Monitoring the EDGeS Project Infrastructure

Filipe Araujo*, David Santiago*, Diogo Ferreira†, Jorge Farinha†,
Patricio Domingues‡, Luis Moura Silva*, Etienne Urbah§, Oleg Lodygensky§,
Haiwu He¶, Attila Csaba Marosi‖, Gabor Gombas‖, Zoltan Balaton‖,
Zoltan Farkas‖, Peter Kacsuk‖

* *CISUC, Dept. of Informatics Engineering*
University of Coimbra, Portugal
E-mail: {filipius, demanuel, luis}@dei.uc.pt
† *E-mail: {defer, jfar}@student.dei.uc.pt*

‡ *School of Technology and Management*
Polytechnic Institute of Leiria, Portugal
E-mail: patricio@estg.ipleiria.pt

§ *LAL Universite Paris Sud, CNRS, IN2P3, France*
E-mail: {urbah, lodygens}@lal.in2p3.fr

¶ *INRIA, LIP, ENS Lyon, France*
E-mail: haiwu.he@inria.fr

‖ *MTA SZTAKI, Computer and Automation Research Institute of the Hungarian Academy of Sciences*
H-1528 Budapest, P.O.Box 63, Hungary
E-mail: {atisu, gombasg, balaton, zfarkas, kacsuk}@sztaki.hu

Abstract

EDGeS is an European funded Framework Program 7 project that aims to connect desktop and service grids together. While in a desktop grid, personal computers pull jobs when they are idle, in service grids there is a scheduler that pushes jobs to available resources. The work in EDGeS goes well beyond conceptual solutions to bridge these grids together: it reaches as far as actual implementation, standardization, deployment, application porting and training.

One of the work packages of this project concerns monitoring the overall EDGeS infrastructure. Currently, this infrastructure includes two types of desktop grids, BOINC and XtremWeb, the EGEE service grid, and a couple of bridges to connect them. In this paper, we describe the monitoring effort in EDGeS: our technical approaches, the goals we achieved, and the plans for future work.

1. Introduction

Interoperability of computing resources that are geographically and administratively dispersed is a goal that the scientific community is pursuing for a large number of years by now [1] and that we can still consider as incomplete. One of the deepest differences that persists, concerns access to resources attached to desktop grids (DGs) and to service grids (SGs). DGs are characterized by an (often) extremely large number of computers orchestrated by a central supervisor, glued together by some middleware like BOINC [2] or XtremWeb [3] (XW). In these DGs, workers (also known as clients) always take the initiative of requesting new jobs to the central server, usually when they are idle. One of them most distinctive features in DGs lies in the fact that computing resources belong to private users, which make them available on an entirely voluntary basis, for altruistic reasons, for fun, or for some other reason. Unlike this, service grids are frequently made of dedicated computing resources, belonging to scientific or academic institutions. Service grids include a scheduler that pushes jobs to computers available in the pool. One of the largest service grids that exist today was brought together by another EU project, called Enabling Grids for E-sciencE [4] (EGEE), which joined together as many as 300 sites from 50 countries. As a consequence of these two fundamentally different kinds of grids, there is a big impediment to make jobs (or tasks or workunits) intended for a DG to run on a SG and vice-versa.

To overcome this difficulty, Enabling Desktop Grids for e-Science [5] (EDGeS), a Framework Program 7 (FP7) European project (see the EDGeS web page at http://www.

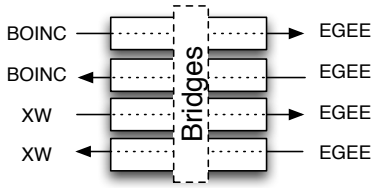

Figure 1. Bridges under development in EDGeS

edges-grid.eu/web/edges/) aims to connect both kinds of grids in a seamless way, such that jobs running in EGEE can run in the extremely large pool of resources made available by either BOINC or XW servers, while also enabling DG projects to reach EGEE. While the EDGeS team is actively solving the interoperability problem, there is more to it besides conception and implementation: the team must deploy a few different infrastructures to port and run existing end users applications, it must adopt and foster standard solutions, and it must commit on dissemination, training and consulting activities to spread the work. Many, if not most, of these activities are partially or almost completely finished by now.

