On the Optimization of GLite-Based Job Submission

Giuseppe Misurelli\textsuperscript{1}, Francesco Palmieri\textsuperscript{2}, Silvio Pardi\textsuperscript{3,4} and Paolo Veronesi\textsuperscript{1}

\textsuperscript{1} INFN CNAF, Viale Berti Pichat 6/2 - 40127 Bologna, Italy
\textsuperscript{2} DII, Seconda Università degli studi di Napoli, Via Roma, 29 – 81031, Aversa (CE), Italy
\textsuperscript{3} INFN Sezione di Napoli, Via Cinthia, 5 – 80126, Napoli, Italy
\textsuperscript{4} INDAM -GNSC Via Madonna del Piano, 10 I-50019 Sesto Fiorentino (FI), Italy
fpalmier@unina.it, spardi@na.infn.it, {giuseppe.misurelli,paolo.veronesi}@cnaf.infn.it

Abstract. A Grid is a very dynamic, complex and heterogeneous system, whose reliability can be adversely conditioned by several different factors such as communications and hardware faults, middleware bugs or wrong configurations due to human errors. As the infrastructure scales, spanning a large number of sites, each hosting hundreds or thousands of hosts/resources, the occurrence of runtime faults following job submission becomes a very frequent and phenomenon. Therefore, fault avoidance becomes a fundamental aim in modern Grids since the dependability of individual resources spread upon widely distributed computing infrastructures and often used outside of their native organizational boundaries, cannot be guaranteed in any systematic way. Accordingly, we propose a simple job optimization solution based on a user-driven fault avoidance strategy. Such strategy starts from the introduction within the grid information system of several on-line service-monitoring metrics that can be used as specific hints to the workload management system for driving resource discovery operations according to a fault-free resource-scheduling plan. This solution, whose main goal is to minimize the execution time by avoiding execution failures, demonstrated to be very effective in incrementing both the user perceivable quality and the overall grid performance.

Keywords: Grid, Job Optimization, Reliability, Fault avoidance.

1. Introduction

1.1 The reliability in Grid environment

The main Grid computing aim is in supplying computing power as an utility, providing an ubiquitous on-demand service through a semi-persistent virtual environment created for solving specific tasks. The most attractive Grid feature is unquestionably the ability for a client to send out very complex and computationally expensive jobs that will be transparently an dynamically located on the Grid nodes and sites providing all the specific runtime resources needed by the involved application to be executed successfully. What is striking about the above feature is how few assumptions can be made about the resource that may be available on such a complex and widely distributed infrastructure. These resources may be widely heterogeneous; each may run a different combination of hardware, operating system and application middleware environment. Furthermore, some of them may be privately available within corporate intranets, whilst others may be publically available over the Internet. Some clusters of resources may be stored in a single location/site, whilst others may be geographically distributed. In this scenario is very likely that the jobs submitted from an unaware client will be transparently relocated for execution on remote machines upon which no trust can be placed. For example, the machines may be outside from the originating client’s organizational boundaries, or may be desktop machines or workstation sharing their computing resources during idle working time that many people have access to and are not strictly dedicated to the Grid tasks.

A runtime optimization strategy ensuring fault avoidance to the submitted jobs may therefore be very desirable in order to potentially prevent a defective, unreliable or wrongly configured node to be selected for running a job that it is not be able to complete, adversely affecting the overall success or performance of the requiring Grid applications. Starting from the above considerations we developed a simple and flexible fault avoidance solution that can be seamlessly implemented within the most common existing middleware frameworks. This solution is based on the introduction within the grid information system of several advanced on-line service status monitoring objects, associated to the individual resources. That object are continuously kept up-to-date and can be used, under the control of client applications, as

Published under licence by IOP Publishing Ltd
specific hints to the workload management system conditioning its resource discovery and selection decisions according to a scheduling plan that ensures the occurrence of no (avoidable) faults. Such objects are produced and continuously kept up to date by specific sensor modules deployed within each local site monitoring system that perform sanity checks on the resources of interest and directly interface the Grid information system catalogs by publishing and consuming the associated monitoring information. A simple prototype of the above framework has been implemented on a test Virtual Organization on the gLite based “GRISU” grid infrastructure spanning several universities and research centers in the south of Italy. The preliminary functional tests ran on it showed interesting performance improvements both in terms of response to user’s reliability and responsiveness demand and in terms of computing infrastructure throughput.

