Benchmarking the EDGI Infrastructure

Serhiy Boychenko and Filipe Araujo

CISUC
Dept. of Informatics Engineering
University of Coimbra, Portugal

Abstract. The European Desktop Grid Initiative (EDGI) is an European project that aims to close the gap between service push-based grids, such as gLite, ARC and UNICORE, and desktop, pull-based, grids, such as BOINC and XtremWeb. Given the inevitably large size of the overall infrastructure, the EDGI team needs to benchmark its components, to identify configuration problems, performance bottlenecks, and to ensure the appropriate QoS levels.
In this paper, we describe our benchmarking effort. To take our measurements, we submitted batches of jobs to demonstration facilities, and to components that we disconnected from the infrastructure. We focused our measurements on the two most important metrics for any grid resource: latency and throughput of jobs. Additionally, by increasing job submission load to the limits of the EDGI components, we identified several bottlenecks in the job flow processes. The results of our work provide important performance guidelines for grid developers and administrators.

1 Introduction

Many different definitions exist for Grid computing [18, 7, 14]. Ian Foster provides the following checklist for a grid [18]: decentralized control of resources; standard, open, general-purpose protocols and interfaces; and the delivery of non-trivial quality of service. CERN, one of the largest users and promoters of grid systems defines grid as "a service for sharing computer power and data storage capacity over the Internet" [7]. CERN's Large Hadron Collider (LHC) was one of the grid driving forces in Europe with huge computational and data storage demands (15 PiB per year almost a decade ago). In response, European projects like the Enabling Grids for E-SciencE (EGEE) [4] emerged to integrate huge computational resources throughout the continent. EGEE knew several versions and its work goes on in projects like the European Grid Infrastructure [5]. It gave birth to well-known middleware, such as gLite, currently maintained by the European Middleware Initiative (EMI) [8].

As these service grids (SGs) were growing, desktop grids (DGs) were starting to become popular as well, by scavenging resources from millions of users. Most of the time, many of the computers that are turned on are simply idle [17]. The abundance of spare resources eventually motivated researchers to take advantage of underutilized computational power. BOINC is a result of this effort [13].

The Enabling Desktop Grids for e-Sciente (EDGeS) [16] was an European project that aimed to bridge the gap between Service Grids and Desktop Grids. The huge computational power owned by volunteers could support faster execution of parameter-sweeping applications submitted through SG standard mechanisms. The European Desktop Grid Initiative project (EDGI) [6] followed the EDGeS project, by including Cloud computing resources and extending service grids beyond gLite [8, 2]. EDGI now supports ARC [1], and UNICORE [11]. From any of these technologies, users may transparently submit jobs to BOINC and XtremWeb [15].

The resulting EDGI infrastructure is a large-scale distributed system, integrating many different technologies. To assess the performance of this infrastructure, we benchmarked the overall job flow process and the 3G Bridge[16], a key EDGI component. Another significant task we performed was to determine the infrastructure limits and the behavior of the system, when these limits are reached. By running tests in an overloaded system, we were able to find existing problems in the EDGI infrastructure, and to identify bottlenecks in job flows.

The rest of the paper is organized as follows: Section 2 presents the EDGI infrastructure. In Section 3 we present the benchmarking methodology and in Section 4 the results. In Section 5 we discuss the results and conclude the paper.

2 The EDGI Infrastructure

The purpose of the EDGI production infrastructure shown in Figure 1 is to provide DG and cloud resources for the ARC, gLite and UNICORE user communities. The gLite CREAM is a lightweight service for job management operation, integrated with gLite at the Computing Element level, enriching it with new functionalities. UNICORE is another Grid middleware, widely used in several supercomputer centers worldwide. UNICORE provides access to different kind of resources, ranging from database storage to computational resources. ARC stands for *Advanced Resource Connector* and offers client Grid middleware tools.