One of the workpackages of EDGeS (known as Joint Research Activity 2 — JRA 2) devotes to monitoring the entire infrastructure, made of DGs, SGs and bridges. The purpose of this paper is precisely to describe the monitoring solutions in use, inside this project. While we adopted a well-known tool, named Ganglia [6], as part of our solution, the specificities of EDGeS make it difficult, if not impossible, to pick one off-the-shelf application to do the entire monitoring job. We had to create many modules and to considerably change Ganglia to fit our own needs. One interesting fact is that we manage to collect most significant data from DGs alone, as these can provide surprisingly rich information in the context of EDGeS. Still, there are many open problems that we are currently working on.

In the remainder of this paper, we present the monitoring problem, its challenges, solutions and current situation. In Section 2, we briefly introduce the EDGeS architecture. In Section 3, we outline the high-level requirements of monitoring EDGeS. In Section 4 we overview some important monitoring architectures. In Section 5, we review the monitoring solution we built. Finally, Section 6 concludes the paper and points to our future work.

2. An Overview of the EDGeS Architecture

EDGeS has two strongly related fundamental goals: i) enable job submissions originating in SGs to reach available DG nodes and ii) enable workunits of DGs to reach SG nodes. The EDGeS project should develop solutions for EGEE as a SG and for XW and BOINC as DGs. This makes for a total of four different bridging operations: BOINC→EGEE, EGEE→BOINC, XW→EGEE and EGEE→XW, as seen in Figure 1. The EDGeS team developed these bridges in JRA 1. Implementation of all these bridges, except the XW→EGEE bridge, follows the 3G Bridge Architecture developed at SZTAKI [7]. We depict the main idea of this architecture in Figure 2.

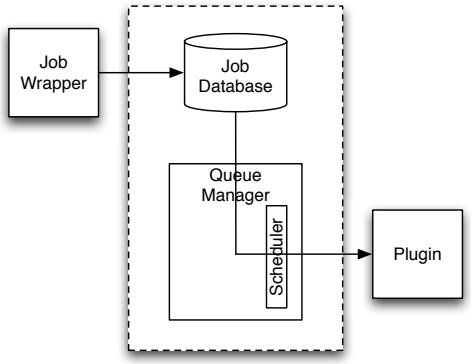

Figure 2. 3G Bridge Architecture

The 3G bridge has one single core (inside the dashed rectangle) that receives, stores and forwards all jobs in transit to the different kinds of grids. When a job enters the core, it is stored in the *job database*, while the handler, which represents the job, goes to the *queue manager*. This handler stays there until the *scheduler* dispatches the job to run. The core should receive and uniformly treat jobs from different originating grids, provided that there exists some form to wrap these jobs from that grid into the 3G bridge. Additionally, the bridge core can also submit jobs to any grid, as long as an appropriate plugin for that destination grid exists.

The job wrapper depends on the particular input grid. In the case of BOINC, the job wrapper requests workunits from the server, deals with the input file, with the executable (to be run in the other side of the bridge) and with bringing back the results to the originating bridge. If the destination grid is EGEE (thus in this example we would have the BOINC→EGEE bridge), this means to run an EGEE job and set configuration variables, like Virtual Organization (VO) name, or the MyProxy information necessary to submit jobs, like host name, port, user name and password.

One important aspect for monitoring is that a single bridge deployment can support all bridging operations. This simplifies a number of tasks, as all bridging information and, more important, configuration are centralized in a single point. In particular, to perform monitoring on the bridge, information on the job wrappers and job database suffices. From the job wrapper, we need to know which clients are in fact pumping jobs to the other side of the bridge, while from the job database, we need to do some counting over the jobs (which ones are waiting in the bridge, for instance).

The only bridge that does not use the 3G architecture is the XW→EGEE bridge. This bridge follows the gliding-in approach [8]. This solution tackles the issue of matching a pull-based grid, where workers request jobs (XW) with a push-based grid, where a scheduler sends jobs to workers (EGEE) in a different way. The approach is to wrap the XW executable and submit this executable to the SG using the standard push-based approach. This executable is known as "pilot-job". Once this executable reaches the SG, it requests tasks and sends results to the XW server, as any other volunteer worker would do in the standard DG. In simplified terms, the bridge pushes a job that pulls the work.