This paper is organized as follows: In section 2 we introduce the fault avoidance concept and the dynamic approach test in our work. In section 3, we describe the interaction between the Grid monitoring system and the Information Index then we introduce the use case of MPI jobs. Section 4 describes the experimental framework used to evaluate the performance of our approach and shows the results obtained in several tests performed in a real Grid environment. In section 5 we discuss related works, we finally provide summary and conclusion in section 6.

2. Fault Avoidance in grid infrastructures

Fault avoidance is a fundamental property in modern Grid infrastructures, deployed over a large number of geographically distributed sites managed by different organizations, since the dependability of individual resources and localized middleware services cannot be guaranteed. In fact, the increased number of interactions between large numbers of computing nodes and Grid services (e.g. runtime capabilities, connectivity, libraries etc.), many of which may be dynamically bound at the execution time and hence not known to the original application developers, will undoubtedly lead to an increased incidence of interaction faults. This may be due to different services supporting different protocols or capabilities, or may also be due to computing nodes that are faulty (also partially) in some way. That’s worse as Grid applications may feature dozens – if not hundreds – of services, a fault in one service may be amplified as the effects of the fault move further down the chain. Clearly, one of the key factors for handling these runtime faults is the Grid’s resource scheduling system that with its selection policies and choices could greatly reduce or exacerbate the above phenomena. Unfortunately, most of the state-of-the-art scheduling approaches have been inherited from the legacy distributed clusters. They are almost exclusively batch systems offering good levels of reliability while sacrificing utilization levels, interactivity experience and dynamic resource control and monitoring. Such trade-off is currently acceptable in production environments mostly due to the lack of stable and really usable alternatives but is no more sufficient to ensure the needed performance and user satisfaction degree as the grid technology scales and attracts new sophisticated and demanding applications.

Most of the widely used schedulers have a static view of many resource features and runtime capabilities and operate almost blindly in their selection without considering up-to-date monitoring information and online functional probes/checks. Accordingly, a really effective fault avoidance solution needs to identify in advance and categorize the most common runtime faults that may occur in a grid and specifically on its available resources and services, in order to provide clues to the underlying scheduling system on how to avoid choosing them and hence prevent as possible application faults with continuous job resubmission, with the effect of greatly improving the overall Grid performance. It is clear that the exact knowledge of the real operational status of the various services offered by the available Grid resources is the crucial information needed to support such a fault avoidance strategy. Hence, a scheduling plan ensuring that no (or minimum) faults will occur could then be made by using online operational information periodically gathered from specific Grid monitoring facilities.

3. Implementing an optimization strategy based on fault avoidance

As a reference platform for implementation we considered the gLite [1] middleware stack, deployed within the EGI (European Grid Initiative) Infrastructure and in some regional Grid Initiatives [2]. The
EGI implementation currently represents one of the largest Grid infrastructures available all over the world. In order to implement the above fault avoidance strategy within the gLite environment, we introduced some simple modifications to several middleware services belonging to the grid monitoring subsystem allowing the end-users to recognize the faulty resources and to avoid them as possible by issuing the proper requests to the Grid resource broker.

