Parallel Processing Letters

FAILURE MANAGEMENT IN GRIDS: THE CASE OF THE EGEE INFRASTRUCTURE

KYRIAKOS NEOCLEOUS, MARIOS D. DIKAIAKOS
Department of Computer Science, University of Cyprus
1678 Nicosia, Cyprus
{*kyriacos,mdd*}*@cs.ucy.ac.cy*

and

PARASKEVI FRAGOPOULOU, EVANGELOS P. MARKATOS
Institute of Computer Science, Foundation of Research and Technology-Hellas
1385 Heraklion-Crete, Greece
{*fragopou,markatos*}*@ics.forth.gr*

Received March 2007
Revised September 2007
Communicated by S.G. Akl

ABSTRACT

The emergence of Grid infrastructures like EGEE has enabled the deployment of large-scale computational experiments that address challenging scientific problems in various fields. However, to realize their full potential, Grid infrastructures need to achieve a higher degree of dependability, i.e., they need to improve the ratio of Grid-job requests that complete successfully in the presence of Grid-component failures. To achieve this, however, we need to determine, analyze and classify the causes of job failures on Grids. In this paper we study the reasons behind Grid job failures in the context of EGEE, the largest Grid infrastructure currently in operation. We present points of failure in a Grid that affect the execution of jobs, and describe error types and contributing factors. We discuss various information sources that provide users and administrators with indications about failures, and assess their usefulness based on error information accuracy and completeness. We describe two real-life case studies, describing failures that occurred on a production site of EGEE and the troubleshooting process for each case. Finally, we propose the architecture for a system that could provide failure management support to administrators and end-users of large-scale Grid infrastructures like EGEE.

Keywords: Grid, failure, EGEE, failure management

1. Introduction

Recent experimental studies have shown that jobs submitted by users to large-scale, multi-institutional Grid infrastructures often fail to complete successfully. For example, data collected and analysed by the WISDOM project [14], which submits tens of thousands of jobs to the EGEE infrastructure [1] in the context

of a drug-design effort [12], indicate that only the 65% of submitted jobs executed successfully. In the case of a Grid-job failure it is up to the end-user or the Grid administrator to detect the failure, to identify its cause, to re-submit the job, and to try to fix the problem(s) that caused the failure. Detecting and managing failures is an important step toward the goal of a dependable Grid. This is an extremely complex task, however, as it currently relies on ad-hoc monitoring and manual intervention. Automating this task seems difficult due to intrinsic characteristics of the Grid environment: Grids are not administered centrally and, therefore, it is hard to access the remote sites in order to monitor failures; also, Grid systems are complex and extremely large; thus, it is difficult to acquire and analyze failure feedback.

In our work, we investigate the reasons behind Grid-job failures, in order to gain an insight on how to build a more reliable Grid infrastructure. We concentrate on the problem of Grid reliability by focusing on jobs that fail to complete successfully, either providing no output or providing incorrect output. This study is conducted in order to unmask the root causes of such failures within the Grid system. We use the term *Grid reliability* as an indication of the extend to which Grid components behave in the way expected by their peers (Grid clients and services). In general, a reliable system is by no means an error-free system; failures would undoubtedly still occur. A reliable system must anticipate and be able to handle failures in various ways, such as by failure *detection, masking, tolerance,* and *recovery.* The handling of failures is particularly complex in large distributed environments such as the Grid, since a lot of components are involved and some may fail while others continue to function properly. A reliable Grid system should fail in predictable ways; if a component fails, the rest of the system must be able to adapt to the changed conditions (such as the lack of a service that has crashed, or the erroneous/incoherent information provided by a faulty service) and maintain an acceptable state; if that is impossible, it should at least be able to recover from the failure and return to the last known correct state.

In the next section, we provide an overview of the architecture, the job-execution model, and the operations of the large-scale Grid infrastructure of EGEE [1]. EGEE is currently the largest Grid infrastructure in operation, comprising 250 sites worldwide with more than 36,000 CPUs, 5PB of storage, supporting over 80 Virtual Organizations. In Section 3, we present possible points of failure in EGEE and describe grid error types and contributing factors. In Section 4, we discuss various services of the EGEE infrastructure that provide error information about failures of Grid components, and assess their usefulness based on error information accuracy and completeness. We also present two case studies, describing representative failures that occurred on the University of Cyprus production site of EGEE (CY01), with an accompanied analysis and troubleshooting process for each case. In Section 5 we discuss the challenges and requirements for a Failure Management System that could support Grid administrators and end users. We close by drawing some conclusions and discussing future work.

2. Grid Computing and EGEE

Computing Grids are usually very large scale services that enable the sharing

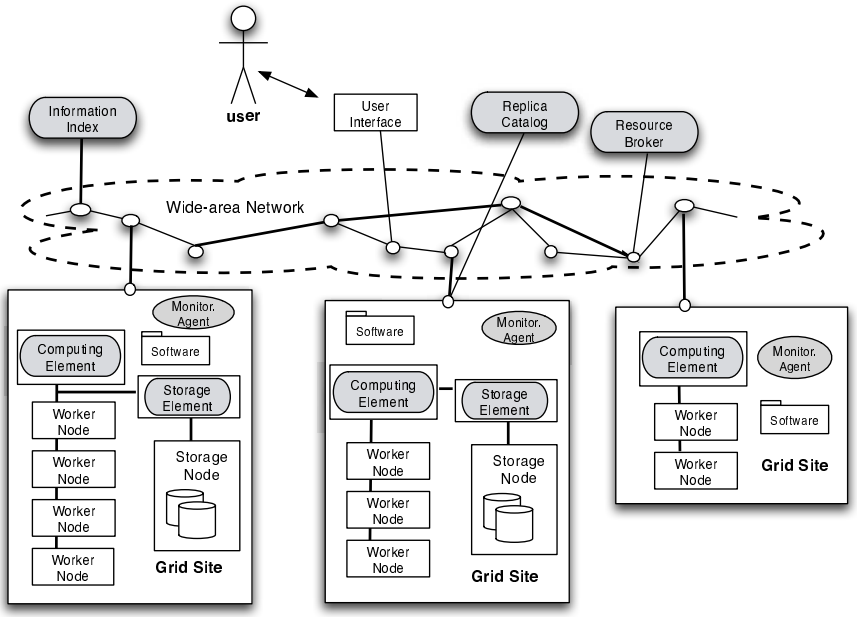

Fig. 1. *Grid architecture*

of heterogeneous resources (hardware and software) over an open network such as the Internet. A Grid is organised in Virtual Organisations (VOs) [20], collections of computational and storage resources, application software, as well as individuals (end-users) that usually have a common research area. Access to Grid resources is provided to VO members through the Grid middleware, which exposes high-level programming and communication functionalities to application programmers and end-users, enforcing some level of resource virtualisation [24]. VO membership and service brokerage is regulated by *access and usage policies* agreed among the infrastructure operators, the resource providers, and the resourse consumers.