A user has several ways of submitting jobs to a Desktop Grid, e.g., using a Computing Element (CE) client or using a web-based EDGI portal. The gLite, ARC and UNICORE CEs were modified to include other infrastructure services, such as monitoring and ATTIC. ATTIC [10] is a peer-to-peer file system aiming to reduce required bandwidth for frequently used job input files. Refer to Figure 1. One should notice that this figure omits the Workload Management System and illustrates a simpler interaction with CEs. [12] provides a brief discussion about this issue. Once the job is staged by the Service Grid, it is submitted to the 3G Bridge through a web service interface. The 3G Bridge is the job-bridging component that serves as the gateway between different kinds of grids. Each supported Service or Desktop Grid technology has its own plugin deployed in the 3G Bridge. Through the available SG handlers, user requests are received in the bridge and forwarded to a DG, through the appropriate plugin. Then, the DG executes the jobs, and returns the corresponding results. Desktop Grid systems are also integrated with OpenNebula Cloud technology [9], to provide additional nodes capable of ensuring timely execution of batches of jobs.

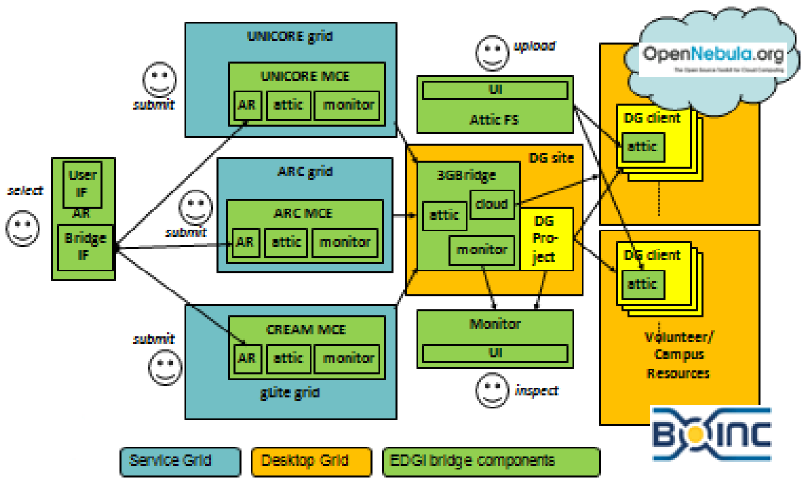

Fig. 1. EDGI Infrastructure.

3 Benchmarking Methodology

We believe that latency and throughput are the most relevant benchmarking metrics for grid users, because they can account for job delays and for numbers of jobs over time. For example, Aiftimieia *et al.* [12] also care for throughputs. To collect all necessary data for every particular component involved in the flow of jobs, we used the existing EDGI monitoring system [3]. In our experiments, we systematically increased the load of the system, to determine which components are responsible for EDGI's bottlenecks.

3.1 Metrics

Latency Latency is the time that elapses between two events of interest in the system. The goal of measuring latency is to identify the delays introduced by the different middleware components of the EDGI infrastructure. Timestamps of events related to job flows are collected from all accessible components of the EDGI infrastructure. We care for overall job completion latency, and for latency in other components of the infrastructure.

Throughput Throughput is the number of jobs processed per time unit. Full system and partial throughput information allows analysis of possible problems and bottlenecks, not only for a particular middleware, but also in the whole job flow process.

3.2 Test Description

Our tests were performed on the EDGI Demo site, which is very similar to the environment of Figure 1. As we show in Figure 2, we use two applications to collect all necessary data for the benchmarking. One of the applications, the Submitter Script, interacts directly with the User Interface (UI) provided by the EDGI infrastructure component (UCC, or gLiteUI for example). The Submitter Script executes commands that manage the job flow and collects the data available at the endpoint. In our experiments, we always used a small application with a very short execution time that is able to run in all Service and Desktop Grid technologies.

The Data Manager Application remotely collects monitoring data from other points. These reports are periodically generated and saved by the EDGI infrastructure in XML files. These files store events related to the entry of jobs to the infrastructure component, submission to the next component in the job flow process and status changes of each particular job. HTTP servers installed and configured on the EDGI infrastructure provide access to these XML report files [20]. The Data Manager Application also includes the Submitter Script data to perform the calculations required for each particular metric.