From the monitoring point of view, this solution is more complex as there is no central entity, like the bridge, to get configuration information from. While in BOINC, the single bridge requests all the workunits for the SG side, in XW the same requests can come from many different sources. This complicates the task of recognizing which tasks were computed by standard DG nodes, and which ones were computed by SG nodes. At the time we write this text, all the works that are intended to cross the bridge have a special indication. We use this indication to distinguish jobs that crossed the bridge, from those who did not.

3. Requirements of EDGeS Monitoring

In a first stage, we defined three main requirements for EDGeS monitoring:

- Performance monitoring. This includes counting the number of jobs/workunits executed in each DG, how many cross the bridge toward EGEE and vice-versa, success rate of jobs and workunits, computing power available in the entire platform, etc.
- Providing information to the bridge operation. In practice, this consists of publishing appropriate information of the DG into the Berkeley Database Information Index (BDII).
- Notification of problems in the bridge.

At the present time, we already tackled the first two, while the third problem is still pending. However, since the second goal is not very relevant to this paper, we focus on the performance monitoring issue.

4. A Short Review of Grid Monitoring Tools

Since many monitoring systems exist already, one of the first steps in our work was to take a decision on which monitoring tools and solutions should we base EDGeS monitoring. Here, we overview just a few. Many more exist, namely for EGEE, but in general, as we explain ahead, they are not entirely fit for our purposes.

Nagios is a system, network and application monitoring environment. Nagios uses a web-based interface to display information of its monitored resources. Nagios has the ability to monitor almost any resource, because it relies on external plugins to monitor virtually any system properties, such as load average and free disk space; the availability of important network services like HTTP, SMTP, POP3, NNTP, PING, or per host network availability and reachability. Nagios includes notification of changes through normal channels (e-mail, pager, cellphone) or via user defined mechanisms; it can define event handlers to run during service or host events for proactive problem resolution (e.g., restarting a HTTP server). It is also extensible through add-ons, for services not included in the standard package. Nagios relies on a central core, Nagios itself, together with the external special purpose plugins. The central core provides basic scheduling. When a resource needs to be checked, Nagios calls the appropriate plugin to perform the check. The plugin is a simple executable program, like a shell or perl script.

Ganglia [6] is a distributed monitoring system oriented toward high performance computing systems such as clusters and grids. One of the crucial goals of Ganglia is scalability as it aims to collect statistics from large sets of sources. These include the name of the machine, the number of CPUs, memory, etc. Objects of interest to Ganglia are nodes, clusters and grids [9]. A node is a computer of 1 to 4 CPUs usually racked. A cluster is a group of nodes and a grid is a group of clusters. However, these definitions are quite flexible and depending on the particular scenarios, the concepts of node, cluster and grid, can have different meanings. Ganglia has several possible uses, for monitoring large scale cluster, "bunch" of machines, data centers, or simply logical partitions. [9] enumerates several scenarios of use for Ganglia.

Ganglia contains three main modules: ganglia monitoring daemon (`gmond`), ganglia meta daemon (`gmetad`) and ganglia web front-end (web). `gmond` is a daemon running on every monitored machine. It collects all the system metrics configured, like CPU, memory, disk, network, etc. When run in a cluster environment (i.e., all nodes can be joined through multicast), the nodes synchronize their databases using multicast. This enables the `gmetad`, which lies above `gmond`, to collect information of all the cluster from a single node. Although we base our monitoring system in Ganglia, we must emphasize that we change the way in which `gmond` runs and we also change its role considerably.

To store and display information, Ganglia uses RRDtool [10] (Round Robin Database) for the different levels of the hierarchy: grid, cluster, host and metric trends. RRDtool can store data for different time granularities ranging from minutes to years. RRDtool can also display plots of data for the different time granularities.

One important additional component of Ganglia is `gmetric`, a command-line tool that lets applications publish application-specific metrics (as opposed to built-in metrics, which are compiled in `gmond` source code and, thus,

collected by default). We take advantage of `gmetric` to dynamically collect and submit data to Ganglia.

Another important issue for us is what kind of monitoring information can we get from the EGEE side. There is a large number of monitoring tools for EGEE, as listed in [11]. We would be particularly interested in having the precise number of times that some application was run in the EGEE side, such that we could compare this number to the number of times it reaches the DG side after crossing the bridge. Unfortunately, as far as we know, this is not possible. One thing however, that we can get, from sites like Gridview [12] and CESGA [13] is extensive information on specific jobs that ran in some VO. This will allow us to see some data concerning the EDGeS VO (though we are not using it yet).

