Masters' Degree in Informatics Engineering

Internship

Final Report

EDGI Project Infrastructure Benchmarking

Serhiy Boychenko Viktorovich

serhiy@student.dei.uc.pt

Advisor:

Filipe João Boavida de Mendonça Machado de Araújo

July, 12, 2012

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Table of Contents

List of Figures

List of Tables

Glossary

3G Bridge	The Generic Grid-Grid Bridge
AlmereGrid	BOINC Test Grid
AR	Application Repository
ARC	Advance Resource Connector
ATTIC	P2P File System
BOINC	Berkeley Open Infrastructure for Network Computing
BSD	Berkeley Software Distribution
CE	Computing Element
CNRS	National Centre for Scientific Research (translated)
CPU	Central Processing Unit
CREAM	Computing Resource Execution And Management
CSV	Comma Separated Values
DG	Desktop Grid
DSP	Digital Signal Processing
EDGeS	Enabling Desktop Grids for E-Science project
EDGI	European Desktop Grid Initiative
EGI	European Grid Infrastructure
EGEE	Enabling Grids for E-sciencE
EMI	European Middleware Initiative
FCTUC	Faculty of Science and Technology (translated)
gLite	Lightweight Middleware for Grid Computing
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IberCivis	Distributed computing platform
IF	Interface
INRIA	National Institute for Research in Computer Science and Automation (translated)
KnowARC	Former ARC development project
LCG	LHC Computing Grid
LHC	Large Hadron Collider
LIPCA	Certification Authority
mCE	Modified Computing Element
NGI	National Genetics Institute
NorduGrid	Nordic Testbed for Wide Area Computing and Data Handling
OpenNebula	Cloud Data Center Management Software
P2P	Peer-to-Peer
QoS	Quality of Service
RDBMS	Relational DataBase Management System
SG	Service Grid

SZTAKI	Computer and Automation Research Institute, Hungarian Academy of Sciences (translated)
UCC	UNICORE Commandline Client
UI	User Interface
UNICORE	Uniform Interface to COmputing REsources
URL	Uniform Resource Locator
VO	Virtual Organization
XML	eXtensible Markup Language
XtremWeb	Open Source Platform for Desktop Grids

Abstract

Many modern scientific facilities such as synchrotrons, particle colliders, satellites, telescopes, and lasers can generate terabytes of data daily. A single computer can take centuries, millenniums, or even longer, to analyse the data coming from these systems. The EDGI project combines already existing grid, desktop grid and cloud resources into a new system, to increase the computational power of e-Science. For this, EDGI created and improved bridging components between different grid technologies. In this way, EDGI can gather huge computational resources for data intensive applications, and still let users keep their old job submission and control interfaces.

Given the amount of resources available, we need to ensure that the EDGI middleware provides adequate performance. In particular, one of the main quality assurance tasks of EDGI is the infrastructure benchmarking, which should ensure a highly predictable and performing system. Coimbra University, one of the partners of the EDGI project is responsible for the benchmarking task.

In this report, 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

1.1. Motivation

Most of the time, many of the computers that are turned on are idle or waiting for some task. Even if the computer is actively used, for browsing the Internet, checking e-mail, or listening to music, only a fraction of its resources is actually involved in such operations [1]. The abundance of spare resources eventually motivated researchers to make use of unutilized computational power. The first initiatives trying to do so were named "Volunteer Computing".

Volunteer Computing infrastructures are, in fact, large-scale distributed systems [2]. Their form is generally pretty simple: the anonymous client (anyone can install a client to "donate" its resources), periodically contacts an infrastructure server requesting jobs and reporting results of completed jobs. The core of the system lies on the server side, an infrastructure that has to deal with the distribution and monitoring of the jobs, collection, validation and organization of results. The anonymity and publicity of Volunteer Computing in some cases makes it unsuitable for some organizations to use, which led to appearance of Grid Computing.

Grid Computing [3] is a form of distributed computing where an organization uses its existing computers to run its own computational tasks. One of the considerable differences of Grid Computing, comparing to Volunteer Computing is the trust on the computational resources. This prevents the need for replication of computation, which is extensively used in Volunteer Grid computing to ensure that computations are correct. While in Volunteer Computing one of the most important problems is to guarantee that there will be no performance loss when the user is actively using its own computer (i.e., it should not perform any volunteer computation in such occasions), Grid Computing has no such problem, since sometimes it is even desirable to have the computation completely invisible and out of user control.

One of the main problems of Grid Computing was the impossibility of creating a grid for every single organization, which needs computational power and cannot get it from Volunteer Computing (due to security or privacy issues of the application, or due to problems related to the infrastructure capacity). As a consequence, in April 2004, the EGEE [4] project started to overcome such limitations of Grid Computing, to improve quality, robustness and consistency of the service provided by Grids, attract new resources and users (researchers) to the network.