The European project Enabling Grids for E-sciencE (EGEE) currently supports the largest grid infrastructure in the world, with more than 250 participating sites, 36,000 CPUs and 5PB disk storage. EGEE uses the LHC Computational Grid (LCG) middleware [5], while the new generation gLite middleware [2] is already being deployed at several sites. An overview of the EGEE architecture is presented in Figure 1

Within EGEE there exist several Virtual Organisations (VOs). Users registered within a specific VO obtain credentials for single Grid sign-on [15] that enables them to have access to the entire set of resources within (belonging to) that particular VO, despite the fact that such resources span different Grid sites across different countries.

Users have access to a User Interface (UI) node for submitting jobs to the Grid, for requesting job status and resources information, and for obtaining the output from completed jobs. In brief, a Grid job is usually a set of input files (the *input*

sandbox) and an executable that processes the given input on a set of Grid resources, according to the user requirements set forth in the Job Description Language (JDL) file that accompanies every Grid job submission. The Job Description Language (JDL) is a user-oriented language for describing jobs [16] and the information obtained from a JDL file is taken into account by the Grid Workload Management System (WMS) [22] components in order to schedule and submit a job. A job can have particular user-defined requirements for the resources it needs, such as computational capacity, physical memory capacity, the proximity (network latency-wise) of certain files that will be used as input, and the availability of specific application software. Grid jobs can be classified as CPU-intensive and data-intensive, depending on the type of work performed.

Jobs are submitted from the UI to a Resource Broker (RB), a central (global) Grid service. The RB is a component of the distributed Workload Management System (WMS) of a Grid infrastructure [23] which performs *matchmaking* by identifying a set of resources that satisfy the job requirements. The matchmaking is done based on data received by querying an Information Index, another central service that complements the WMS by providing up-to-date information about the state of Grid resources, usually spanning several sites.

If the matchmaking is successful, the job is sent from the RB to the matching Computing Element (CE) for execution. A Computing Element is at site level and it is comprised of the Grid Gate node and several Worker Nodes (WNs). The services running on the Grid Gate node are primarily responsible for authenticating users, accepting jobs, and performing resource management and job scheduling (the last two services comprise the *batch system*). The Worker Nodes are usually powerful machines in terms of processing power and memory capacity, and are responsible for executing jobs arriving at the site, as dictated by the batch system on the Grid Gate. If a job successfully completes execution, the result is then sent back to the Resource Broker and the user is able to access it from there using the User Interface. A UML diagram depicting the life cycle of a typical Grid job can be seen in Figure 2.

During job scheduling and execution, if any input files are necessary, they are either sent by the user during submission (included in the *input sandbox*), or they are already resident on a Storage Element (SE) and the user needs only to specify their location. This brings us to the central Data Management services: the Replica Catalog holds information about the location of various replicas of a file held at the Storage Elements of various sites, and the File Transfer Service is responsible for replicating files across different Storage Elements that are close to Computing Elements, as needed by various jobs.

In general, the *output sandbox* contains the result of a job after it has run on a CE, and contains a set of files that were specified by the user (e.g. a file that contains what would be the output of the console if a job was running on the user's computer). The entire set of output files from a completed job can either be transferred onto the RB (as part of the *output sandbox*) that the user will collect using the User Interface. Alternatively, the output files can be saved onto a Storage Element and registered with the Replica Catalog so the user can access them in the future (most probably these will be very large files of intermediate results that will serve as input to another job).

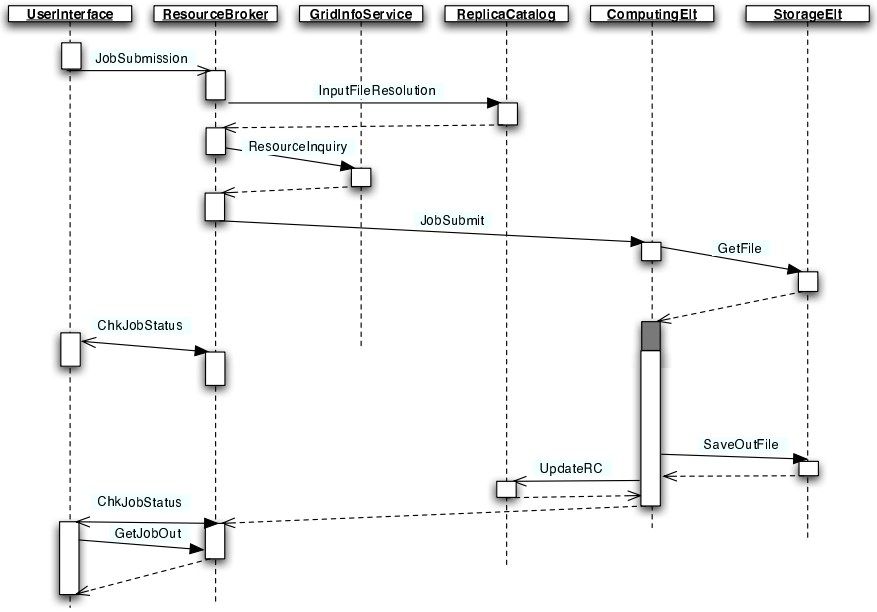

Fig. 2. Life-cycle of a typical Grid job

For more efficient project management, EGEE is divided into different federations/regions, and into each such federation resides a Regional Operations Centre (ROC) that is responsible for supporting and monitoring a set of EGEE-participating Grid sites, the Resource Centres (RCs). Division into federations is typically dictated by geographic proximity; as an example, the South East Europe (SEE) federation has a Regional Operations Centre based in Greece and several production sites (Resource Centers or RCs) in Bulgaria, Cyprus, Greece, Israel, Romania, Serbia, and Turkey. Apart from ROCs and RCs there is also the EGEE-wide Grid Operations Centre (GOC), responsible for coordinating and monitoring the operation of the Grid infrastructure, and a total of four Core Infrastructure Centres (CICs) which provide monitoring and operational troubleshooting services, acting as second-level support to ROCs. It is also worth mentioning that usually there is one Certification Authority (CA) for every participating country (any country that contributes resources to the project). The CA is responsible for issuing X.509 certificates for grid users, hosts, and services. There are also optional Registration Authorities (RAs) at each site so the CA can delegate some of its management functions [15].

