Available online at www.sciencedirect.com
ScienceDirect
Journal of Systems Architecture 52 (2006) 727–736
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

Evaluating the reliability of computational grids from the end user's point of view ☆

Eduardo Huedo *, Rubén S. Montero, Ignacio M. Llorente

Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain

Received 12 December 2005; received in revised form 11 April 2006; accepted 21 April 2006
Available online 9 June 2006

Abstract

Reliability, in terms of Grid component fault tolerance and minimum quality of service, is an important aspect that has to be addressed to foster Grid technology adoption. Software reliability is critically important in today's integrated and distributed systems, as is often the weak link in system performance. In general, reliability is difficult to measure, and specially in Grid environments, where evaluation methodologies are novel and controversial matters. This paper describes a straightforward procedure to analyze the reliability of computational grids from the viewpoint of an end user. The procedure is illustrated in the evaluation of a research Grid infrastructure based on Globus basic services and the GridWay meta-scheduler. The GridWay support for fault tolerance is also demonstrated in a production-level environment. Results show that GridWay is a reliable workload management tool for dynamic and faulty Grid environments. Transparently to the end user, GridWay is able to detect and recover from any of the Grid element failure, outage and saturation conditions specified by the reliability analysis procedure.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Grid computing; Globus Toolkit; GridWay meta-scheduler; Fault tolerance; Quality of service

1. Introduction

The key aspect that differentiates Grid computing from other distributed computing paradigms is site autonomy. Grid technology allows the interconnection of resources scattered across several administrative domains, each with its own security policy and distributed resource management system. Consequently, site autonomy generates outages of resource and network elements due to local administration decisions (e.g. scheduled switch-off to perform maintenance tasks). This means that resources shared within a virtual organization can

☆ This research was supported by Consejería de Educación de la Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE), through BIO-GRIDNET Research Program S-0505/TIC/000101, and by Ministerio de Educación y Ciencia, through the research grant TIC2003-01321. The authors participate in the EGEE project, funded by the European Union.

* Corresponding author. Tel.: +34 91 394 75 38; fax: +34 91 394 75 27.

E-mail addresses: ehuedo@fdi.ucm.es (E. Huedo), rubensm@dacya.ucm.es (R.S. Montero), llorente@dacya.ucm.es (I.M. Llorente).

1383-7621/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2006.04.003

be added or removed continuously. Moreover, as Grid resources belong to different administrative domains; once a job is submitted, it can be freely cancelled or suspended by the resource owner.

On the other hand, the performance offered by Grid infrastructure components may change dynamically during the run-time of an application. Grid users access resources that are being exploited by other Grid users, as well as by internal users. This fact may cause, where local job managers does not guarantee exclusive access to compute resources, that initially idle hosts become saturated, and vice versa. Moreover, in dedicated batch systems, the saturation of the resource may increase the queue wait time to an unacceptable value. We should also consider that most of Grid infrastructures use shared public networks, mainly research and academic networks, to interconnect resources.

Grid component outage (due to site autonomy) and performance variability (due to resource sharing) are seen by the end user as a failure condition or a quality of service (QoS) degradation, respectively. These conditions together with real failures, like system crash or network disconnection, make failures the rule rather than the exception in Grid environments. Therefore, one of the most valuable characteristics of Grid tools, apart from the performance they can achieve, is fault resilience. In this sense, reliability in a grid is considered by the end user as Grid component fault tolerance and minimum QoS achievement. We foresee that the reliability concern will be more and more critical in the expansion and extended use of Grid technology [1].

Evaluation of Grid infrastructures is a novel, controversial and difficult task. The capabilities of a computational grid, made up of heterogeneous components with dynamic characteristics, are usually provided by three layers [2]: Grid fabric (worker nodes, operating systems, local schedulers or network links), core Grid middleware (job submission, resource discovery and monitoring, or data management services) and user-level Grid middleware (meta-schedulers, resource brokers or Grid portals.). Therefore, we must take into account that the evaluation of a Grid infrastructure should analyze the coordinated use of all its components to execute jobs with representative execution profiles. It is clear that reliability must be a concern in the development of services on each layer in a Grid infrastructure. However, outages, failures and performance contract violations are frequent phenomena in spite of the taken precautions. Therefore, user-level Grid middleware is the final responsible for providing detection and recovery strategies to meet the reliability requirements of the end user.