4.1. Discussion of Monitoring Solutions

In the Service Grid→Desktop Grid bridge, the Workload Management System (WMS) treats a DG as a single Computing Element (CE). Hence, for monitoring, we could treat the DG part as another Computing Element integrated in EGEE. Unfortunately, there are a couple of inconveniences to this idea. One problem is that we still have the Desktop Grid→Service Grid bridge to monitor. Another more subtle issue is that we can get a lot of data from BOINC or XW servers, regarding execution of individual applications, but not quite the same from the EGEE side. We thus treat BOINC and XW differently from other CEs, the shortcoming being that we need to join two irreconcilable worlds.

From the monitoring point of view, DGs and SGs could not be more different. While the components of a SG obey to a strongly centralized administration, usually, DGs do not exist under the same administrative roof, as nodes in a DG belong to private unrelated users (an exception to this comes from DGs running at faculty labs, for instance). This makes it impossible to install dedicated monitoring components in a DG, as no administrator can impose such installation. Moreover, owners of private PCs might not be interested in installing additional software (this might even be very complex).

Our particular choice of Ganglia was motivated by the flexibility and simplicity of this tool. A very interesting feature of Ganglia is that it uses the RRD Tool, to store periodic-based information. Utilization of the RRD Tool in Ganglia provides a powerful way of showing individual and aggregate statistics of the parameters under observation. While we changed many things in the "normal" use of Ganglia, we still use this powerful feature, as we describe in Section 5.6.

However, Ganglia cannot solve all the problems by itself, as we cannot install `gmond` on every privately owned computer. A bulk approach, with queries on the BOINC or XtremWeb server databases directly, seems to be our only option to read data from DGs. Despite controlling each one of the users individually, the DG server does so in the normal control flow of computation. It does not impose additional messages to the clients or any change in the communication pattern, which is always initiated by the client (e.g., because client can reside behind a firewall and may be inaccessible). Hence, our approach on DGs changes nothing in the client and uses the extensive database information already in place in the servers. Although we used Ganglia as a starting point, practice showed us that the base version of Ganglia required some changes, before reaching production in our specific setting.

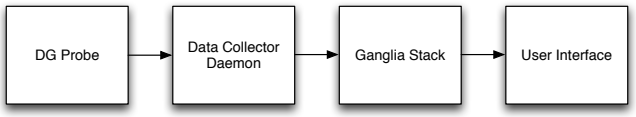

Figure 3. Architecture Blocks of EDGeS Monitoring

Furthermore, we have other unsolved challenges in front of us, like achieving a deeper integration of monitoring with EGEE, supporting restart of services or notification of failures, just to name a few. Some of these issues are easily solvable with Nagios, which can also monitor EGEE sites [14]. We are currently evaluating the possibility of using Nagios to augment our current service.

5. Monitoring Architecture

The current version of our monitoring architecture comprises the following parts:

- the BOINC/XW probe;
- the Data Collector daemon;
- the Ganglia stack;
- the User Interface.

We can further refine the organization of our architecture in a rough two-part division: one serves the purpose of collecting, and processing general data, the other supports the web view of data (although it also manages and stores data). The first part includes the BOINC/XW probe and the Data Collector Daemon, the other part includes the Ganglia stack and the User Interface. We depict the block view in Figure 3 with the interactions existing in the whole system.

Data flow starts in probes, like the BOINC and XW probes, which collect data from the BOINC and XtremWeb databases. For example, these probes can collect the number of workunits that crossed the bridge, how many are waiting to be downloaded by any kind of worker, etc. These data follow to the Data Collector Daemon, which act as a hub, receiving, then filtering, (optionally) storing and sending data to whatever devices want to receive it. We use Java Enterprise Edition technologies, like Java Message Service and Enterprise Java Beans as the core technology for this

component. The user interface, which displays system information for administrators, is backed by the Ganglia stack that stores and manages data for plots (daily, weekly or monthly). For instance, to display periodic plots of the workunits running on some desktop grid, the Ganglia stack, must receive data on this parameter regularly (e.g., every 5 minutes, but this depends on the granularity one wishes to have on the parameters).