3. Failures in EGEE

In order to realize the full potential of a large-scale Grid infrastructure such as EGEE, the infrastructure needs to be made *dependable*. As a measure of dependability we use the ratio of successfully fulfilled job requests over the total number of jobs submitted to the resource brokers of the infrastructure. Two large scale

computational experiments (the FlexX and Autodock data challenges) conducted by the WISDOM [14] project over EGEE in August 2005, showed that only the 65% of submitted jobs executed successfully. Additionally, in a recent nine-month long characterization study based on South-Eastern-Europe resource brokers, we found that only 48% of the submitted jobs completed successfully [19]. Consequently, the dependability of large-scale Grids needs to be improved substantially. Detecting and managing failures is an important step toward this goal. Currently, this is an extremely complex task that relies on ad-hoc monitoring and user intervention. The main components of the Grid architecture where errors that lead to job failures may occur, are presented below:

- The Resource Broker, the Grid node that is used to find an appropriate set of resources (at a Grid site) to execute the job. This node holds the user's input sandbox when the job has been submitted, and also the output sandbox after the job has terminated, until the user retrieves this output (or until the sandbox expires, depending on how the system was configured). A failure at this point can result in delays for users to retrieve job status information or their job output, or even corruption or permanent loss of job output.
- A part of the Computing Element of the site selected for job execution. The Computing Element consists of the Grid Gate node - often called CE, meaning 'CE head' - and the Worker Nodes, which are also CE nodes. Points of failure here are the Grid Gate and the specific Worker Nodes that have been allocated for the job. It is worth mentioning here that Grid Gate unrecoverable failures are rare and we have not witnessed any on our site during the four-year operation of EGEE, so far; on the occasions that the Grid Gate crashed or had to be restarted due to heavy load or abnormal CPU or hard drive temperatures, no job was lost as we noted upon restarting the machine (it recovered all job information from the WNs and started new *job manager* processes for each job). In contrast, WN failures are normally unrecoverable and resubmission is needed.
- The Storage Element (SE) holding input files that are necessary for the job is another point of failure, in the cases that the particular SE crashes, has no network access, or its filesystem is corrupted. In such an event, if there are no replicas (copies) of the necessary files on different SEs and the user has no local copies to upload, we can talk about an unrecoverable failure; however it is usually the case that if a file is only on one SE, it has probably been generated by a job and the user can resubmit that job to generate it again. Still, this wastes CPU and end-user time.
- The Information Index (II), which collects and publishes the status of Grid resources. The information published by the II is used and updated by other subsystems, like the WMS and the CE. Any failure of the Information Index results to a collapse of the RB and to a serious malfunctioning of the overall Grid infrastructure. Furthermore, a failure of the II to update its cached information on time may result to suboptimal decisions taken by other Grid components and eventually to the failure of jobs due to mismatches between job requirements and allocated resource status.
- The underlying network: any failures in the network infrastructure (links,

services, etc) that interconnects Grid sites and services may result to disconnections between different Grid components and, thus, to failures of the Grid infrastructure.

The factors that can cause a failure in the aforementioned components and a Grid-job disruption are the following:

- **Hardware faults:** if the job is running on the specified machine at the time of an *unrecoverable* hardware error, e.g. a hard drive burns (most common), RAM or motherboard failures, power supply failure, etc.
- **O/S misconfiguration:** this relates mainly to operating system services that are not properly configured. One common example is implementing firewall changes on a site. This can lead to closing ports that are needed for site inter-node communication, or blocking site hosts from communicating with each other altogether. The Grid Gate may lose communication with a worker node (WN) running a user job, or a WN may lose communication with the site Storage Element (or with another site's Storage Element) and be unable to make necessary data transfers. A closely related issue is the inability to run MPI [8] jobs when 'inter-worker-node' communication is blocked.
- **Network access disruption/misconfiguration:** a factor that can lead to job failure (or more accurately in this case, leading to CPU time being wasted), is a site losing Internet access. The firewall example mentioned above also applies to network misconfiguration, if the changes are implemented on the network 'perimetric' firewall. A user waiting for too long to retrieve the job output may decide to resubmit the job, rendering the previous one useless when the site recovers from network access failure, if we assume that the previous job was still running while the network was down.
- **Security breaches/attacks:** Computing Element, Storage Element or Resource Broker takeover by an unauthorized user (commonly known as a 'cracker') can result to malicious acts like corruption of job data, job termination, sandbox deletion etc. Such attacks are usually related to security holes of the operating system and Grid middleware, weak root passwords and inappropriate firewall configuration. Denial of Service (DoS) attacks may also disrupt job completion or temporarily prevent access to job output by cutting a site off the Resource Broker that has delegated the job and is waiting for the output. Note that discussing actual security incidents related to the EGEE infrastructure is not possible under most circumstances, since this could provide potential attackers with useful information.
- **Middleware misconfiguration:** attempts to correct problems or perform updates on a Grid site can lead to job failure or a more general service disruption. This relates to Grid site administrator errors, as well as to bugs in new releases of the middleware that introduce unwanted configuration. According to [17], a large number of service disruption occurrences is the direct result of a regular performance or security software upgrade that leads to configuration errors. Some examples of misconfiguration: setting too short a wallclock time for a job queue and as a result jobs die before completion,* publishing wrong

*this implies altering the queue wallclock time while jobs are running

resource data and matchmaking results in accepting a job while no compatible resources exist to satisfy it, causing bandwidth loss and adding overhead to the overall time needed for serving the user; killing the wrong job by issuing a scheduler or resource manager command (as root on the Grid Gate node).

- **Middleware bugs:** Grid job failures can result from bugs in middleware code; for instance, failures relating to the grid Workload Management System (WMS), including the components residing on the Resource Broker and the submitting User Interface nodes, observed in [12]. Further information can be given on this particular type of failures once the appropriate experiments are conducted on the EGEE grid infrastructure and the aborted jobs are analysed.
- **User mistakes:** such failures can result from (a) JDL file problems, for example the user may include an inaccurate specification of job requirements that will result in the job failing to start; (b) user software can cause errors during job execution leading to the job being terminated abnormally; and (c) problems with user certificate proxies attached to the job, most commonly the absence of a valid proxy during submission, as well as the expiration of an originally valid proxy while the job is running. All the cases mentioned here are under user control and have nothing to do with the malfunctioning of other components of the system, so corrective action can only be assumed on part of the user, and not by any form of automatic job resubmission mechanisms.

4. Error information sources for EGEE

The following main sources can be used to retrieve information about errors on the EGEE testbed:

(a) Site Availability Monitoring (SAM) report web site (formerly known as Site Functional Tests (SFTs)).
(b) Grid Statistics (GStat) web site.
(c) GGUS and EGEE-SEE ticketing systems.
(d) CIC broadcasts and GOC entries for site downtime.
(e) Machine logs, diagnostic commands output, and databases.