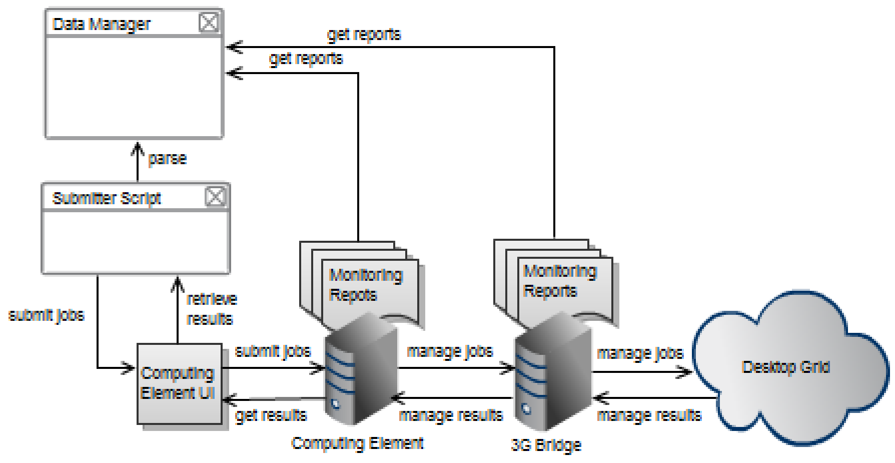

Fig. 2. Developed applications data collection scenario.

To measure latencies and throughputs, we collected, aggregated and processed timestamps in several points of the infrastructure (refer to Figure 3).

Timestamps T1 and T8 are collected on the submitter endpoint of the infrastructure, by the Submitter application. The remaining timestamps are collected

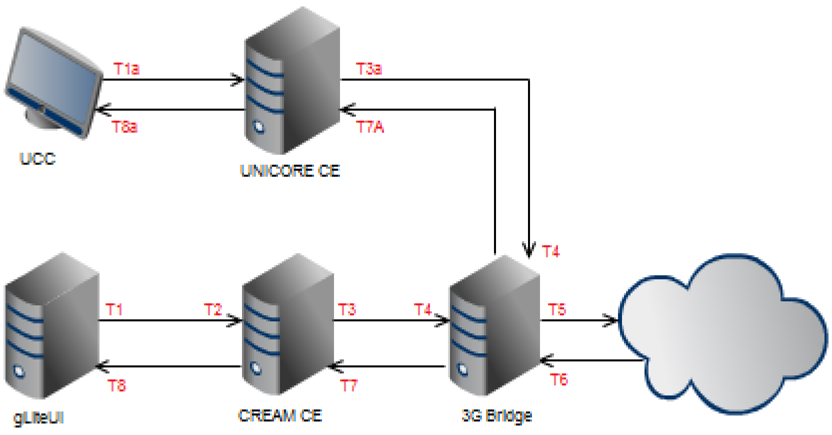

Fig. 3. Timestamps collected for throughput and latency measurment.

by the Data Manager application from the monitoring data generated by the components involved in the job flow. After collecting all the previous timestamps, data processing extracts the following useful intermediate timestamps necessary for posterior performance analysis:

T1, T1a : job submission on UI
T2, T2a : job entry to the Computing Element
T3, T3a : job submission to the 3G Bridge
T4 : job entry to the 3G Bridge
T5 : job submission to the Desktop Grid
T6 : job completion on 3G Bridge
T7, T7a : job completion on Computing Element
T8, T8a : job completion on UI

4 Benchmarking Results

In this section we present benchmarking results of the gLite and UNICORE technologies. Unfortunately, ARC was in an earlier stage of development that did not allow us to push it to the same limits as gLite and UNICORE.

4.1 gLite CREAM Results

The tests for measuring throughput and latency were run on a production infrastructure called EDGI Demo, installed at SZTAKI, Budapest, Hungary. In order to ensure correctness of results, we repeated the execution of each experiment 10 times. One should notice that some of the experiments take many hours to complete, which made it unpractical to use a larger number of samples. The configuration of every experiment allowed submission of the jobs during one hour, with a predefined submission rate, varying from 20 to 100 jobs per minute, with increments of 20 jobs for each new batch of tests. After test completion, results were collected and processed by the Data Manager Application.