In our architecture, the flow of monitoring data goes through the following hops: i) a DG probe reads the Desktop Grid database, ii) sends data to the Data Collector Daemon, which iii) spawns one `gmetric` process. iv) From `gmetric`, data goes through the usual path in the Ganglia architecture to reach `gmetad`, which writes data in the `RRD Tool` database. Finally, v) when the user visits the appropriate page and requests a given parameter, a Java servlet in the Java Server Faces (JSF) framework will issue a request to the `RRD Tool` and produce a plot on-the-fly (if needed).

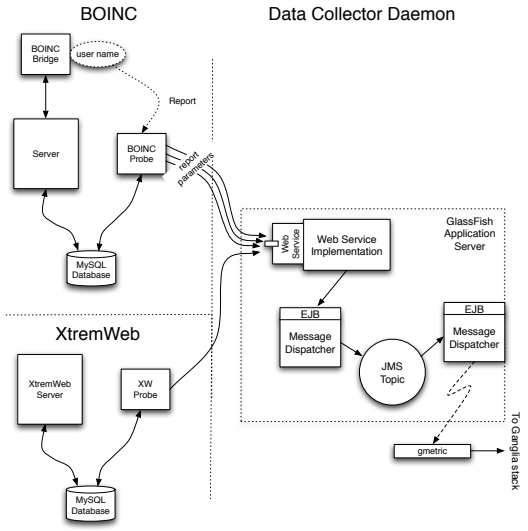

Figure 4. Data Collector Daemon

5.1. DG Probe that reads the Desktop Grid database

The BOINC and XW servers store all their data (users, workunits, which users have downloaded which workunits, the success of the workunits, etc.) in a MySQL database. To query this database, we use a DG probe that sends SQL queries to the server.

Since the operation of BOINC and XW probes are similar (for both bridge directions), to conserve space we only list the parameters we collect for the Desktop Grid→Service Grid direction for a BOINC DG. The list of parameters is pretty similar in XW, while for the opposite direction, this probe only collects data about workunits (WUs) that have crossed the bridge, thus reaching BOINC or XW. At the time we write this document, we collect data for the following parameters:

- **Running Workunits**. The number of WUs downloaded from the DG server both by the bridge and by regular clients. To the DG server, the bridge is simply another client, although very powerful.
- **Waiting Workunits**. The number of WUs in the DG server waiting to be downloaded (from clients and bridge).
- **Past Workunits**. The number of WUs that are finished (from clients and bridge).
- **Success Rate of Past Workunits**. If s is the number of WU results that clients submitted with success, and a is the overall number of WU results clients submitted, success rate is s/a.
- **Past Crossing-Bridge Workunits**. This parameter counts the number of finished WUs that crossed the bridge toward the SG.
- **Success Rate of Past Crossing-Bridge workunits**. s/a as the general success rate of all WUs, but this metric is restricted to the jobs that crossed the bridge.
- **CPUs Available**. Number of CPUs that are available.
- **GFLOPS**. Number of floating point operations per second in the entire DG divided by 10^9. We restrict this number to the CPUs that downloaded WUs in the last 6 hours.

One should notice that some of these parameters do not concern the bridge, but they may, nevertheless, convey some important information. Consider the case of success rate. It can be useful to compare the success rate of workunits that cross the bridge against the overall success rate of the same application. To get each one of these parameters we run SQL queries on the BOINC database. To install the BOINC probe one has to set the reporting interval of the probe. We currently set this to 300 seconds to reduce the impact of the probe as much as possible, while keeping a reasonable granularity for parameters.