These sources are analysed below, while at the end of this section we make an assessment of the usefulness of each one of these sources, based on the accuracy and completeness of the error information provided.

4.1. Analysis of error information sources

A. Site Availability Monitoring (SAM) report web site. EGEE maintains a central "reporting web site" [9] (restricted certificate-based access) for publishing test-job results for all sites of the infrastructure, primarily serving Grid managers and administrators. From there, authenticated users can further access detailed reports for each site that show the last few entries of the SAM tests. SAM pages show the results of tests performed automatically, approximately every 1 to 3 hours, and the results of extra tests submitted by the administrators of Resource Centres (RCs) or responsible Regional Operations Centre (ROC) managers and administrators.

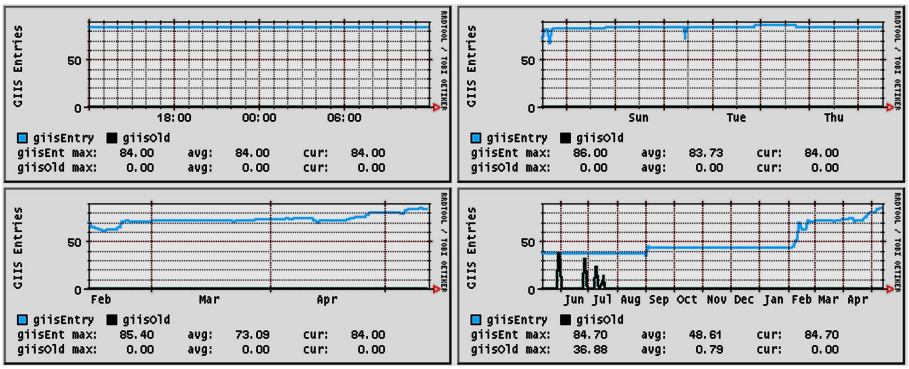

Fig. 3. GStat GIIS entries at site CY01

Test jobs are short jobs designed to check the health of the various Grid components of a site. This testing is done using the DTEAM Virtual Organisation, which exists mainly for running such internal tests on the entire infrastructure. It is worth noting here that DTEAM jobs are typically short, around 10 minutes of CPU time, unlike production-VO jobs that usually take several hours to complete. A SAM test probe consists of several subtests, checking Grid aspects such as the operational status of the Workload Management System of a site, the middleware version and the version of the Certification Authority RPMs (Linux software packages) installed.

B. Grid Statistics (GStat) web site. GStat is an application that monitors the "health" of the Grid Information System [3]. From the main GStat web page[†] Grid administrators can navigate to retrieve information collected for every EGEE Resource Center. The most interesting point there is the graphs showing error (alert) levels and various other metrics, usually going as far back as the last 12 months. From these graphs one can examine the stability of a Grid site, and possibly how long an error lasted.

An example of such graphs is given in Figure 3, which shows the number of local Grid Index Information Server (GIIS) entries that reveal site resources (hardware, services, supported software environments, policies, etc) for site "CY01." A site's GIIS normally runs on the Grid Gate and collects information about all resources present at the site [18]. GIIS entries are requested by the GStat server by running an LDAP [6] search command every few minutes; the data returned to GStat is the reply from the GIIS of the corresponding Grid Gate.

The data in Figure 3 consists of a number of 'normal' (up-to-date) entries indicated with the light-coloured line. The number of entries found varies from time to time due to the dynamic nature of the Grid (more specifically resulting from site configuration changes and changes of the software environment installed by the various VOs on the site). This means the number of normal entries can fluctuate and the site's information system could still be considered error-free and up to date; on some occasions however, the entries abruptly drop to zero (or quite lower than the current value), perhaps due to some network fault that causes timeouts or even

[†]http://goc.grid.sinica.edu.tw/gstat/

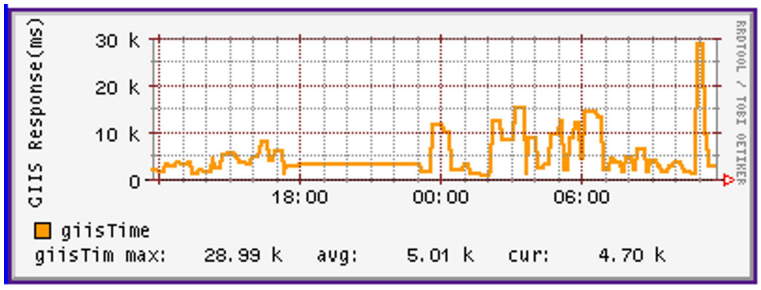

Fig. 4. GStat GIIS response time for site CY01

disconnections or a failure of the GIIS daemons running on the Grid Gate. For example, in the top-right graph of Figure 3, the two occurrences of a sudden fall followed by the immediate sudden rise are indications of such errors. In the bottom graphs, the timeline is even wider, 3 months for the bottom-left and a year for the bottom-right, so traces of such errors disappear completely.

The other type of GIIS entries found on the same graph (Figure 3) are so-called 'old' entries, meaning the information system of the site may not be up to date (the timestamp is older than 10 minutes). Such entries are shown on the graphs with the darker line. For a site to pass testing, old entries must not exist, and the darker line should be at zero; an example of a problem with old entries can be seen on the bottom-right graph (between May and July), during a period in which the site suffered major network problems, indicated by the rise of the darker line above zero.

Other points of interest in GStat pages are total and per-VO CPU and job statistics, storage space reporting, as well as estimated and actual response time for each supported VO. All this information is given in the form of graphs except the latest values which are given as numbers.

It is also worth mentioning here the SmokePing network latency monitoring tool [10], which provides network monitoring metrics for EGEE sites. These metrics give additional insight to site administrators and they are particularly useful when combined with GStat measurements or SAM results, in order to narrow down the set of components that may be responsible for a failure. For example, if GStat shows increased response time for GIIS as in Figure 4, the SmokePing graph can indicate whether this is a general problem with the site's network. As shown in Figure 5, this was indeed the case: notice that the network latency shows an order of magnitude increase during the same interval that the GIIS response time displayed a similar increase. Suppose on the other hand that GIIS showed high response time (above the usual range of 2-15 s) but SmokePing graphs showed normal latency (usually around 40 ms)[‡] in such a case, the fault origin would most probably lie on the CE, due to heavy load, abnormal CPU or hard drive temperature etc.