Several authors have previously addressed reliability in Grids. For example, Hwang et al. [3] present a failure handling system based on workflows. The drawback of their approach is that application code must be changed to deliver events like task start/end or the occurrence of user exceptions. Jin et al. [4] propose a framework for the adaptive deployment of failure detectors and, based on it, a policy-based failure handling mechanism to choose the appropriate failure recovery method. Kola et al. [5] provide a classification of faults in large distributed systems (with the main focus on grids). Finally, Lanferman et al. [6] first introduced migration as a technique for obtaining fault tolerance in grids. Other works have proposed solutions for applications composed of communicating tasks [7–9].

The aim of this paper is, firstly, to present a procedure which allows the evaluation of the reliability of a computational Grid environment from the end user's point of view; and, secondly, to apply it to evaluate a Grid environment based on Globus basic services and the GridWay[1] meta-scheduler. In Section 2, we describe the reliability analysis procedure. Then, the functionality of GridWay and its fault tolerance features are described in Sections 3 and 4, respectively. In Section 5, we show the application of the procedure to evaluate the reliability of a research testbed, and we present several experiments performed on a production-level Grid infrastructure. Finally, Section 6 presents the main conclusions of our work.

2. A procedure to evaluate grid reliability

Computational Grid environments are difficult to efficiently harness due to their heterogeneous nature, the unpredictable changing performance and the frequent failure and outage conditions. Adaptive scheduling and execution are some of the techniques proposed in the literature [10–13] to achieve a reasonable degree of application performance and fault tolerance.

Therefore, a suitable methodology for reliability evaluation should help to determine the fault toler-

[1] www.gridway.org

ance and dynamic adaptation capabilities of the Grid environment. However, such functionality is difficult to measure. To this end, we define a reliable Grid environment as the one in which a job can, transparently to the user, continue its execution (at least from the beginning) in other resource when each one of the following conditions of failure and loss of QoS takes place [14]:

- *Job related*:
  - Job cancellation (failure): A Job could be cancelled for several reasons, for example by the local resource management system when it exceeds the wall time limit, or by the system administrator to preserve system performance.
  - Job suspension (QoS loss): Higher priority jobs continuously entering the system could prevent the execution of the Grid job. Equally, the system administrator or the local resource manager could freely hold a waiting job, preventing its execution.
- *System related*:
  - System crash (failure): Grid resources could unpredictably fail. These failures comprise hardware, base software, and Grid middleware components. Moreover, system administrators are freely to shutdown their resources, for example, due to local site maintenance. Moreover, failures in the client system should be also considered.
  - System saturation (QoS loss): Resource load changes dynamically, since Grid resources are shared between other Grid and non-Grid users (possibly with higher priority).
- *Network related*:
  - Network disconnection (failure): Grid connections could unpredictably fail. Moreover, system administrators are freely to disconnect their resources, for example, due to local site maintenance.
  - Network saturation (QoS loss): Network traffic varies dynamically, since Grid resources are in general connected through public, non-dedicated, networks.

Therefore, the reliability assessment process consists on artificially generating the six failure conditions proposed above.

Application-related failures must be handled at the application level, and therefore they are out of the scope of this procedure. This failures comprise design or coding errors (memory leaks or numerical exceptions) and other errors caused by shared access (wrong access permissions, full disks or memory outages). In this sense, it is important that the underlying middleware should provide information about the exit status of each job.

3. The GridWay meta-scheduler

The Globus Toolkit has become a de facto standard in Grid computing [15]. Globus services allow secure and transparent access to resources across multiple administrative domains, and serve as building blocks to implement the stages of Grid scheduling [16]. However, the user is responsible for manually performing all the submission steps in order to achieve any functionality. Moreover, the Globus Toolkit does not provide any native support for job migration and therefore for *adaptive execution*, a key aspect to achieve reliability in a grid as discussed in the previous section.