5.2. The Data Collector Daemon

One of the components of our architecture is the Data Collector Daemon (DCD), shown in Figure 4. We wrote this daemon using technologies of Java Enterprise Edition (JEE), more precisely, Java Message Service (JMS) and Enterprise Java Beans (EJB), running on a GlassFish [15] Application Server. The fundamental component of the DCD is a JMS topic, where components exchange information. The BOINC and XW probes report their data to a web service. This web service connects to an EJB called "Message Dispatcher", which in turn publishes data to the JMS topic. By logically disconnecting the web service and the

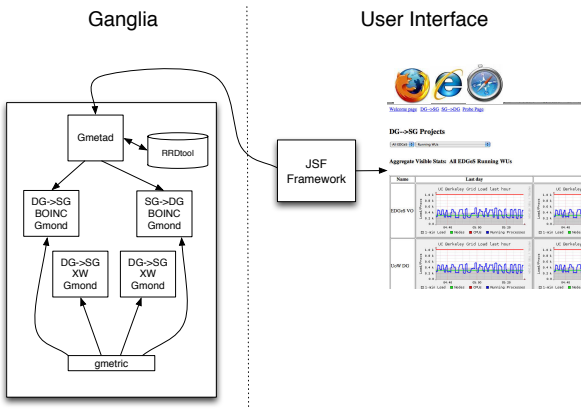

Figure 5. From `gmetric` to the web display

Message Dispatcher, we allow them to easily run in different machines if necessary. The path closes in another EJB, called "GMetricProcessorBean", which subscribes to the topic and launches an external process to run Ganglia's `gmetric`.

One of the most advantageous features of DCD is the decoupling it provides between production and consumption of data. In fact, this is quite typical to the message-oriented approach we are using here. In this way, we can easily add new sources of data, and new receivers, all without having to halt the platform. Consumers and producers do not need to be aware of each other, they are just connected by the JMS topic. Since, at the present time, we are mainly concerned with displaying periodic data of some parameters, data only flows to the Ganglia stack. However, there are many other services that should not go to Ganglia and that need a separate data flow. Perhaps one of the most important examples here would be to raise an alarm, when some parameter runs out of its acceptable range. This is very easy to do using standard JMS filtering.

5.3. The `gmetric` to `gmetad` Data Flow

The DCD spawns a process to execute the `gmetric` tool included in Ganglia. This `gmetric` process publishes the reported metric to `gmetad` (see Figure 5). We report data through `gmetric`, instead of getting it directly from the `gmond`, to overcome the limitations of `gmond`, as the latter cannot deal with new parameters. If we want to change parameters on the fly (e.g., adding the average waiting time of tasks in the bridge, or some other parameter), we would have to recompile `gmond`. While the DG probe runs in the same local area network as the BOINC or XW databases, the DCD and the entire Ganglia stack reside in a completely different place. Moreover, for each DG we only need to add one DG probe that accesses a single web service to report results to the Ganglia-based monitoring stack. `gmetric` sends new values to the multicast channel of the cluster, where all the listening `gmond`'s receive it. Later, `gmetad` collects this data from `gmond` and stores it in the RRD Tool.

As we can see in Figure 5, we run four instances `gmond`. Two are for the Service Grid→Desktop Grid direction, BOINC and XW, while the other two are for the Desktop Grid→Service Grid direction, BOINC and XW, also. This separation has to do with the internal organization of Ganglia. This enables Ganglia to store data from both bridges separately, with all the possibilities that come with this separation, like aggregation of results (which show up in the web user interface). For example, we can have separate views of the number of workunits in BOINC projects that crossed the bridge starting from *separate* DGs, but we can also have the *total sum* of such number starting from *all* DGs (this is a sum computed by Ganglia).

One important aspect regards the list of applications that exist in the web user interface. This list emerges automatically, as new probes connect to the JMS topic. In the first step, data of the newly connected application will reach the `gmond` and eventually the web interface without additional human intervention. However, the opposite step of *removing* an application requires manual intervention in `gmond`.

5.4. The `gmetad` to Data Display Flow

`gmetad` stores its information in local RRD files, which follow the format specified by the RRD Tool. The `gmetad` creates an XML summary that dictates the organization of the RRD files on disk. Then, we use a JDOM [16] parser, to convert this summary to Java structures that enable us to know exactly where to find the appropriate RRD file for a given parameter, in a given cluster (DG or EGEE). We cache the parser-generated information for five minutes before disposing and creating a new one.

To create the human readable data plots, we rely on the tools that come with RRD. To conserve CPU utilization, we minimize the number of times we need to create the plots. Specifically, the first time some user requests a plot, we build the corresponding png file [17] and store it on disk in a place accessible to the Java servlet that creates the HTML with the plots. Then, we keep this png file for a time period that depends on the time-span of the plot. For example for last-day statistics we keep this file 5 minutes, before creating a new instance. For last-week statistics we keep it for 1 hour, while for last-month statistics we keep it for 12 hours. If a second user requests the same plot within this time limits, the web server replies with the cached plot, otherwise, it generates a new one. In this way, we reduce the latency felt by the user for subsequent requests, while, at the same time, we contain the monitoring system's load.