C. GGUS ticketing systems. The third error information source under evaluation consists of the Global Grid User Support (GGUS) and 2nd-level ticketing systems for each federation of the project. The ticketing systems in EGEE are used

[‡]from our own observations these are the normal values during error-free periods

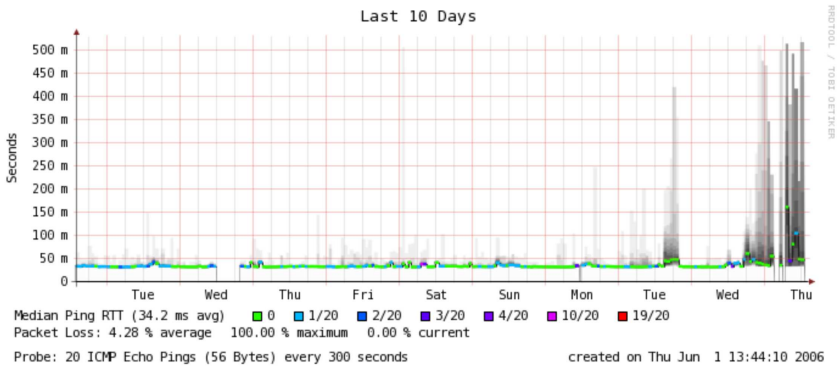

Fig. 5. SmokePing network latency for site CY01

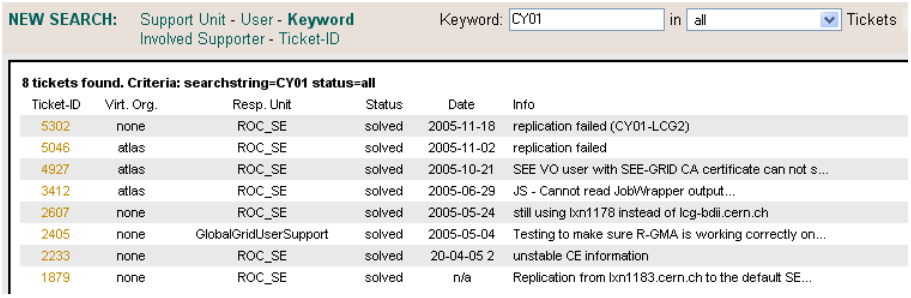

Fig. 6. GGUS Search Interface – Searching for tickets by keyword

in other organisations to efficiently manage tasks and requests. A ticket corresponds to a task-request and has various attributes such as the name and e-mail of the ticket submitter (initiator), the name of the responsible federation, the name and e-mail of the person who is assigned to the task, the ticket status (open, pending, solved, etc), and a log that shows the reason for opening the ticket, the work done, and how the problem was solved. To give an example, all tickets that were created for site CY01 from the beginning of the EGEE project in April 2004 up to the end of January 2006 are shown in Figure 6.

As far as grid operational support is concerned, the ticketing systems are mainly used to report component failures as well as needed updates for sites. GGUS tickets are typically opened because of an error that appears in the SAM (Site Availability Monitoring) reports or the GStat monitoring website; such tickets are opened by on-duty Core Infrastructure Centre (CIC) personnel. Once a GGUS ticket is opened, it is also visible to the affected federation's ticketing system, and intermediate updates are done there, but everything is also visible in the GGUS system, including the solution, the full log, and the ticket closing time and date. For this reason, a combination of both global and regional helpdesks is not necessary to make more sense of this type of error information, i.e. we only need to access the GGUS entries. Note that federation-level tickets can be opened also, and in such cases GGUS has no corresponding entries, but these tickets mostly relate to needed updates (and

not failures) that each federation has chosen to handle internally.

D. CIC broadcasts and GOC entries for site downtime. Site managers are required to broadcast information related to site downtime events through the Core Infrastructure Centre (CIC) web site; this information is subsequently e-mailed to all affected parties. CIC e-mails often contain information related to the error that caused the site manager to set the site in maintenance mode; at other times, downtime events are associated with performance or security upgrades and are not related to errors. Frequently the downtime announcements follow a series of SAM failures and a resulting ticket prompting the site to fix the errors; at other times the administrator declares the site down before the operations support has the time to open a ticket.

Site managers must also declare downtime in the Grid Operations Centre (GOC) website, in order to place the site's status in *maintenance mode* instead of *production*. Such entries are typically one short phrase, e.g. "CE hard drive burned," and also contain the start and end dates and times of the downtime event. As it was previously mentioned, the entire list of downtime events is visible from the GStat website.

E. Machine logs and diagnostic commands' output. The last category of error information sources used in EGEE consists of data found on the nodes of a Grid site:

(a) The machine logs, such as messages, and /var/log/globus-gatekeeper.log, found in /var/log/. Information stored inside these logs is usually produced by Operating System or middleware-specific sensors and, in some cases, is made available through distributed monitoring systems like GridICE [4].
(b) The output of various diagnostic commands executed on the machines that are involved in the error (while the error persists), such as ps aux, diagnose -j, checkjob -v <jobID>; and
(c) The Logging and Book-keeping Service (LBS) database records found on the Resource Broker (RB), which can reveal detailed error information spanning many sites of many different countries, usually an entire region.

4.2. Case studies

In this section we present some of the more interesting case studies involving error detection, analysis and correction. These studies were conducted between December 2005 and February 2006 on the University of Cyprus EGEE Grid site ("CY01"), in parallel to standard maintenance operations. The analysis was aided by diagnostic commands and log information, and the output of the diagnostic commands was recorded while the failures persisted.

Case study 1: DTEAM VO jobs queued indefinitely. As mentioned in section 2, the DTEAM VO is used for testing the EGEE infrastructure. Standard test jobs are automatically submitted every three hours to all EGEE sites, and the results are published on the website of the Site Availability Monitoring system (SAM). Other DTEAM jobs can also be submitted manually by site administrators, for running non-standard tests on the infrastructure.

At one point, there was a series of DTEAM jobs queued on site "CY01" that for some reason (unknown at the time) failed to start. This was a mixture of SAM-related jobs and DTEAM jobs coming from other sites of the federation that were testing a Grid service. By the time this was noticed by "CY01" site administrators, the number of jobs had reached 30 (the normal is usually 1 to 2 such jobs), and they were all in status 'queued', while there were enough free resources to execute 4 of them immediately. This caused several SAM entries to fail.

This problem persisted for several days, and we had to deal with it at first by manually forcing the queued jobs to run on idle processors. It was then discovered that the problem was caused by an erroneous job scheduler and resource manager configuration (site administrators' responsibility). These components are somewhat complicated, and their configuration non-intuitive, especially in the case of Maui [7,21], the middleware component responsible for handling job scheduling on a large number of EGEE sites. The need for reconfiguring Maui and the underlying resource manager (Torque [13]) became evident, but this involved more than a few hours of work, so we had to make a quick workaround to fix the problem, mimicking our actions of manually starting queued jobs: this was a fairly simple script that read the output of the job queue every 4 hours, and detected which queued jobs belonged to DTEAM; the script was then forcing jobs to start execution (whenever possible, based on the free resources), while logging the output of the force-run command.