To overcome these limitations, we have developed an experimental framework that allows an easier and more efficient execution of jobs on a dynamic Grid environment in a "*submit* & *forget*" fashion. The core of the GridWay meta-scheduler [17] is a personal *submission agent* that performs all scheduling stages and watches over the correct and efficient execution of jobs on Globus-based grids. Adaptation to changing conditions is achieved by both *adaptive scheduling* and *adaptive execution*. Once the job is initially allocated, it is rescheduled when one of the following circumstances occurs:

(1) *Grid related*:
   - A "better" resource is discovered (opportunistic migration).
   - The remote resource or its network connection fails (fail-over migration).
   - The submitted job is cancelled or suspended by the resource administrator or the local resource management system.

(2) *Application related*:
   - Performance degradation or performance contract violation is detected in terms of application intrinsic metrics (*self-monitoring* application).
   - The resource requirements or preferences of the application change (*self-migrating* application).

Job execution is performed in three separated steps by the following modules:

(1) *Prolog*: It prepares the remote system by creating a experiment directory and transferring the input files from the client.
(2) *Wrapper*: It executes the actual job and obtains its exit status code.
(3) *Epilog*: It finalizes the remote system by transferring the output files back to the client and cleaning up the experiment directory.

Migration is commonly implemented by restarting the job on the new candidate host, therefore the job should generate restart files at regular intervals in order to restart execution from a given point. However, for some application domains the cost of generating and transferring restart files could be greater than the saving in compute time due to checkpointing [18]. Hence, if the checkpointing files are not provided the job should be restarted from the beginning.

The rescheduling reason is evaluated to decide if the migration is feasible and worthwhile. Some reasons, like job cancellation or failure, make the migration process immediately start, even if the new host presents lower rank than the current one. Other reasons, like new resource discovery, make the migration process start only if the new selected host presents a higher enough rank. In this case, the time to finalize and file transfer costs [18] must be considered to evaluate if the migration is worthwhile.

When a migration order is finally granted, the *wrapper* is cancelled (if it is still running), then the *prolog* is submitted to the new candidate resource, preparing it and transferring all the needed files to it, including the `restart files` from the old resource. After that, the *epilog* is submitted to the old resource (if it is still available), but no output file staging is performed, it only cleans up the remote system. And finally, the *wrapper* is submitted to the new candidate resource.

GridWay fully implements the Distributed Resource Management Application API (DRMAA)[2] [19] standard proposed by the Global Grid Forum.[3]

[2] www.drmaa.org
[3] www.ggf.org.

4. GridWay support for fault tolerance and quality of service

GridWay provides the application with the fault detection capabilities needed in a Grid environment, by:

- Handling the Globus GRAM (Grid Resource Allocation and Management) *job manager* callbacks [20]. The GRAM callbacks notify submission failures that include connection, authentication, authorization, RSL (Resource Specification Language) parsing, executable or input staging, credential expiration, among others.
- Periodically probing the Globus *job manager* [20]. If the *job manager* does not respond after a given number of tries, then a resource or network failure is assumed.
- Parsing the standard output of the *prolog*, *wrapper* and *epilog* executables. In the case of the *wrapper*, this is also useful to capture the job exit code, which is used to determine whether the job was successfully executed or not. If the job exit code is not set, the job was prematurely terminated, so it failed or was intentionally cancelled.

When an unrecoverable failure is detected, GridWay retries the submission of *prolog*, *wrapper* or *epilog* a number of times specified by the user. If the failure persists, GridWay performs an action chosen by the user among two possibilities: stop the job for manually resuming it later, or automatically generate a rescheduling event to migrate the job.

Application performance slowdown is detected by GridWay by means of two mechanisms:

- A *performance evaluator* is periodically executed at each *monitoring* interval to evaluate a rescheduling condition. Different strategies could be implemented, from the simplest one based on querying the Grid information system about workload parameters to more advanced strategies based on detection of performance contract violations [21]. A mechanism to deal with application own metrics is provided since the files processed by the *performance evaluator* could be dynamically generated by the running job. The rescheduling condition verified by the *performance evaluator* could be based on the performance history using advanced methods like fuzzy logic, or comparing the performance with the initial performance attained, or a base performance.

- A running job could be temporally suspended by the resource administrator or by the local queue scheduler on the remote resource. The submission agent takes count of the overall *suspension time* of its job and requests a rescheduling if it exceeds a given threshold. Notice that the *maximum suspension time* threshold is only effective on queue-based resource managers.