Throughput In the EDGI Demo site, we identified two different throughputs that are relevant for CREAM CE: job submission throughput and job completion throughput. Here, we focus on the latter. To measure the completion throughput, up to the N-th job, we use the gLite UI's submission timestamp (T1) of the first job and the gLite UI's finish timestamp (T8) of the last job of interest to us, according to the following formula:

$$Throughput = \frac{N}{T8_{N^{th}job} - T1_{firstjob}} \tag{1}$$

In every batch of jobs, we omitted 15% of the initial results considering such period as warm up. In Figure 4, we illustrate the average throughput of the system as a function of the job submission throughput. We show standard deviation as error bars in the figure.

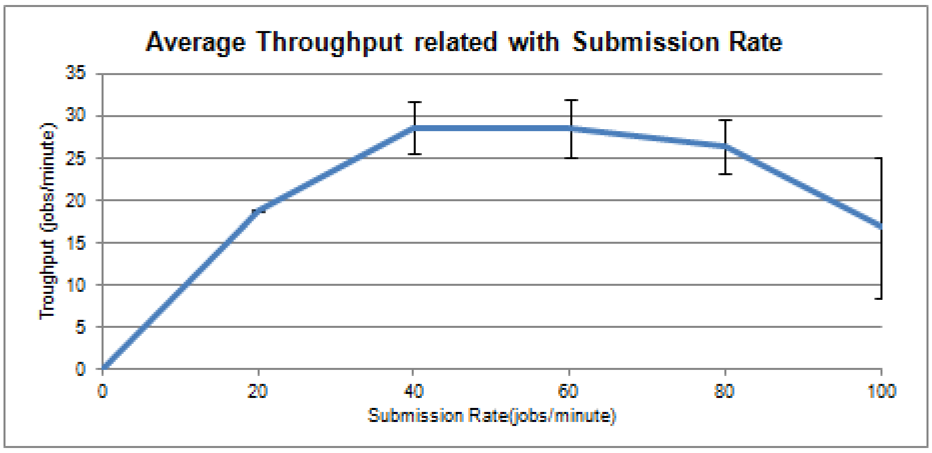

Fig. 4. CREAM average throughput related to submission rate.

One should notice the loss of throughput for higher submission rates. To understand what causes this effect, we proceeded to analyze the throughput of each particular element of the system. In the plot of Figure 5, we show the results of this analysis. The "Expected throughput" is the throughput that we would expect from a system without job delays, i.e., the same as the submission throughput. The blue part of the column shows the percentage of throughput actually obtained at given points of the infrastructure, besides the endpoint, which is the gLite UI. The smaller the blue part, the worse. We can see that problems start immediately in the gLite submission, but that they get amplified in the finalization. The finalization throughput is also the responsible for the low

UI throughput, as the latter is always limited by the slowest component (as any component is limited by the slowest upstream one).

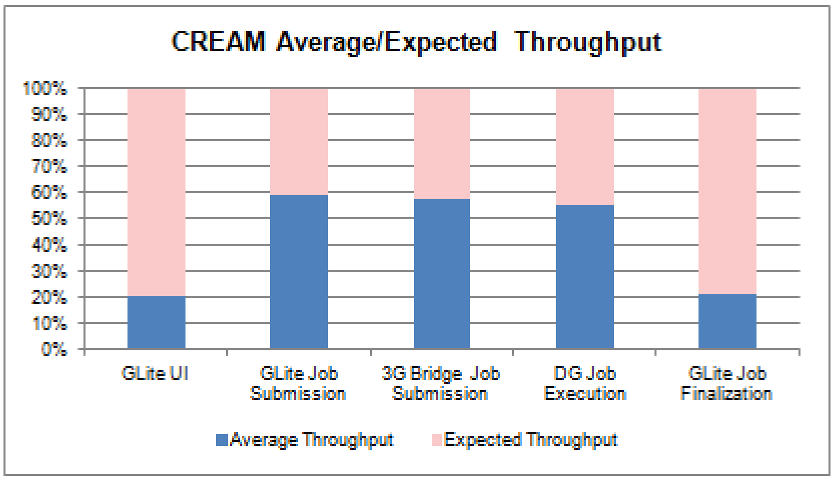

Fig. 5. CREAM Average/Expected throughput for 100 jobs/minute submission rate.