After a few days of tuning the configuration of Maui and Torque, the problem rarely appeared; when it did, the workaround handled it successfully. DTEAM jobs are still likely to be queued (not indefinitely but for several hours) for various other reasons; by monitoring our site over an extended period of time, we observed that DTEAM VO members may at any point submit a large number of jobs on the site, and the result is that the job scheduler avoids starting some of them as a result of the fair share policy implemented (i.e. the 'maximum running jobs' limit set for testjobs is exceeded and Maui does not allow more DTEAM jobs to run before others terminate).

To sum up, the cause of the problem here was the lack of proper resource manager and job scheduler configuration, although it can be reduced to a more general problem with abusing the DTEAM VO and using up the few available CPU slots provided by the sites for SAM tests. The symptoms were treated first due to the urgency of the matter, while the subsequent fine-tuning of the resource manager and the job scheduler configuration addressed the root cause of the problem. The primary tool used to identify the problem was the Site Availability Monitoring report website, and the tool that lead to understanding the problem was *qstat*, which is part of the middleware diagnostic commands. While the actual tool for resolving the problem permanently is part of the middleware (i.e. configuring maui properly), the temporary patch that was applied by writing a small bash script belongs in the UNIX toolset and it is not part of the middleware. Note that patching things up in this manner is not uncommon practice for site administrators, especially for problems where there is no other solution available.

Case study 2: Active Worker Node dies. In this case study, a Worker Node crashed while a job was running on it, causing the job to be completely lost and also creating a second problem with resource allocation. In the following output

obtained from qstat,[§] notice production job of the LHCb experiment [11], with ID 74896.ce101. It appears to be running (Status [S] = Running [R]).

```
[root@ce101 root]# qstat
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
73933.ce101        STDIN            atlas004               0 Q atlas
74896.ce101        STDIN            lhcb002        00:33:58 R lhcb
```

However, the output of diagnose -j (Maui job scheduler command) shows a problem with this job, which is demonstrated in the log excerpt below:

```
[root@ce101 root]# diagnose -j
...
74896               Running DEF    1 DEF  3:00:00:00 1    1  lhcb002     lhcb
...
WARNING:  active job '74896' has inactive node wn107.grid.ucy.ac.cy
allocated for 1:18:17:03 (node state: 'Down')
```

After checking to see what was the problem with worker node wn107, we could neither connect to the machine remotely nor ping; the machine was also inaccessible from its console. The WN had crashed due to hard drive overheating. After restarting the failed node, the job was exiting (Status = E) from the queue but this state persisted for several minutes. This can be seen below from the new output of qstat:

```
[root@ce101 root]# qstat
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
73933.ce101        STDIN            atlas004               0 Q atlas
74896.ce101        STDIN            lhcb002        00:33:58 E lhcb
```

The output of diagnose -j erroneously showed that the job was running normally. The only difference from the previous message is that the warning on 'wn107 being down' was no longer present, which means that the job scheduler was updated with the information that the WN was started, and the server daemon (pbs_server) of the resource manager on the CE could connect to the torque client (pbs_mom) on the restarted WN. This job had to be killed manually, since the data from it being executed on wn107 had been lost when the machine died, and it was certain that the job could not recover. The new output of diagnose -j (after restarting the failed WN) can be seen below:

```
[root@ce101 root]# diagnose -j
...
74896               Running DEF    1 DEF  3:00:00:00 1    1  lhcb002     lhcb
-  1:18:33:49   [NONE] [NONE] [NONE]    >=0    >=0    NC0    [lhcb:1] [NONE]
```

Another related problem was that the worker node that had crashed, was later reserved and could not be utilized for a fresh job, despite the fact that its CPUs were idle. This can be seen from the output of other Maui commands, such as showres -n and checkjob <jobID>.

As it turned out, the job stayed in the queue with 'exiting' status, despite the attempts to delete it (qdel), suspend it (mjobctl -s), and similar modifications with Torque and Maui commands. All such attempts failed because the job was at a state that could not accept modifications. Next, the reservation that was made on wn107 was removed manually using Maui command releaseres ¡jobID¿, so at least the node was free to serve another job.

The job was later removed from the queue (after spending more than several hours in 'exiting' status, even persisting after a restart following a middleware upgrade) by manually deleting the resource manager job-specific files from the Grid Gate, and restarting the resource manager.

[§]command of PBS Torque resource manager used to show the status of batch jobs

To sum up case study 2, the problem originated due to a middleware bug that did not allow the job scheduler to 'understand' that one of the worker nodes had crashed and a job was lost, so manual modifications by the site administrator were necessary in order to clear the failed job and allow the restarted worker node to be utilised by new jobs. The primary tool used to identify the problem was part of the middleware diagnostic commands (diagnose -j). The tools that lead to understanding the problem were also various middleware diagnostic commands related to the Local Resource Management System (LRMS) and the job scheduler, as were the tools for providing the actual (manual) solution. Note here that the system administrator happened to execute this particular command that resulted in detecting the problem, and it was not through an alert created by a monitoring tool that the detection of the problem was made. A system like Nagios would have been useful for issuing such an alert, i.e. that host wn107 had crashed. This alert could have been sent through e-mail or sms, and the actions described above would have been assumed earlier.

4.3. Assessment of sources

Based on the analysis and the case studies presented above, we can derive the following assessment on the usefulness of individual error-information sources.

Site Availability and GStat Monitoring: From our experience, the SAM reports are usually accurate in indicating Grid site problems. The only drawback is that production jobs run for much longer than test jobs, and this may cause some errors to escape the SAM testing; also, the frequency of the SAMs may not be as high as needed to catch all errors. For this reason we can also combine some monitoring information from GStat, but this is not easy to do automatically because the graphs are in image format and the data used to generate the graphs are not readily accessible.

Furthermore, SAM relies on end-to-end tests that do not always help in identifying the root causes of observed Grid-component failures. As we saw earlier, the identification of the causes of failures requires the examination of log files and/or the invocation of diagnostic commands.

Ticketing systems: By reading the tickets posted by end-users and Grid administrators, we can find out notifications of problems that arise in EGEE, along with human-produced commentaries on these problems. It is often useful to combine information extracted from tickets (ticket timestamp, problem category, etc.) with error-related information retrieved from SAM tests and logging systems in order to detect the root causes of errors.

GOC and CIC downtime: While easy to gather and easy to separate between "downtime due to errors" and "downtime due to standard maintenance tasks" without the need to automate the process (such entries are infrequent), these sources present important drawbacks: the most important one is that the site manager or administrator publishing this information may be covering up for other types of failures. Furthermore, some downtime may not be announced due to negligence or lack of motivation, since more downtime will be accounted for that site (on some occasions, short failures may pass unnoticed). This source is possibly both inaccurate and incomplete.