Given the considerable fragmentation of resources that resulted from a number of different middleware existing for deploying grids, there was a need to create a bridging technology between different Grid systems, which could provide even more computational power to e-Science. To fulfil this goal, the EDGeS [5] project started in the first day of 2008. It aimed at creating bridge technology between different kinds of middleware, more concretely the EGEE project and the BOINC [6] and XtremWeb [7] Desktop Grids. Together with bridging technology, new interfaces and tools for application development, new trust mechanisms (Application Repository [8] for example) were implemented, formal procedures for computation resources usage of new e-infrastructure were defined and established.

In 2010, the EDGeS project gave place to the new EDGI [9] project. One of the main objectives (Figure 1) of the EDGI project is consolidation of results achieved in its antecedent phase (EDGeS) by allowing the integration of previously deployed infrastructure with new technologies (like ARC [10], UNICORE [11]) and extending Desktop Grids with Cloud technologies (like OpenNebula [12]) to improve the QoS provided by DGs. Enhancing e-infrastructure with new capabilities will allow to provide DG resources for EGI [13] and NGI user communities, improve QoS and provide more computational power for already existing users.

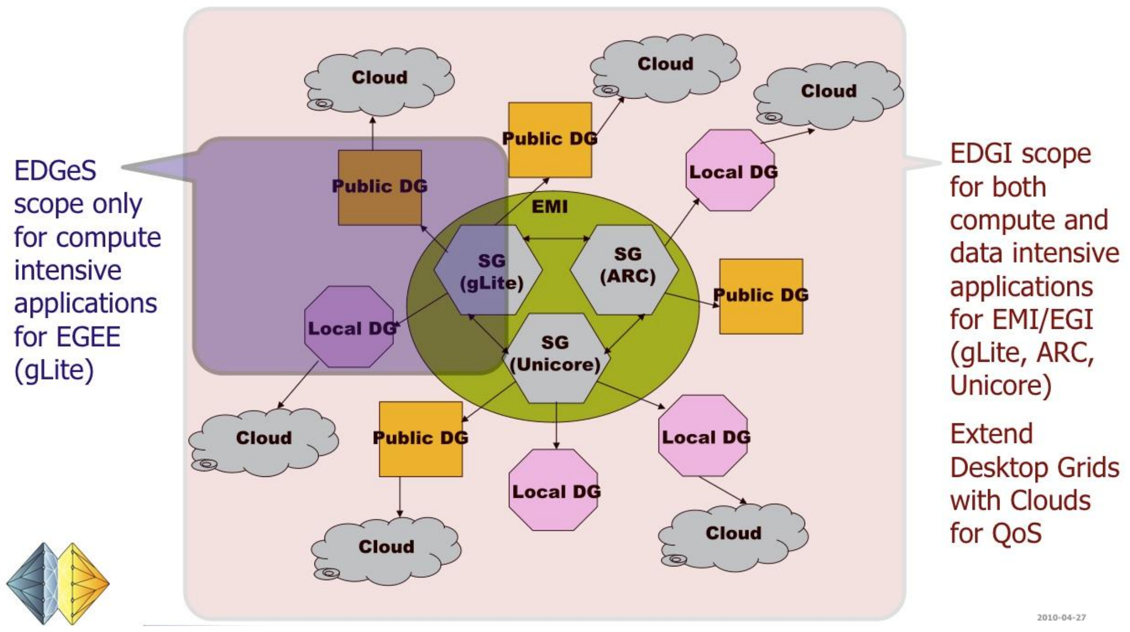

EDGeS scope only for compute intensive applications for EGEE (gLite)
EDGI scope for both compute and data intensive applications for EMI/EGI (gLite, ARC, Unicore)
Extend Desktop Grids with Clouds for QoS
Cloud
Cloud
Cloud
Public DG
Public DG
Local DG
EMI
SG (gLite)
SG (ARC)
Public DG
Local DG
SG (Unicore)
Cloud
Public DG
Local DG
Local DG
Cloud
Cloud
2010-04-27

Figure 1: Scope of EDGI project

The University of Coimbra is one of the current partners of the EDGI project, actively participating in the initiative. One of its main tasks is benchmarking the developed infrastructure. The process of benchmarking consists of running a script or set of scripts for information gathering, to measure performance of some object (program, system, and infrastructure). In the EDGI project benchmarking, benchmarking should also allow comparison of the performance of different elements with different configurations, detection of possible bottlenecks and improvement of configurations in different points of the whole infrastructure.

1.2. Objectives

The main goal of this internship is to benchmark the EDGI infrastructure (the production environment of Figure 2) and contribute to the quality of services provided by the project software. To assure that the ultimate goal will be successfully achieved we have identified several secondary objectives.