To understand if CREAM CE finalization could be the bottleneck of the EDGI Demo site, we analyzed the source code of the CREAM CE. We realized that there is an element responsible for job status update, called UpdateManager. UpdateManager is a Thread that retrieves jobs from a database and queries the 3G Bridge for the actual status of the job (IDLE, PENDING, RUNNING) using a Web Service. After getting response from the 3G Bridge, the UpdateManager handles the received response, updating the corresponding job status in the database and generating data for monitoring. When the number of jobs in the list is small, this model might work very effectively, but when the number of jobs in the list increases, the job status check processes consumes much more resources.

The CREAM CE job finalization is not the only bottleneck we identified. As CREAM CE was increasingly unable to keep up with the job submission rate, the number of failed jobs was increasing accordingly. This happens due to connection timeouts during the execution of the job submission commands.

Latency The latency analysis of the other EDGI Demo elements (3G Bridge and Desktop Grid) was also performed for comparison purposes and possible detection of infrastructure problems.

We used the timestamps collected at different points of the architecture to compute the latencies introduced by the different components. In Figure 6 we illustrate the latencies for different job submission rates.

We can observe a drastic increment of the CREAM CE job finalization time for heavier loads. After submitting 40 jobs/minute, we increased load to 60

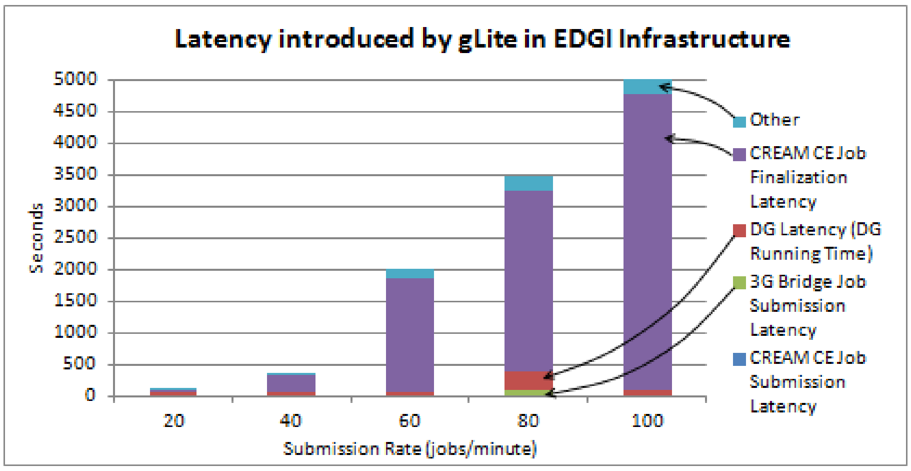

Fig. 6. Latency introduced by EDGI infrastructure components.

jobs/minute. This corresponds to a load increment of 150%. However, latency rose from 262.9 to 1796.1 seconds, a 685% increase. More detailed analysis allowed us to confirm that the CREAM CE finishing time directly influences the overall performance of the job running time and consequently of the whole experiment. The finalization bottleneck directly affects the overall system latency, by slowing down the CREAM CE job finalization process.

4.2 UNICORE Results

Benchmarking tests of the UNICORE CE were run on the infrastructure installed and provided by the University of Paderborn. We have used the UNICORE Command-line Client (UCC) user interface to send commands to the UNICORE server. Similarly to the tests previously executed on CREAM, we repeated each experiment 10 times. We varied submission rates between 10 and 40 jobs per minute. We were unable to push UNICORE to 100 jobs/minute submission rate (as we initially planned), due to performance problems described in the following section. In Figure 7, we illustrate the overall throughput of the system as a function of the job submission throughput.