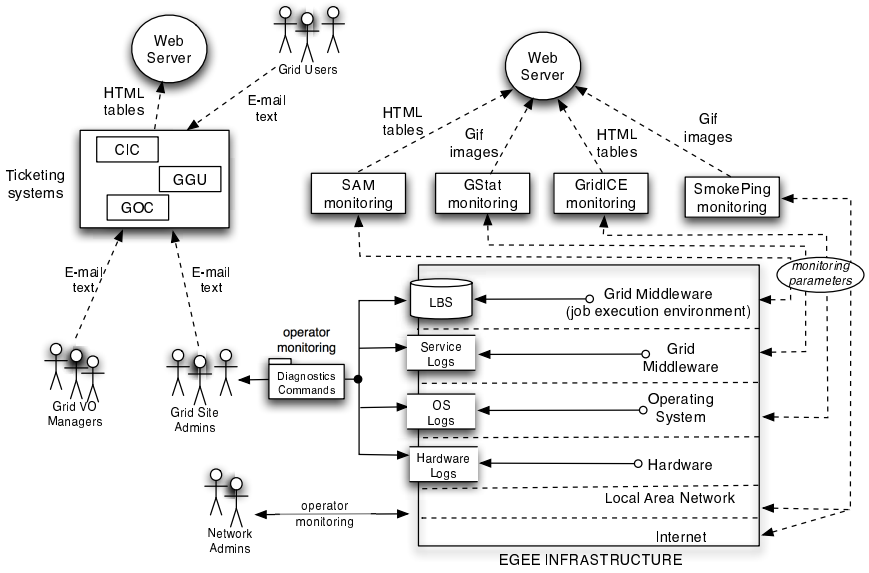

Fig. 7. Failure Management in EGEE.

Machine logs, diagnostic commands output, and databases: The machine logs and the LBS database do not rely on human intervention for their production, and we can therefore consider them the most accurate and complete error information sources from the ones examined here. Processing and integrating these logs requires extra work since they are in non-standard formats. On the other hand, obtaining diagnostic command output of real value is also tricky, since this will only be of use if it is obtained at the right time, i.e. while the error persists and perhaps even before a subsequent change of the machine state that can hide the initial error information.

5. Towards a Failure Management Infrastructure

The capability to manage failures in a large-scale Grid infrastructure requires the establishment of tools that support the discovery of failures and the detection of their causes. Discovery and detection are both very important for administrators and users: administrators can employ such tools to "debug" the infrastructure and to reduce the mean time to repair (MTTR) between failures. Grid users can take advantage of failure management systems in the context of monitoring and debugging their jobs.

Failure detection and management in EGEE relies on a large number of system attributes, which are monitored or measured on a continuous basis by EGEE's monitoring systems discussed in the previous section. These monitored attributes represent the status of components belonging to the different layers of the infrastructure (hardware, middleware, services). They are stored inside the corresponding monitoring systems, although some of them are also kept in log files and databases

on various Grid nodes. EGEE's monitoring systems publish on the Web a selection of the monitored attributes, using different non-standard formats (HTML tables, images, etc). Additional access to monitored attributes can be achieved with the help of diagnostic libraries, which are typically available to system administrators with special access privileges. Higher-level failure-related information can also be registered by Grid users and administrators through a number of existing ticketing systems; this information is also published on the Web, usually in free-text format. An overview of the failure management mechanisms that are currently in place in EGEE, is shown in Figure 7.

Following the discussion in Section , it is evident that existing error-information sources and monitoring systems can be used by Grid users and system administrators to identify Grid components or services that are failing to operate. Nevertheless, individual sources do not support the investigation of the root causes behind observed failures due to the following reasons:

- Typically, such an investigation requires the integration of data from different error-information sources. However, the integration of error-related data published through the Web is quite difficult and costly due to the non-standard encodings adopted by the different monitoring systems, which render the different data sets syntactically and semantically incompatible. Furthermore, most of the monitoring systems do not provide standardized interfaces and protocols that could be used to export their data in raw formats amenable to integration and further processing.
- Due to the scale and complexity of Grid infrastructures like EGEE, the datasets collected by their monitoring systems are large and complex. Consequently, the automatic identification of error conditions and component failures requires the implementation of advanced data management and mining techniques, which are beyond the scope and the capabilities of existing monitoring systems.
- The information that is collected and published by EGEE's monitoring systems normally represents the status of Grid resources and not the factors that cause the failures. The identification of these factors often calls for extensive experimentation involving the invocation of diagnostic tools and the application of expert knowledge. Automating these processes requires the elicitation of prior knowledge and its representation through artificial intelligence techniques.
- The users of a failure management system will benefit from feedback information and guidance that is translated to their individual context. For example, a Grid end-user may wish to find out the exact step where a failure occured in the life-cycle of a failed job, in the context of Figure 2. On the other hand, a Grid administrator would like to know which hardware or software component of his infrastructure is failing and why, in the context of Figure 1.

With these requirements in mind, we propose an architecture for a Failure Management System of EGEE, which we depict in Figure 8. The proposed system contains a set of *wrappers* that extract on a continuous basis information from various error-information sources and monitoring systems of the EGEE infrastructure. This

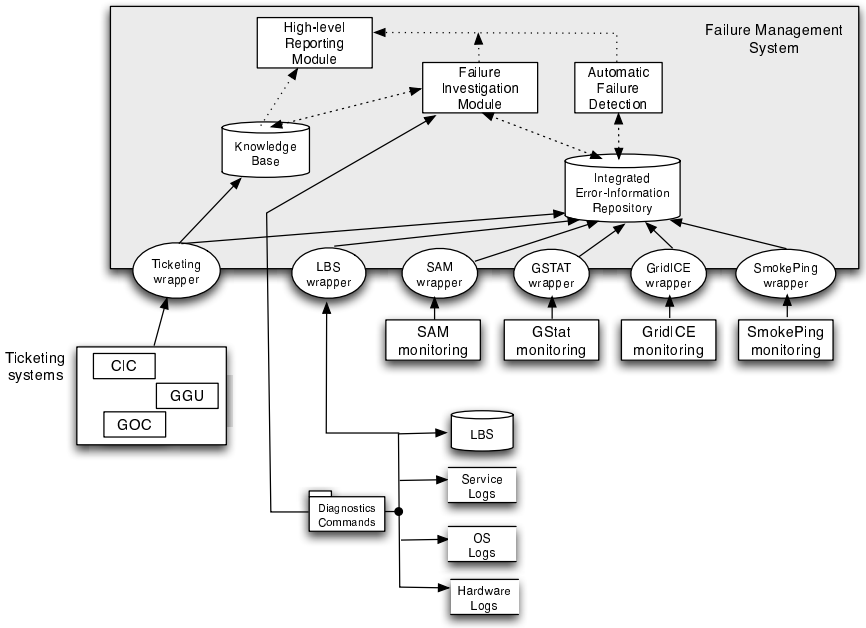

Fig. 8. Failure Management System Architecture.