The job exit conditions can be checked through the `drmaa_wif*()` routines included in the DRMAA interface [19], which is fully implemented by GridWay, and so the appropriate corrective action can be undertaken if application-related failures occur. Regarding failures in the client system, GridWay persistently saves the status of each submitted job and is able to recover each job's contact and life-cycle.

5. Experiences

In this section we evaluate the reliability of two testbeds based on Globus pre-Web Services components, according to the criteria described in Section 2. First, we consider a small-scaled, controlled testbed, to easily generate failure conditions. Then, we perform large-scale experiments over a real, production-level, testbed.

Both testbeds are based on an "hourglass" model, where clients have access to a wide range of services provided through a limited and standardized set of protocols and interfaces. In particular, Grid resources only provide Globus basic services for security, job allocation, discovery, monitoring, and data transfer. The workload management functionality is provided by GridWay, acting as user-level Grid middleware. Note that there is no need for specific migration or fault tolerance Grid-wide services. This approach considerably simplifies Grid deployment and provides a straightforward resource sharing, as resources are accessed by using de facto standard protocols and interfaces.

In the first set of experiments, the six failure conditions proposed in Section 2 were artificially generated as follows:

- *Job related*:
  - Job cancellation (failure): Several running jobs were cancelled by using the `qdel` command on resources managed with PBS or SGE. GridWay detected the job cancellation as the job exit code was not specified.
  - Job suspension (QoS loss): Several running jobs were suspended by using the `qhold` command on resources managed with PBS or SGE. Job suspension was detected when the job remained in the SUSPEND state of the Globus GRAM module longer than the given threshold specified by the user.
- *System related*:
  - System crash (failure): During the execution of a job, the resource was shut down. GridWay was able to detect the failure when the polling of the job failed.
  - System saturation (QoS loss): During the execution of a job, the resource was saturated with an artificially generated load. GridWay was able to detect the performance degradation suffered by the application through its `performance profile`, which reflected a reduction in the performance metric provided by the application. This case is described in more detail below.
- *Network related*:
  - Network disconnection (failure): During the execution of a job, the resource was disconnected from the network by tugging on the wire. Again, GridWay detected the failure when the polling of the job failed.
  - Network saturation (QoS loss): Network saturation is only considered if it causes polling failures, so the behavior would be the same as in a system crash. Nevertheless, GridWay uses network status information to rank resources during the scheduling process [18].

GridWay transparently handles all the described failures, resulting, independently on the failure nature, in the rescheduling of the job to other resource from the last available checkpoint [17].

In order to better illustrate the procedure described above, we consider in detail the resource failure due to a system saturation, which in fact is the most complex failure condition to handle. The job is equipped with a monitor to record the overall system load and the percentage of CPU devoted to the job itself. This information is used by the *performance evaluator* module to detect performance slowdowns.

