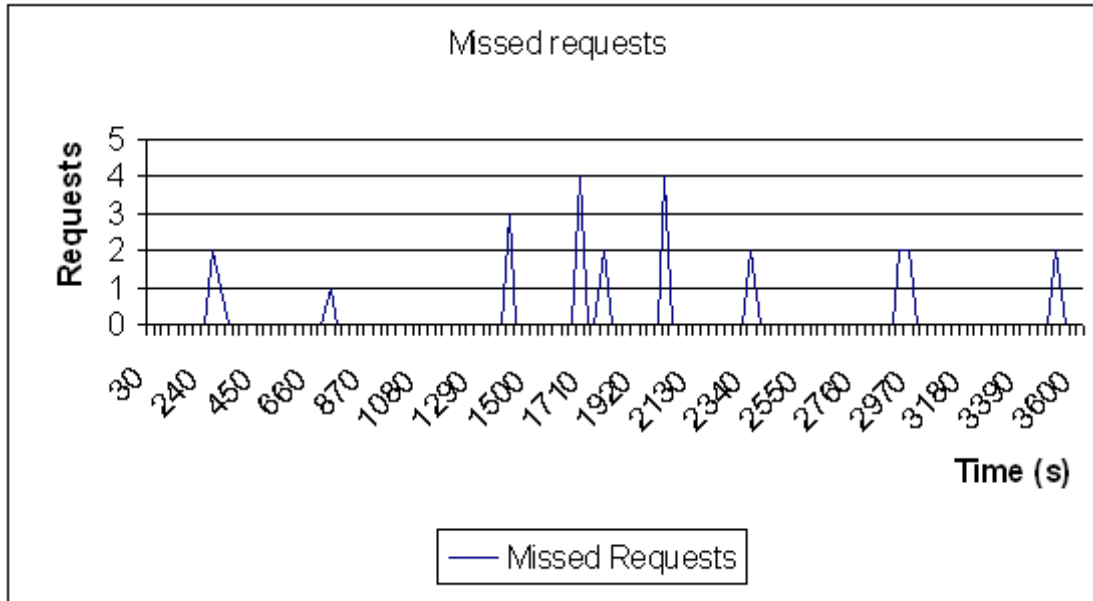


Figure 7: OGSA-DAI Requests missed during 1 hour with burst workload

VM to run a software load-balancer (VM-LB); one VM to run the main OGSA-DAI server; and a third VM where we create a hot-standby replica of the OGSA-DAI server. The VM-LB also runs some other software modules that will be responsible for the detection of software aging or other potential anomalies. When something anomalous is detected this module will trigger a rejuvenation action. In this action we do not restart the main server right away: we start the standby server, all the new requests and sessions are sent by the LB to this second server. The session-state is migrated from the primary to the secondary server and we wait for all the on-going requests to be finished in the primary server. When we are able to do this we can restart the main server without losing any in-flight request or session state. We call this a "clean" restart. During that restart process no in-flight request is lost, since we have a window of execution where we have both servers running in active mode. When all the requests are finished in the main server, then the server is restarted, as shown in Figure 8.

For lack of space, we refer the interested reader about the details of this rejuvenation mechanism to [17]. The most we can say is that the deployment of this framework is straightforward and does not require any change to the applications or the middleware containers. They are also neutral to the virtualization layer. Our scheme was applied to some other benchmarks in [17] and in the next section we present the results collected with OGSA-DAI.