Currently all the production sites belonging to the EGI Infrastructure, are monitored through a central system named Service Availability Monitoring (SAM) [3]. SAM is a nagios-like application that periodically performs several probes/checks on Grid sites and nodes of interest, in order to verify the certification status, the data transfer capability and the interaction with some basic gLite services/roles. SAM is actually used to individuate faulty nodes/sites and raise alarms to the relative site administrator in order to fix the problem. The involved nodes and sites are excluded from the Grid and from user visibility only if the error persists beyond a specific time threshold. However, SAM doesn’t contribute directly to increase the Grid reliability as it can be perceived from the user point of view, thus new and more flexible monitoring facilities have to be introduced. We started from the consideration that the computing resources participating to a Grid usually don’t publish any online information about the runtime operational status of their service portfolio or about the possible faults occurred on it (e. g. an host certificate is expired or some crucial service is down for some reason). Moreover, due the strong interdependency among the Grid resources, we considered that the best monitoring approach is performing periodical sanity checks on the services offered on the individual resources and providing such information to the users through the Grid information system. To do that, we reengineered the SAM online service health probing architecture by introducing a new set of monitoring objects managed by a local instance of Grid Nagios, a customized version of Nagios released within the gLite middleware platform to create personalized probes.

We identified a set of typical fault scenarios that are crucial for the success of Grid jobs and data-management activities and developed the corresponding runtime functional probes within the Nagios [4] monitoring environment. These custom “sensors” are written in Perl and have similar behavior. Each nagios sensor simulates typical grid client behavior. It uses the required grid service interface to perform a specific operation to be probed on a node (e.g. submit job, transfer file). In particular, the result of each sensor check has been represented by a string containing the probed status. For example, if the probed service does not respond, the sensor returns status Unknown. If the service responds, sensor analyzes the output and if the output is correct, the returned status is Ok, otherwise it is set to Failure. Each sensor is periodically scheduled by the Site Nagios services to perform its checks and publish the corresponding results on the site-BDII catalog. New data structures needed for representing the status/operational information returned by the sensors within the Grid information systems have been developed by defining specific Job Description Language (JDL) [5] parameters, associated to job requests, to be checked by the glite Resource Manager, the Workload Management System (WMS) during the first stage of the resource brokering process. These parameters are based on Condor classified advertisements (ClassAds) and structured as attribute-value pairs published on the Grid top-level information system database. In such a way, we can easily embed some specific service operational status request into the scheduling strategy in order to improve the efficiency and the utilization of the Grid and avoiding, as possible, execution failures. ClassAds comparisons are performed in the matchmaking phase starting from JDL attributes in an evaluation environment that maps any attribute accessed to the match candidate under consideration.

In detail, the users submitting a job to the Grid, can describe in advance their service status requirements by referencing the proper monitoring metric, by using JDL expressions containing a Requirements and a Rank ClassAds expression. This drives the discovery of grid-resources that are “certified” to support the needed services. Requirements is an expression that defines the conditions to be met by the resource. It can include the check for extended attributes or specific runtime environment variables that can be used for referencing the proper service support capabilities on the involved node.
Rank specifies the preference criteria to be adopted to select one of the resource instances that satisfy the Requirements. In addition, authorization checks are also performed against the Virtual Organization attributes. This lead to the formulation of a schedule plan conditioned, if possible, from the above requirements. In the following script (figure 1) we show a JDL example with the requirements of discovering grid-resources that support the MPICH library service. Specifically, we are looking for computational resources within the “matisse” Virtual Organization that publish the VO-matisse-MONITOR-MPICH-OK variable in their runtime environment. Such variable indicates that the MPICH library is correctly configured and is fully operational so that the involved job can run, if possible on sites/nodes supporting MPICH services.

| Type        | "Job"                           |
|-------------|---------------------------------|
| JobType     | "MPICH"                         |
| CpuNumber   | 16                              |
| Executable  | "mpi-start-wrapper.sh"           |
| Arguments   | "mpi MPICH"                     |
| StdOutput   | "mpich-test.out"                |
| StdError    | "mpich-test.err"                |
| InputSandbox| {"mpi-start-wrapper.sh","mpi-hooks.sh","cpi.c"} |
| OutputSandbox| {"mpich-test.err","mpich-test.out"} |
| RetryCount  | 1                               |
| Requirements| Member("VO-matisse-MONITOR-MPICH-OK", other.GlueHostApplicationSoftwareRunTimeEnvironment) && Member("MPICH",other.GlueHostApplicationSoftwareRunTimeEnvironment) |