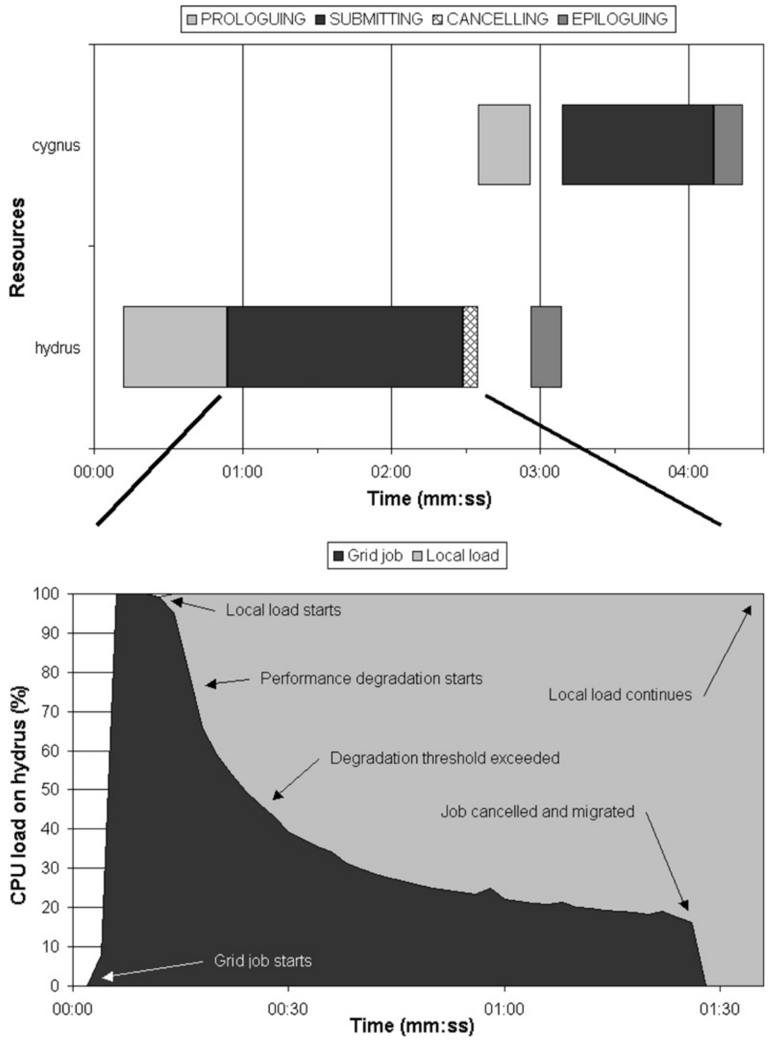

Fig. 1. Execution profile (top) and CPU load on hydrus (bottom) when a migration is performed due to a performance degradation.

Let us consider two machines of the testbed hydrus and cygnus. A job is initially submitted to hydrus and starts executing. At time step 00:14 an artificial load is introduced in the system, which reduces the effective performance received by the job below the minimum established by the user (50% of total CPU load, time step 00:24). GridWay detects this QoS loss condition at time step 01:28, and reschedules the job on cygnus. Fig. 1 shows the execution profile of this job.

In order to analyze the reliability of the environment in a real-life situation, we have performed a large-scale experiment with a Bioinformatics application. Each experiment consists of the execution of a protein structure prediction algorithm over a family of 80 orthologous proteins (proteins performing the same function in different organisms), where each protein was analyzed in a separate job [22].

Five experiments were conducted over a joint IRISGrid[4] and EGEE[5] testbed composed of 13 Spanish sites, and a total of 528 CPUs. The testbed is described in Table 1. All sites were connected by means of the Spanish Research and Education Network, RedIRIS,[6] as seen in Fig. 2. The testbed results in a very heterogeneous infrastructure, since it presents several middleware (different versions of LCG-2 and Globus), architectures (Alpha, Intel

[4] http://www.irisgrid.es
[5] http://www.eu-egee.org
[6] http://www.rediris.es

Table 1
IRISGrid and EGEE resources contributed to the experiment

Testbed	Site	Resource	Processor	Speed	Nodes	RM
IRISGrid	RedIRIS	heraclito	Intel Celeron	700 MHz	1	Fork
		platon	2 × Intel PIII	1.4 GHz	1	Fork
		descartes	Intel P4	2.6 GHz	1	Fork
		socrates	Intel P4	2.6 GHz	1	Fork
	DACYA-UCM	aquila	Intel PIII	700 MHz	1	Fork
		cepheus	Intel PIII	600 MHz	1	Fork
		cygnus	Intel P4	2.5 GHz	1	Fork
		hydrus	Intel P4	2.5 GHz	1	Fork
	LCASAT-CAB	babieca	Alpha EV67	450 MHz	30	PBS
	CESGA	bw	Intel P4	3.2 GHz	80	PBS
	IMEDEA	llucalcari	AMD Athlon	800 MHz	4	PBS
	DIF-UM	augusto	4 × Intel Xeon (HT)	2.4 GHz	1	Fork
		caligula	4 × Intel Xeon (HT)	2.4 GHz	1	Fork
		claudio	4 × Intel Xeon (HT)	2.4 GHz	1	Fork
	BIFI-UNIZAR	lxsrv1	Intel P4	3.2 GHz	50	SGE
EGEE	LCASAT-CAB	ce00	Intel P4	2.8 GHz	8	PBS
	CNB	mallarme	2 × Intel Xeon	2.0 GHz	8	PBS
	CIEMAT	lcg02	Intel P4	2.8 GHz	6	PBS
	FT-UAM	grid003	Intel P4	2.6 GHz	49	PBS
	IFCA	gtbcg12	2 × Intel PIII	1.3 GHz	34	PBS
	IFIC	lcg2ce	AMD Athlon	1.2 GHz	117	PBS
	PIC	lcgce02	Intel P4	2.8 GHz	69	PBS