information is stored in a common *Integrated Error-Information Repository*, which provides a simple API for exporting selected data-sets. An *Automatic Failure Detection* module applies simple algorithms and models to detect monitored-attribute values signaling the occurrence of errors in the infrastructure. The *Failure Investigation Module* uses various expert rules to pinpoint the possible causes of errors and to suggest the invocation of diagnostic commands that may help the user to further investigate an incident and to identify its possible causes. To this end, the Failure Investigation Module uses also a *Knowledge Base*, where information about prior failure cases is stored. Finally, a *High-level Reporting Module* translates the information about identified failures and feedback provided by the system to the context of the user of the Failure Management System.

6. Conclusions

Detecting and managing failures is an important step toward the goal of a dependable Grid. However, this is currently an extremely complex task due to the complexity, the scale, and the multi-institutional span of Grid infrastructures. In this paper, we examined the problem of failure detection and management in the context of EGEE, the largest Grid infrastructure in operation world-wide. We identified the sources that provide information about errors on the EGEE Computing Grid, and assessed these sources in terms of their usefulness, accuracy and completeness of the error information provided. Moreover, we presented and analyzed the error types that can lead to Grid job failure. We presented in detail two case

studies of Grid errors by describing the problem symptoms, the root cause of the failure, and the troubleshooting process that was used to resolve the problem.

The experiences described in this paper show that manual failure management in large-scale infrastructures such as EGEE is a tedious and cumbersome process. Furthermore, that current middleware systems do not provide adequate support for handling failures and for supporting Grid dependability. Therefore, we need to develop tools that will support system administrators and end-users to identify failures of Grid components and to investigate their route causes. These tools should provide a higher-level representation of failures, integrating information from the variety of error-information sources presented earlier. Furthermore, they should ease the troubleshooting process undergone by Grid system administrators by automating diagnostic and corrective functions, and helping them cope with the complexity of error-information provided by underlying monitoring systems through proper abstractions and uniform user-interfaces. Also, we need to develop systems and algorithms for processing the information collected by the various failure-information sources in order to support the automatic identification and prediction of failures, in order to improve the dependability of the Grid's operation.

Acknowledgements

This work was supported in part by the European Commission under projects EGEE (Contract IST-2003-508833) and the Network of Excellence CoreGRID (Contract IST-2002-004265) of the Sixth Framework Progamme of the European Union. The authors wish to thank Chryssis Georgiou, George Tsouloupas and Demetris Zeinalipour-Yazti for their helpful comments and suggestions, Nicolas Jacq for insights on the results of the WISDOM data challenge concerning grid reliability, as well as Fabrizio Pacini and Zdenek Salvet for clarifications on the internals of the Workload Management System of EGEE.

References

[1] Enabling Grids for E-SciencE project. http://www.eu-egee.org/.
[2] gLite Middleware. http://glite.web.cern.ch/glite/ (accessed June 2006).
[3] Grid Statistics (GStat) description. http://goc.grid.sinica.edu.tw/gstat/filter_help.html (accessed June 2006).
[4] GridICE: a distributed monitoring tool for Grid systems. http://grid.infn.it/gridice/ (accessed June 2007).
[5] LCG Middleware. http://lcg.web.cern.ch/LCG/activities/middleware.html (accessed June 2006).
[6] Lightweight Directory Access Protocol, open source implementation, website. http://www.openldap.org (accessed June 2006).
[7] Maui Administrator's Guide. http://www.clusterresources.com/products/maui/docs/mauiadmin.pdf (accessed May 2006).
[8] MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org/docs/mpi-11.ps (accessed June 2006).
[9] Site Functional Tests for EGEE sites. https://lcg-sft.cern.ch/sft/lastreport.cgi (accessed June 2006).

[10] SmokePing network latency measurement tool. http://oss.oetiker.ch/smokeping/ (accessed June 2006).
[11] The Large Hadron Collider beauty experiment, homepage. http://lhcb.web.cern.ch/lhcb/ (accessed June 2006).
[12] The WISDOM (Wide In Silico Docking On Malaria) Data Challenge, general statistics. http://wisdom.eu-egee.fr/malaria/grid_stat.php?menu_grid=general (accessed June 2006).
[13] Torque Administrator's Manual. http://www.clusterresources.com/torquedocs21/ (accessed May 2006).
[14] WISDOM: Initiative for grid-enabled drug discovery against neglected and emergent diseases. http://wisdom.eu-egee.fr/ (last accessed June 2006).
[15] Internet X.509 Public Key Infrastructure – Certificate and Certificate Revocation List (CRL) Profile. http://www.ietf.org/rfc/rfc3280.txt (accessed March 2006), 2002.
[16] Job Description Language: Attributes Specification. http://edms.cern.ch/document/590869/, May 2006.
[17] Aaron Brown. Coping with human error in IT systems. ACM Queue magazine, http://www.acmqueue.com, November 2004.
[18] Stephen Burke, Simone Campana, Antonio Delgado Peris, Flavia Donno, Patricia Mendez Lorenzo, Roberto Santinelli, and Andrea Sciaba. gLite 3.0 User Guide. https://edms.cern.ch/document/722398/, May 2006. Document Status: PRIVATE.
[19] G. DaCosta, M. D. Dikaiakos, and S. Orlando. Nine months in the life of EGEE: a look from the South. In *Proceedings of 15th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2007)*, October 2007.
[20] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3):200–222, 2001.
[21] Sophie Lemaitre, Jeff Templon, Steve Traylen, Markus Schulz, and Davide Salomoni. Maui Cookbook. http://grid-deployment.web.cern.ch/grid-deployment/documentation/Maui-Cookbook.pdf (accessed May 2006).
[22] F. Pacini. gLite Workload Management System service. https://edms.cern.ch/document/572489/, May 2006.
[23] D. Thain and M. Livny. *Grid 2: Blueprint for a New Computing Infrastructure*, chapter 19: Building Reliable Clients and Services. Elsevier, Morgan Kaufmann, 2nd edition, 2004.
[24] M. Xu, Z. Hu, W. Long, and W. Liu. *Grid 2: Blueprint for a New Computing Infrastructure*, chapter 14: Service Virtualization: Infrastructure and Applications. Elsevier, Morgan Kaufmann, 2nd edition, 2004.