We noticed that UNICORE was not able to handle more than 40 jobs/minute very well, stopping to respond to the UI commands. To understand what causes this effect, we proceeded to analyze the preparation throughput and finishing throughput of the UNICORE system. Figure 8 shows the results of our experiment. The lower part of the column is the actual throughput, while the upper part of the column is the expected (but unreached) throughput. The problem resided on the job submission from the client application (UCC) to the UNICORE server. This time was growing quite fast relatively to the job submission rate. Whenever a request takes too much time to submit, the client application reports an error and stops execution of the request. There are several problems

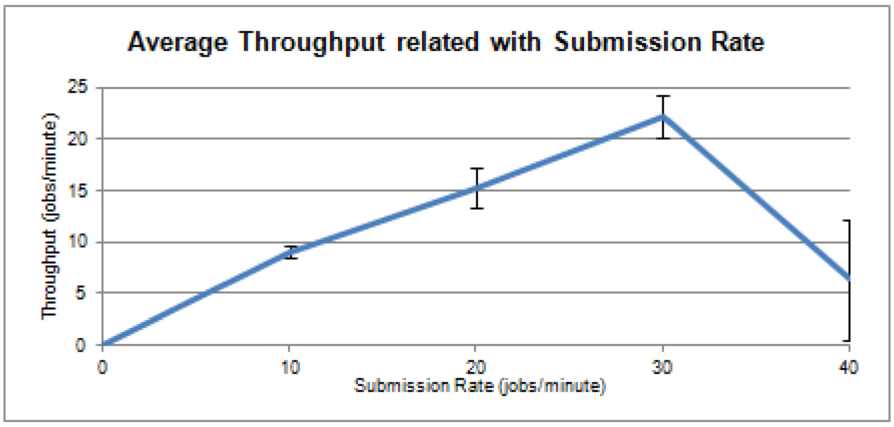

Fig. 7. UNICORE average throughput related to submission rate.

that may contribute to this issue, creating a bottleneck on the request submission part. The job submission procedure in UNICORE is quite heavy. The first step is authentication of the request, by querying a database. The next step, after the request is authorized is performing a Web Service call. A staging process follows to run the executable. It creates a working directory, it has file Input/Output operations, communication with one EDGI component called Application Repository, and database insertion operations. After successful staging, the request is transformed to a 3G Bridge compatible format and submitted to the bridge. All these steps require communication with external services, making them a possible cause of this bottleneck.

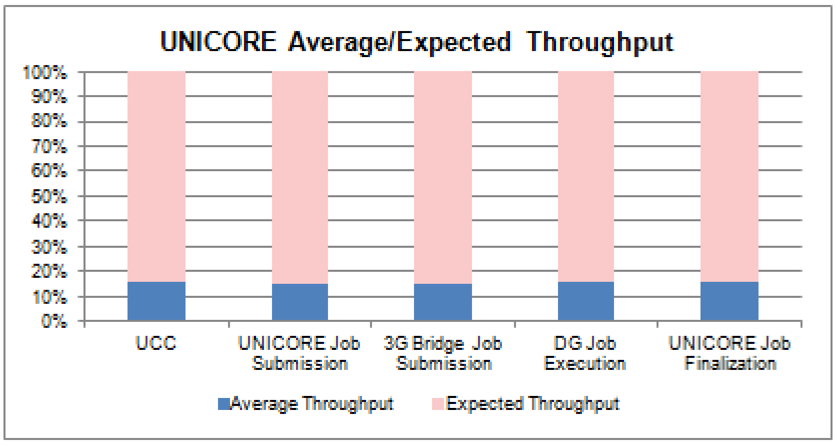

Fig. 8. UNICORE Average/Expected throughput for 40 jobs/minute submission rate.