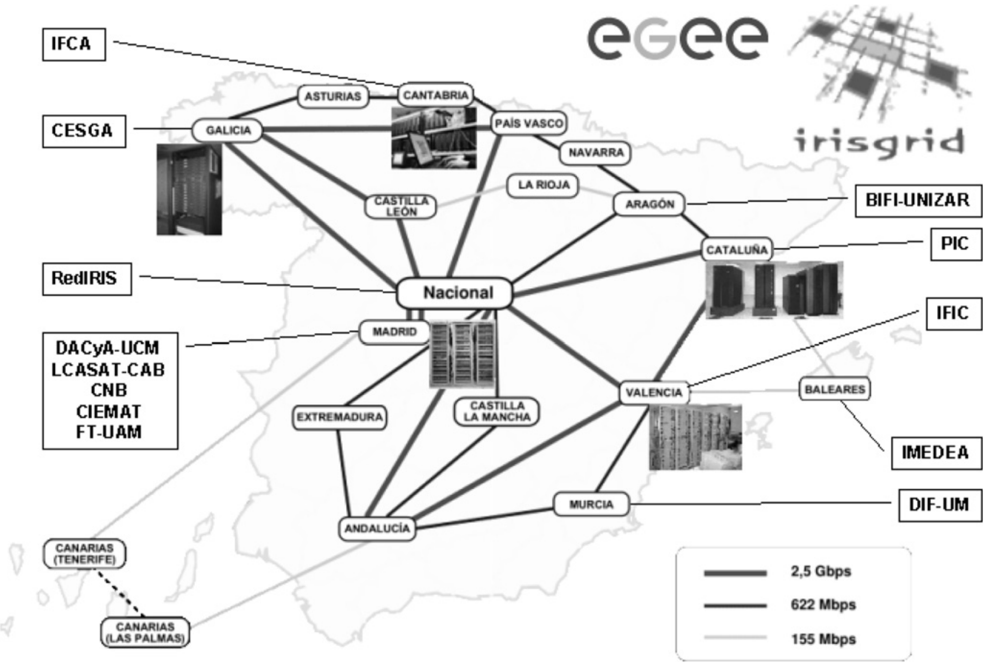

Fig. 2. Geographical distribution and interconnection network of sites.

and AMD), processor speeds (from 450 MHz to 3.2 GHz), resource managers (PBS, SGE and fork), and network links. In order to not saturate the whole testbed, only a maximum of four CPUs were simultaneously used on each resource, so a maximum of 64 CPUs were used.

During the whole time employed for each experiment, some of the job executions failed or were

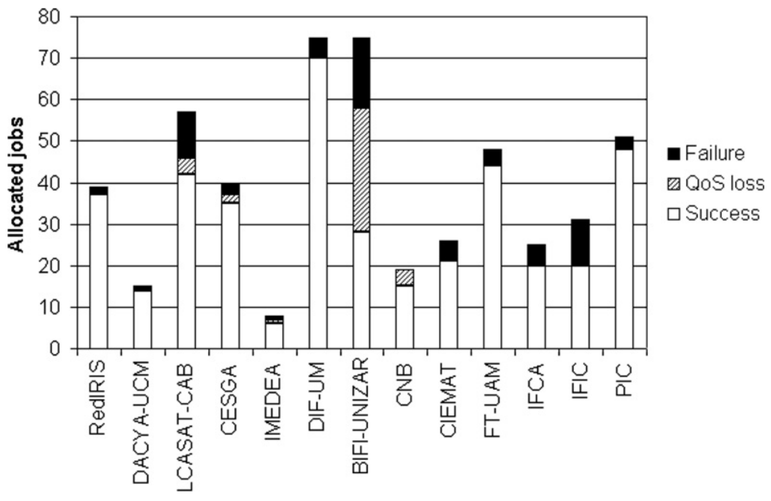

Fig. 3. Distribution of total successful executions, failure conditions and QoS loss conditions across sites.

suspended for a long time. GridWay detected these failure and QoS loss conditions, and migrated those jobs to other resources. Fig. 3 shows the distribution of total successful executions, failure conditions (job cancellation, system crash or network disconnection) and QoS loss conditions (job suspension) across sites.