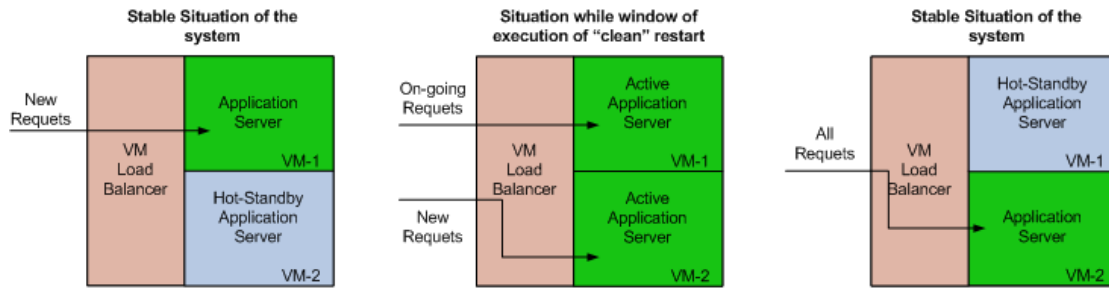


Figure 8: Virtualized Clustering for Server Rejuvenation process

5 Experimental results

To demonstrate the effectiveness of our rejuvenation mechanism we decided to make a deployment of OGSA-DAI using our virtualized clustering scheme followed by the installation of some software modules that provide the support for anomaly and aging detection and the triggering of rejuvenation actions. Since the problem of OGSA-DAI WSI 2.2 was spotted due to the memory leaks of Axis1.2 we decided to configure our framework to trigger some planned restarts depending on some thresholds of the memory usage. The memory use is collected every 10 seconds and when the available memory falls down that defined threshold our mechanism applies a clean and planned restart that, as a matter of fact, does not produce any downtime to the OGSA-DAI service.

In Figure 9 and Figure 10 we present the latency and throughput figures. First, we executed an experiment for one hour with 3 clients using a burst distribution workload. In this case we used the normal version of OGSA-DAI, with session scope.

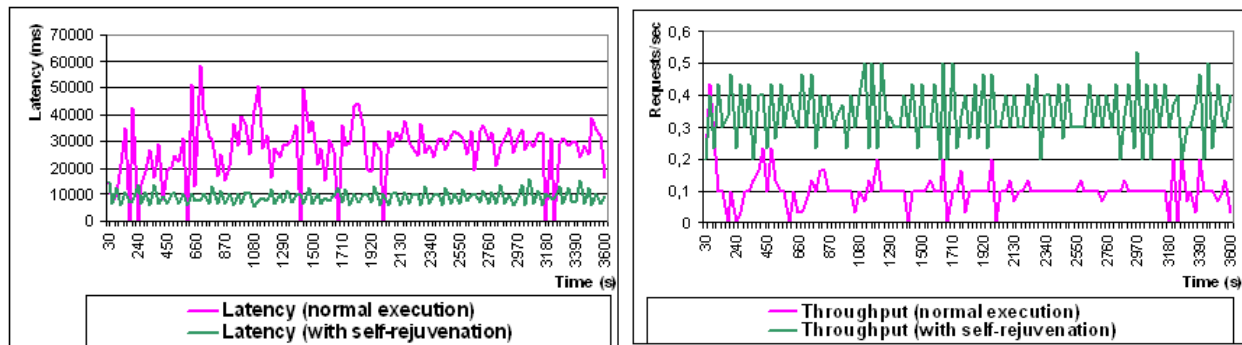


Figure 9: Latency (normal execution) vs Latency (with self-rejuvenation) Figure 10: Throughput (normal execution) vs Throughput (with self-rejuvenation)

Then we executed the same experiment but this time by applying our self-recovery mechanisms based on proactive restarts of the OGSA-DAI server. Our mechanism was configured to trigger a rejuvenation action when OGSA-DAI memory usage achieved around 50% of maximum memory usage (1024MB).

In Figure 11 we present the graph with the memory usage from one of servers. The graph clearly shows the moments in time when a rejuvenation action is triggered. In Table 2 we present more detailed numbers. We can observe that our approach achieve a very acceptable performance without any disruption of the service. Our approach was able to achieve a better throughput and latency than the default OGSA-DAI without missing any request. This proves the effectiveness of our scheme: we have been able to increase the availability and the performance of a Grid-Service that suffers from internal memory leaks in the SOAP layer. Our scheme brings a contribution for dependability.

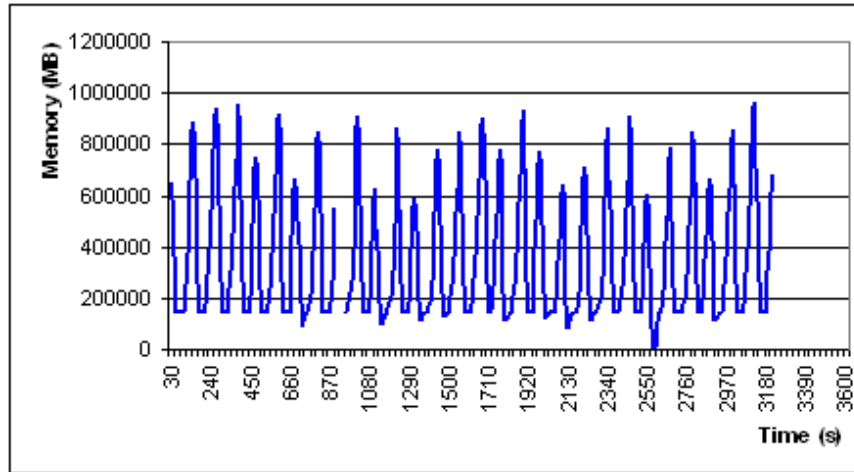


Figure 11: Memory Usage Performance using Self Recovery mechanism

| | Avg. Latency | Avg. Throughput | Avg. Memory Usage | Miss Requests |
|--------------------------------------|--------------|-----------------|-------------------|---------------|
| OGSA-DAI (normal version) | 29013,2 ms | 0,105 req/s | 1100784,4 Bytes | 25 |
| OGSA-DAI (with self-rejuvenation) | 9049,5 ms | 0,345 req/s | 313384,0 Bytes | 0 |