Figure 1. Simple JDL for MPICH test

4. Performance evaluation and analysis

In the following section, we evaluated both the performance and effectiveness of the proposed fault-avoidance framework by working on a simple proof of concept prototype implementation built on the gLite based “GRISU” grid infrastructure, the southern Italian EGEE component, consisting of more than 10,000 CPU distributed on 4 geographical regions: Puglia, Sicilia, Sardegna and Campania. The evaluation tests, ran on the test virtual organization Matisse, have been selected from the common simulation framework used by the local scientific community, in order to match a real use case. We submitted a large bunch of parallel jobs, requesting the Message Passing Interface (MPI) library services, which perform basic simulation activities by using 16 processors at a time. MPI is the most popular programming library for parallel computing available in the C, C++, and Fortran programming environments. Its diffusion within the European gLite-based Grids has greatly increased in the last years to support the emerging needs of scientific applications featuring a large degree of parallelism. The setup and configuration of MPI services on the generic worker nodes is a complex duty, especially when the site administrator has to manage different library implementation flavors, different programming environments and different underlying network fabrics. Moreover, the support of different MPI configurations is a requirement for many scientific communities and the online verification of a correctly working configuration is fundamental to avoid job aborts or abnormal behaviors.

Currently MPICH, OPENMPI and MVAPICH are the most popular MPI flavors, and whereas in the EGEE production infrastructure more than 30% of the working sites publish the support for these libraries, a consistent part of them does not pass the basic SAM configuration sanity check. In figure 2 we can observe a screenshot of the SAM web interface, where a set of sites that haven’t passed the MPI certification test are evidenced.
In our prototype implementation we modified the standard nagios sensor performing the MPICH SAM TEST sanity checks in order to publish the positive results on the Grid information system through the specific runtime variable VO-matisse-MONITOR-MPICH-OK. This value is propagated to the collective Grid information system (the top BDII in the gLite environment), so that the user can set the presence of this tag as a mandatory job requirement in the associated JDL script. To analyze the improvements on the runtime stability of the grid directly perceivable by the users we at first calculated an approximation of the worst-case job execution failure rate when no fault avoidance strategy is implemented. By querying the Top BDII services (the collective informative system) we discovered 49 resources that publish the MPICH support available within the Matisse Virtual Organization. We then verified that only 36 resources between the above ones passed the MPICH nagios test at job execution time, so that we calculated a worst-case failure rate value of 0.265. We ran two sequences of tests needed to evaluate the introduced performance improvements on a significant number of jobs:

1. Submission and analysis of 550 jobs on the Grid through the WMS without failure-avoidance management.
2. Submission and analysis of 550 jobs on the Grid through the WMS with the specific requirement for the tag VO-matisse-MONITOR-MPICH-OK on the runtime environment.

To achieve more confidence in the obtained results, each run has been repeated 10 times and the average performance metric values have been calculated. During the first test sequence, we obtained a total of 144 job faults, with a failure rate equal 0.261, that seems to be in accordance with the calculated value. So we can argue that, in absence of any specific Requirements or Ranking tag in JDL, an increasing load on the Grid produces a uniform usage of the available resources. The aim of the second test block is quantifying the effective performance improvements introduced by our fault-avoidance strategy within the grid runtime system. First, by using the standard gLite matchmaking client (glite-wms-job-list-match) we observed that the Requirements expression in the JDL script worked correctly forcing the WMS to filter out the 13 resources, which failed the nagios MPICH probe. After the complete execution of all the 550 jobs enforcing the MPI runtime status check through the JDL requirements mechanism, we detected a total of only 4 aborted jobs with a failure rate equal to 0.007, two order less than the previous case. The graph in figure 3 shows the percentage of job successfully completed when the fault-avoidance strategy is enforced, compared with the standard WMS behaviour.
5. Related work and discussion