Another problem could be the client application. To ensure that the Submitter Script keeps up with the defined submission rate, several submitter threads are created. Since there is no reuse of already running instances of the client application, every time a request is executed, a new UNICORE UI application instance is started. This process requires a lot of computational power on the machine where the tests are run. We ran some CPU Usage and Memory Usage tests, which confirmed the high demand of resources by UCC, especially CPU processing power.

UNICORE latency measurements allowed us to identify two important latencies to work with: CE Preparation Latency and CE Finishing Latency. Figure 9 shows these latencies for the UNICORE system. We could conclude that the latency of the 3G Bridge remained constant independently of the technology used upstream. Latency of the Desktop Grid stayed pretty much constant for the amount of jobs we submitted. We can also conclude that the latency of the UNICORE system is quite stable. We observed that a quite significant portion of average job execution time of a UNICORE job is due to the finalization latency. This can be explained by the staging out process, where UNICORE reports jobs as finished when all data is downloaded and cleaned from the 3G Bridge and Desktop Grids.

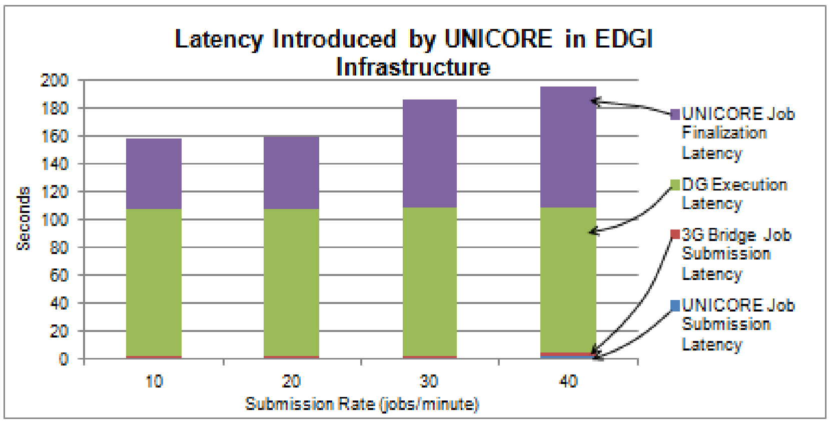

Fig. 9. UNICORE in the EDGI infrastructure.

4.3 3G Bridge Results

Besides the test ran on the EDGI Demo site, we ran benchmarking tests on the 3G Bridge installed on our own local machines. This version of the bridge is newer than the one we tested in [19]. The main goal of the 3G Bridge benchmarking was to check if bottlenecks exist in this component. From the tests executed on the EDGI Demo, no performance problem is apparent in the 3G Bridge, because CREAM CE, which lies upstream exhibited problems first, starting at 80 jobs/minute. We have installed the latest available version of the 3G Bridge. For test purposes, we configured the NullHandler plugin, which marks jobs as finished, as soon as it receives a job status query. After performing latency analysis, we could not find any noticeable problem with this metric. For throughput, we were also unable to detect any anomaly in the 3G Bridge performance, up to 100 jobs/minute. The average throughput in all experiments was equal to the expected throughput. Based on these results, we could conclude that there are no performance issues or any bottleneck spotted on the 3G Bridge for the currently existing demands.

5 Discussion and Conclusions

The main conclusion we can take is that the EDGI Demo site can deal with up to 1 job/second without demonstrating any particular problems. After this point, scalability problems emerge on the CREAM Computing Element. Throughput and latency tests confirmed that CREAM CE job finalization process was not efficient enough to handle higher workloads. Evaluation of the source code of the CREAM Computing Element identified the UpdateManager component as the source of this bottleneck. Since this component checks jobs status sequentially, our results suggest that it should be multi-threaded. A second source of performance problems of CREAM CE lied on the submission of jobs. As we try to increase job submission rate, the number of jobs that we are unable to submit also grows.

UNICORE did not show any latency related problems in our measurements. However, throughput losses after the 40 jobs/minute threshold prevented us from executing tests with higher submission rates. We could conclude that one of the causes for this problem is the UNICORE Client-Server communication, making this part the major bottleneck of the system. A deep analysis of the different phases of the job flow revealed problems on the UNICORE server request processing. This also impacted the client CPU consumption.