6. Conclusions

We have proposed a reliability analysis procedure for computational Grid environments from the end user's point of view. The procedure is based on a set of probes, which require the user-level Grid middleware to incorporate fault tolerance and dynamic adaptation capabilities. We have illustrated the procedure in the evaluation of a research Grid environment based on Globus basic services and the GridWay meta-scheduler.

Moreover, we have demonstrated the GridWay support for fault tolerance in a production-level environment with a real application. Results show that a reasonable level of reliability for the end user can be attained through the GridWay meta-scheduler over any Globus-based infrastructure. Moreover, the experiences show that failures and outages of resources are frequent phenomena, even in production-level Grid environments.

Acknowledgements

We would like to thank all the institutions involved in the IRISGrid initiative and the EGEE project, in particular those who collaborated in the experiments. They are Red Académica y de Investigación Nacional (RedIRIS), Departamento de Arquitectura de Computadores y Automática (DACyA) at Universidad Complutense de Madrid (UCM), Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas (LCASAT) at Centro de Astrobiología (CAB), Centro de Supercomputación de Galicia (CESGA), Instituto Mediterráneo de Estudios Avanzados (IMEDEA), Facultad de Informática (DIF) at Universidad de Murcia (UM), Instituto de Biocomputación y Física de Sistemas Complejos (BIFI) at Universidad de Zaragoza (UNIZAR), Instituto de Física de Cantabria (IFCA), Instituto de Física Corpuscular (IFIC), Centro Nacional de Biotecnología (CNB), Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT), Departamento de Física Teórica (FT) at Universidad Autónoma de Madrid (UAM) and Port d'Informació Científica (PIC).

References

[1] J.M. Schopf, B. Nitzberg, Grids: The Top Ten Questions, Scientific Programming, special issue on Grid Computing 10 (2) (2002) 103–111.

[2] M. Baker, R. Buyya, D. Laforenza, Grids and Grid technologies for wide-area distributed computing, Software – Practice and Experience 32 (15) (2002) 1437–1466.

[3] S. Hwang, C. Kesselman, Grid workflow: a flexible failure handling framework for the grid, Journal of Grid Computing 1 (3) (2003) 251–272.

[4] H. Jin, X. Shi, W. Qiang, D. Zou, DRIC: dependable grid computing framework, IEICE Transactions on Information and Systems E89-D (2) (2006) 612–622.

[5] G. Kola, T. Kosar, M. Livny, Fault in large distributed systems and what we can do about them, in: Proceedings of