The interest in making Grid infrastructures fault-aware has received a certain attention in literature [6], [7], [8], [9]. For example, several fault detection service architectures have been developed for grid computing systems (e.g., [10], [11], [12]. Other applications adopted some ad hoc fault-tolerance mechanisms, which cannot be reused, nor shared among them [13]. All the above approaches require the introduction of new complex services and significant modifications to several crucial components of the grid architecture (i.e. the resource broker). On the other side, our proposed approach is based on the specialized use of already available features and mechanisms with no impact on strategic middleware services. We only require the introduction of some simple, flexible and modular monitoring plugins each dedicated to specific sanity checks. These features make our fault avoidance solution implementable in a seamless and straightforward way in existing production grids. It can also be easily extended, by adding new plugins specialized for checking new services, as they are added to the grid middleware platform.

6. Conclusions

The introduction of monitoring metrics on the Grid information system that can be used to easily implement a user-driven fault avoidance strategy opens up new interesting scenarios to increase the grid stability and facilitate operations in a production infrastructure. By implementing the above strategy in the gLite environment we obtained interesting performance improvements particularly perceivable from the user’s point of view. This effect has been evidenced through a set of preliminary tests that showed the positive impact of the proposed approach in terms of number of jobs successfully completed. In future works we plan to deploy this approach in a large scale Grid and obtain further feedbacks in order to detect the best metrics to publish, to improve the data model and maximize the improvements on the infrastructure’s stability and reliability. Finally, we plan to investigate the introduction of a site reputation concept as a new metric to be used during the resource discovery process.

References

1. E. Laure, F. Hemmer, et al., Middleware for the Next Generation Grid Infrastructure. In Computing in High Energy and Nuclear Physics (CHEP), Interlaken, Switzerland, September 2004.
2. Francesco Palmieri, Francesco Palmieri begin_of_the_skype_highlighting end_of_the_skype_highlighting, Silvio Pardi: Towards a federated Metropolitan Area Grid environment: The SCoPE network-aware infrastructure. Future Generation Comp. Syst. 26(8): 1241-1256 (2010)
3. A. Duarte, P. Nyczzyk, A. Retico and D. Vicinanza. Monitoring the EGEE/WLCG grid services, Journal of Physics: Conference Series 119 (2008).
4. Nagios. The Industry Standard In Open Source Monitoring, http://www.nagios.org/
5. F. Pacini. Job Description Language HowTo, http://server11.infn.it/ workload-grid/docs/DataGrid-01-TEN-0142-0 2.pdf, (2003).
6. J. H. Abawajy and S. P. Dandamudi. Fault-tolerant grid resource management infrastructure. Journal of Neural, Parallel and Scientific Computations, 12 pp. 208–220, (2004).
7. S. Hwang and C. Kesselman. Gridworkflow. A flexible failure handling framework for the grid. In 12th International Symposium on High-Performance Distributed Computing (HPDC-12 2003), 22-24 June 2003, Seattle, WA, USA, pp. 126–137. IEEE Computer Society, (2003).
8. J. B. Weissman. Fault tolerant computing on the grid: What are my options? In HPDC ’99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing, p. 26. IEEE Computer Society, (1999).
9. J. B. Weissman. Fault-tolerant wide area parallel computation. In Proceedings of IDDPS’2000 Workshops, pp. 1214–1225, (2000).
10. J.H. Abawajy. Fault detection service architecture for grid computing systems. In Lecture Notes in Computer Science, volume 3044/2004, pp. 107 – 115. Springer-Verlag, (2004).
11. P. Stelling, I. Foster, C. Kesselman, G. von Laszewski, C.Lee. A fault detection service for wide area distributed computations. In Proc. 7th Symposium on High Performance Computing (HPDC), pages 268–278, (1998).
12. B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. A monitoring sensor management system for grid environments. In HPDC, pp. 97–104, 2000.
13. H. Soonwook. A Generic Failure Detection Service for the Grid, Ph.D. thesis, institution = "University of Southern California, (2003)."