Unlike the Computing Elements, the 3G Bridge seems to be efficient enough for the EDGI Demo site. This component is definitely not the major bottleneck of the system. In fact, we were unable to observe any unpredicted behavior in the 3G Bridge measurements, because this component responds very well to the job submission rates we used. In general, we concluded that performance and quality of service provided by the 3G Bridge were highly satisfactory, both in isolated and in EDGI Demo measurements.

Acknowledgments

The research leading to these results is funded by the European Union Seventh Framework Program (FP7/2007-2013) under grant agreement no 261556.

References

1. Advanced resource connector. `http://www.nordugrid.org/arc/`. Visited on June 19, 2012.
2. Computing resource execution and management service. `http://grid.pd.infn.it/cream/`. Visited on June 19, 2012.

3. Edgi infrastructure monitoring web page. `http://edgi.dei.uc.pt/NewEDGIMonitoring/`. Visited on June 19, 2012.
4. Egee portal: Enabling grids for e-science. `http://www.eu-egee.org/`. Visited on June 19, 2012.
5. Egi. `http://www.egi.eu/`. Visited on June 19, 2012.
6. European desktop grid initiative. `http://edgi-project.eu/`. Visited on June 19, 2012.
7. Gridcaf'e: the place for everybody to know about grid computing. `http://www.gridcafe.org/`. Visited on June 19, 2012.
8. Home - european middleware initiative. `http://www.eu-emi.eu/`. Visited on June 19, 2012.
9. Opennebula is an open-source project developing the industry standard solution for building and managing virtualized enterprise data centers and cloud infrastructures. `http://opennebula.org/`. Visited on June 19, 2012.
10. P2p data sharing software architecture. `http://www.atticfs.org/`. Visited on June 19, 2012.
11. Uniform interface to computing resources. `http://www.unicore.eu/`. Visited on June 19, 2012.
12. Cristina Aiftimiei, Paolo Andreetto, Sara Bertocco, Simone Dalla Fina, Alvise Dorigo, Eric Frizziero, Alessio Gianelle, Moreno Marzolla, Mirco Mazzucato, Massimo Sgaravatto, Sergio Traldi, and Luigi Zangrando. Design and implementation of the glite cream job management service. *Future Gener. Comput. Syst.*, 26(4):654–667, April 2010.
13. David P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.
14. Rajkumar Buyya and Srikumar Venugopal. A gentle introduction to grid computing and technologies. *CSI Communications*, 29(1):9–19, July 2005. Computer Society of India (CSI).
15. Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Hérault, Frédéric Magniette, Vincent Néri, and Oleg Lodygensky. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Comp. Syst.*, 21(3):417–437, 2005.
16. Miguel Cárdenas-Montes, Ad Emmen, Attila Csaba Marosi, Filipe Araujo, Gábor Gombás, Gabor Terstyanszky, Gilles Fedak, Ian Kelley, Ian Taylor, Oleg Lodygensky, Péter Kacsuk, Róbert Lovas, Tamas Kiss, Zoltán Balaton, and Zoltán Farkas. Edges: bridging desktop and service grids. In *2nd Iberian Grid Infrastructure Conference (IBERGRID 2008)*, Porto, Portugal, May 2008.
17. Patricio Domingues, Paulo Marques, and Luis Silva. Resources usage of windows computer laboratories. Technical Report Technical Report. `http://www.cisuc.uc.pt/view_member.php?id_m=207`, CISUC, January 2005.
18. Ian Foster. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.
19. Naghmeh Ivaki, Diogo Ferreira, and Filipe Araujo. Benchmarking the 3g-bridge in the edges infrastructure. In *Cracow Grid Workshop (CGW'09)*, September 2009.
20. Jozsef Kovacs, Filipe Araujo, Serhiy Boychenko, Mathias Keller, and Andre Brinkmann. Monitoring unicore jobs executed on desktop grid resources. In *35th Jubilee International Conference on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2012)*, Opatija, Croatia, May 2012.