Table 2: Detailed results from 1 hour of execution

6 Conclusions

In this paper, we have presented the application of a self-healing mechanism that was developed by us in a case-study of a Grid-Service: OGSA-DAI. The normal version of this middleware suffers from some aging problems due to the use of Axis v1.2. The resulting performance is really unstable and we have been able to spot some undesired hang-ups when applying a burst distribution. We decided to apply our rejuvenation mechanism based on the thresholds alerts in the memory usage and we have been able to increase the availability and performance of OGSA-DAI without incurring in any additional investment of hardware: it is only necessary to install a virtualization layer and a set of software modules that provide support for self-recovery actions. The results presented in this paper clearly show the potential of our approach to achieve a dependable Grid service.

7 Acknowledgments

This research work is supported by the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265) and the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2004-07739-C02-01.

References

- [1] A.Avrizter, E.Weyuker. *Monitoring Smoothly Degrading Systems for Increased Dependability*. Empirical Software Eng. Journal, Vol 2, No 1, pp. 59-77, 1997
- [2] Apache.[Online] <http://httpd.apache.org/docs/>

- [3] V.Castelli, R.Harper, P.Heidelberg, S.Hunter, K.Trivedi, K.Vaidyanathan, W.Zeggert. *Proactive Management of Software Aging*. IBM Journal Research & Development, Vol. 45, No. 2, Mar. 2001
- [4] K.Cassidy, K.Gross, A.Malekpour. *Advanced Pattern Recognition for Detection of Complex Software Aging Phenomena in Online Transaction Processing Servers*. Proc. of the 2002 Int. Conf. on Dependable Systems and Networks, DSN-2002
- [5] A.Tai, S.Chau, L.Alkalaj, H.Hecht. *On-board Preventive Maintenance: Analysis of Effectiveness and Optimal Duty Period*. Proc. 3rd Workshop on Object-Oriented Real-Time Dependable Systems, 1997
- [6] Y.Huang, C.Kintala, N.Kolettis, N. Fulton. *Software Rejuvenation: Analysis, Module and Applications*. Proc. of Fault-Tolerant Computing Symposium, FTCS-25, June 1995
- [7] K.Vaidyanathan, K.Trivedi. *A Comprehensive Model for Software Rejuvenation*. IEEE Trans. on Dependable and Secure Computing, Vol, 2, No 2, April- 2005
- [8] K.Kaidyanathan, K.Gross. *Proactive Detection of Software Anomalies through MSET*. Workshop on Predictive Software Models (PSM 2004), Sept. 2004
- [9] 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
- [10] M. Rosenblum and T. Garfinkel. *Virtual Machine Monitors: Current Technology and Future Trends*. IEEE Internet Computing, May 2005, Vol. 38, No. 5.
- [11] R. Figueiredo, P. Dinda, J. Fortes. *Resource Virtualization Renaissance*. IEEE Computer, 38(5), pp. 28-69, May 2005
- [12] Renato J. Figueiredo , Peter A. Dinda , Jos A. B. Fortes. *A Case For Grid Computing On Virtual Machines*. Proceedings of the 23rd International Conference on Distributed Computing Systems, p.550, May 19-22, 2003
- [13] OGSA-DAI.[Online] <http://www.ogsadai.org.uk/>
- [14] Projects that use OGSA-DAI.[Online] <http://www.ogsadai.org.uk/about/projects.php>
- [15] XEN Source.[Online] <http://www.xensource.com/>.
- [16] W. Hoarau, S. Tixeuil, N. Rodrigues, D. Sousa, and L. Silva. *Benchmarking the OGSA-DAI Middleware Core*-GRID Technical Report Number TR-0060. October 5, 2006.
- [17] Luis Silva, Javier Alonso, Paulo Silva, Jordi Torres and Artur Andrzejak. *Using Virtualization to Improve Software Rejuvenation* Submitted for publication.