5.5. Organization of Monitoring Data

We need to organize the directories of the RRD Tool (as part of Ganglia) according to the EDGeS infrastructure

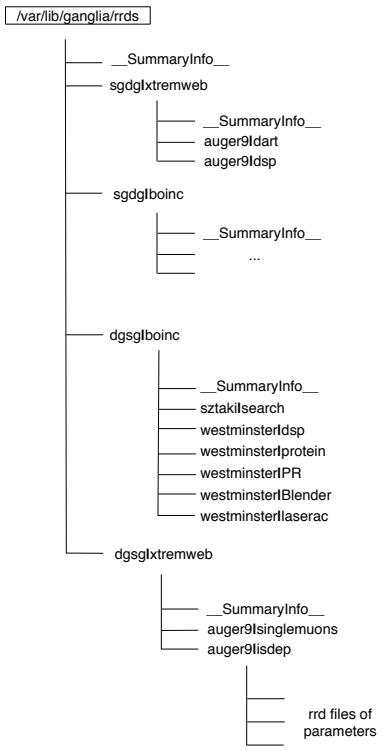

Figure 6. Internal organization of information in our Ganglia-based monitoring tool

and, in particular, we must make the separation between both bridge directions explicit, to show this on the web site. Refer to Figure 6. In the first level, we split contents in four parts, one for each direction of the bridge, both in XW and BOINC. These accounts for four directories `dgsg|boinc`, `dgsg|xtremweb`, `sgdg|boinc` and `sgdg|xtremweb`. In addition to these two directories, the RRD tool adds another directory deemed `__SummaryInfo__`, to aggregate parameter values from all directions. This directory is useful to aggregate results for all applications in one of the directions of BOINC or XW. For example, if we have the number of CPUs for `sztaki|search`, `westminster|dsp`, etc., the `__SummaryInfo__` directory of the corresponding level stores the aggregate number of CPUs, resulting from addition of *all* applications of BOINC running a Desktop Grid→Service Grid bridge.

5.6. Monitoring User Interface

We use the GlassFish application server to generate the web pages that display data collected by monitoring. The current address of our monitoring web site is http://edges.dei.uc.pt/EDGeSMonitoring/. We keep a second site (http://edges.dei.uc.pt:8080/EDGeSMonitoring/) for testing purposes. Since we changed the normal operation of the Ganglia stack, and due to the specific requirements of the EDGeS project, we decided to build our own web-based application from scratch, using the Java Server Faces framework. One should notice that programming costs of this decision are not that high, because it is easy to create and use the png files from the `gmetad` in our own setting. More precisely, we can easily manipulate Ganglia as we need despite not using the native Ganglia User Interface. To implement some features of our web site (e.g., getting the entire list in a drop-down list or using some special tags in a HTML table), we had to use a couple of Tomahawk components [18], like `datalist` and `dataTable`. These components enrich the standard JSF implementation.

At the current time, the interface consists of the following main web pages: “Welcome page”, “About”, “Desktop Grid→Service Grid”, “Service Grid→Desktop Grid”, “Parameters” and “Contact”. Everywhere in the web site, there is a menu on top, with links to any of the aforementioned pages. The “Welcome page” contains some static administrative information about the EDGeS project, as well as some configuration information. The “About” page contains the credits of the EDGeS project team, of web page maintainers and designers. The “Parameters” page contains the description of the parameters we collect and display in the “Desktop Grid→Service Grid” and “Service Grid→Desktop Grid” pages. The “Contact” page contains mail address of the web site maintainers.

The Desktop Grid→Service Grid and Service Grid→Desktop Grid have similar controls: an XtremWeb/BOINC radio button, a drop-down list of Grids (typically administratively different sites) and another drop-down list for applications (i.e., BOINC or XW applications that run in clients). Figure 7 gives an overview of the Desktop Grid→Service Grid page. The default view contains three plots per application per parameter, corresponding to the last 24 Hours, to the Last Week and to the Last Month. Additionally, we can replace the plots by numerical values. To switch on or off some of the views we use javascript, thus reducing the interaction with the web server.