the Euro-Par 2005LNCS, vol. 3648, Springer-Verlag, 2005, pp. 442–453.
[6] G. Lanfermann, G. Allen, T. Radke, E. Seidel, Nomadic migration: fault tolerance in a disruptive grid environment, in: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), 2002.
[7] E. Gabriel, G. Fagg, A. Bukovsky, T. Angskun, J. Dongarra, A fault-tolerant communication library for grid environments, in: Proceedings of the 17th ACM International Conference on Supercomputing (ICS'03), 2003.
[8] W. Namyoon, C. Soonho, J. Hyungsoo, et al., MPICH-GF: Providing Fault Tolerance on Grid Environments, in: Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), 2003.
[9] A. Luckow, B. Schnor, Migol: A fault-tolerant service framework for MPI applications in the grid, in: Proceedings of the 12th European PVM/MPI Users' Group MeetingLNCS, vol. 3666, Springer-Verlag, 2005, pp. 258–267.
[10] E. Huedo, R.S. Montero, I.M. Llorente, The Grid Way framework for adaptive scheduling and execution on grids, Scalable Computing – Practice and Experience 6 (3) (2006) 1–8.
[11] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S.M. Figueira, J. Hayes, G. Obertelli, J.M. Schopf, G. Shao, S. Smallen, N.T. Spring, A. Su, D. Zagorodnov, Adaptive computing on the grid using AppLeS, IEEE Transactions on Parallel and Distributed Systems 14 (4) (2003) 369–382.
[12] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, J. Shalf, The Cactus Worm: experiments with dynamic resource discovery and allocation in a grid environment, Journal of High Performance Computing Applications 15 (4) (2001) 345–358.
[13] S.S. Vadhiyar, J.J. Dongarra, Self-Adaptivity in Grid Computing, Concurrency and Computation: Practice and Experience 17 (2–4) (2005) 235–257.
[14] H.-M. Lee, K.-S. Chung, S.H. Jin, D.-W. Lee, W.-G. Lee, S.Y. Jung, H.-C. Yu, A fault tolerance service for QoS in grid computing, in: Proceedings of the International Conference on Computational Science (ICCS 2003)LNCS, Springer-Verlag, 2003, pp. 286–296.
[15] I. Foster, C. Kesselman, Globus: a metacomputing infrastructure toolkit, Journal of Supercomputer Applications 11 (2) (1997) 115–128.
[16] J.M. Schopf, Ten actions when superscheduling, Tech. Rep. GFD-I.4, Scheduling Working Group – The Global Grid Forum, 2001.
[17] E. Huedo, R.S. Montero, I.M. Llorente, A framework for adaptive execution on grids, Software – Practice and Experience 34 (7) (2004) 631–651.
[18] R.S. Montero, E. Huedo, I.M. Llorente, Grid resource selection for opportunistic job migration, in: Proceedings of the 9th International Conference on Parallel and Distributed Computing (Euro-Par 2003)LNCS, vol. 2790, Springer-Verlag, 2003, pp. 366–373.
[19] H. Rajic, R. Brobst, W. Chan, F. Ferstl, J. Gardiner, J.P. Robarts, A. Haas, B. Nitzberg, J. Tollefsrud, Distributed resource management application API specification 1.0, Tech. Rep., DRMAA Working Group – The Global Grid Forum, 2003.
[20] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A resource management architecture for metacomputing systems, in: Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel ProcessingLNCS, vol. 1459, Springer-Verlag, 1998, pp. 62–82.
[21] F. Vraalsen, R.A. Aydt, C.L. Mendes, D.A. Reed, Performance contracts: predicting and monitoring grid application behavior, in: Proceedings of the 2nd International Workshop on Grid Computing (Grid 2001), 2001.
[22] E. Huedo, U. Bastolla, R.S. Montero, I.M. Llorente, A framework for protein structure prediction on the grid, New Generation Computing 23 (4) (2005) 277–290.

Eduardo Huedo received his M.E. (1999) and Ph.D. (2004) in Computer Science by the Universidad Complutense de Madrid (UCM). He is an Assistant Professor of Computer Architecture and Technology in the Department of Computer Architecture and System Engineering at UCM. Previously he worked as a Post-Doctoral Researcher in Grid Computing in the Advanced Computing Laboratory at Centro de Astrobiología (CSIC-INTA), associated to NASA Astrobiology Institute. His research areas are Performance Management and Tuning, Parallel and Distributed Computing and Grid Technology.

Rubén S. Montero received his B.S. in Physics (1996), M.S. in Computer Science (1998) and Ph.D. in Computer Architecture (2002) from the Universidad Complutense de Madrid (UCM). He is an Associate Professor of Computer Architecture and Technology at UCM. He has held several research appointments at ICASE (NASA Langley Research Center), where he worked on computational fluid dynamics, parallel multigrid algorithms and Cluster computing. Nowadays, his research interests lie mainly in Grid Technology, in particular in adaptive scheduling, adaptive execution and distributed algorithms.

Ignacio M. Llorente received his B.S. in Physics (1990), M.S. in Computer Science (1992) and Ph.D. in Computer Architecture (1995) from the Universidad Complutense de Madrid (UCM). He is an Executive M.B.A. by Instituto de Empresa since 2003. He is a full Professor of Computer Architecture and Technology in the Department of Computer Architecture and System Engineering at UCM and Senior Scientist at

Centro de Astrobiología (CSIC-INTA), associated to NASA Astrobiology Institute. He has held several appointments since 1997 as a Consultant in High Performance Computing and Applied Mathematics at ICASE (NASA Langley Research Center). His research areas are Information Security, High Performance Computing and Grid Technology.