6. Conclusion and Roadmap

In this paper, we described the monitoring architecture we have created so far for the EDGeS project. It contains four main components: a probe that runs SQL queries on the BOINC/XW server, the Data Collector Daemon, a Ganglia stack to treat data and a web user interface. One of the fundamental components of our architecture are the DG probes. While conceptually simple, DG probes are powerful enough to virtually collect all data we are currently observing for the entire platform. Current parameters concern mainly workunit accounting. One of the major shortcomings of this approach, however, is the difficulty to get in the EGEE side the same

Figure 7. Aggregate view of applications

information we get in the DG side, regarding execution of specific applications.

In the long term, and as the size of the EDGeS platform is expected to grow to a larger scale, we plan to include functionality tests, as described in Section 3. As a final vision for EDGeS monitoring, we will evaluate the possibility of integrating our current work in a powerful monitoring platform, like Nagios. While we need to have the possibility of keeping our tailor-made view of the grid (because, in fact, we need to view exchanges of jobs between different grids), an external framework can bring with it many other services that can significantly empower EDGeS, like standardized probing, user management or failure notification mechanisms, just to mention a few examples.

Acknowledgment

The EDGeS (Enabling Desktop Grids for e-Science) project receives Community funding from the European Commission within Research Infrastructures initiative of FP7 (grant agreement Number 211727).

References

[1] I. Foster, "What is the grid? - a three point checklist," *GRIDtoday*, vol. 1, no. 6, July 2002. [Online]. Available: http://www.gridtoday.com/02/0722/100136.html

[2] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *5th Intl Workshop on Grid Computing (GRID 2004), 2004, USA, Proceedings*. IEEE Computer Society, 2004, pp. 4–10.

[3] F. Cappello, S. Djilali, G. Fedak, T. Hérault, F. Magniette, V. Néri, and O. Lodygensky, "Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid," *Future Generation Comp. Syst.*, vol. 21, no. 3, pp. 417–437, 2005.

[4] "EGEE Portal: EGEE Portal," http://www.eu-egee.org/.

[5] M. Cárdenas-Montes, A. Emmen, A. C. Marosi, F. Araujo, G. Gombás, G. Terstyanszky, G. Fedak, I. Kelley, I. Taylor, O. Lodygensky, P. Kacsuk, R. Lovas, T. Kiss, Z. Balaton, and Z. Farkas, "Edges: bridging desktop and service grids," in *2nd Iberian Grid Infrastructure Conference (IBERGRID 2008)*, Porto, Portugal, May 2008.

[6] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 5-6, pp. 817–840, 2004.

[7] "JRA1.1: Prototypes of Bridge from Desktop Grids to Service Grids," Deliverable DJRA1.1 of the EDGeS Project, Jun. 2008.

[8] O. Lodygensky, G. Fedak, F. Cappello, V. Neri, M. Livny, and D. Thain, "Xtremweb & condor sharing resources between internet connected condor pools." in *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2003, p. 382.

[9] "Ganglia description," http://www.ibm.com/developerworks/wikis/display/WikiPtype/ganglia.

[10] "RRDtool — about RRDtool," http://oss.oetiker.ch/rrdtool/index.en.html.

[11] "EGEE-II SA1-documentation collection," http://egee-docs.web.cern.ch/egee-docs/list.php?dir=./mig/production/\&.

[12] "Gridview: Visualization and monitoring tool for lcg," http://gridview.cern.ch/GRIDVIEW/.

[13] "EGEE Accounting Portal," http://www3.egee.cesga.es/gridsite/accounting/CESGA/egee_view.html.

[14] E. Imamagic and D. Dobrenic, "Grid infrastructure monitoring system based on nagios," in *GMW '07: Proceedings of the 2007 workshop on Grid monitoring*. New York, NY, USA: ACM, 2007, pp. 23–28.

[15] "glassfish: Glassfish - open source application server," https://glassfish.dev.java.net/.

[16] "JDOM," http://www.jdom.org/.

[17] "PNG (portable network graphics) home site," http://www.libpng.org/pub/png/.

[18] "Myfaces tomahawk - apache myfaces - tomahawk," http://myfaces.apache.org/tomahawk/.