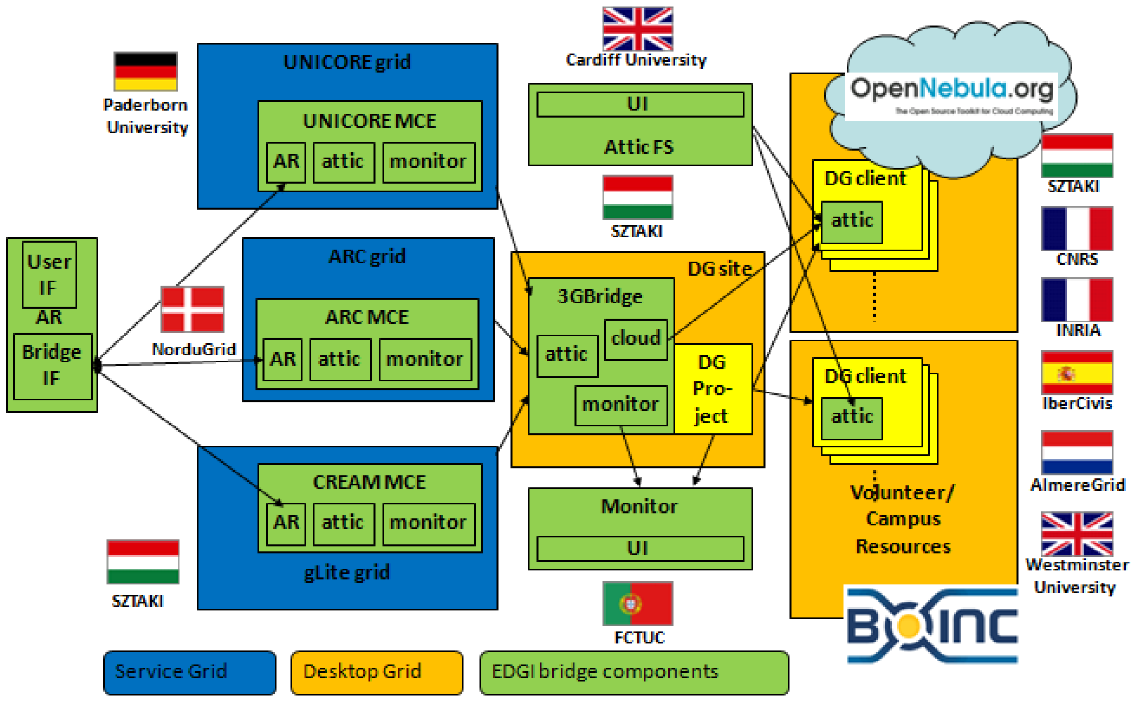


Figure 2: EDGI Infrastructure

One of these goals concerns the definition of a benchmarking strategy and methodology. We focused on the definition of metrics that would precisely describe the performance of the EDGI structure components.

Another secondary goal is to provide detailed performance information about the EDGI infrastructure to find system's bottlenecks. The results of our measurements should contribute to the quality of services provided by the EDGI infrastructure and enable middleware developers to fix problems before the project ends. This work also aims to review some of the project's code, to identify the source of the bottlenecks.

We also aim to provide comparison between different Computing Elements and simplify system performance tuning for DG and SG administrators. This involves merging measurement results, to determine the most architecturally efficient approach.

1.3. Progress Report

This section focuses on the description of the main tasks performed during the internship period, providing some details about challenges and difficulties we faced and managed to overcome.

In our work we focused on the description of the methodology and results of the EDGI infrastructure benchmarking. It is important to note that a big part of the work consisted of analysing the architecture of EDGI's components, gaining permissions and access to infrastructure resources and communicating with different parties in order to ensure the correctness of the benchmarking processes and results. Certificate attainment from LIPCA [14] took around one month and was the first task to perform. While the certification authority was issuing the necessary permits to let us access the Grid resources, we performed the study of the EDGI infrastructure components, to understand the architecture of the system and respective components (namely gLite with CREAM CE [15], ARC, UNICORE and 3G Bridge [16]). Different technology analysis allowed us to define metrics and its measurement process.

After getting access to the EDGI Demo site (which involved many communication with system administrators of different Grids), we started experiments with different technologies. After familiarization with middleware, we developed two applications: Submitter Script and DataManager Application. These applications allowed performing automated tests on previously analysed systems.

Using developed tools we started performance measurements on gLite with CREAM mCE, installed on EDGI Demo infrastructure at SZTAKI. We ran many tests with different configurations detecting the problems and limitations of gLite middleware. Our measurements' results were presented in the interim internship report, highlighting performance loss problem we have identified during our tests.

In the second semester, we have identified improvements to be done on developed applications. On Submitter Script we improved middleware configuration system, implemented thread pools to decrease resource consumption and assurance of submission rate. On DataManager application stability improvements were carried out and metric parsing and merging process was redesigned. Additionally JProfiler tests were performed on DataManager, detecting possible memory leaks and unexpected resource usage by our application.

Using improved tools we started execution of new batches of tests. First we executed more tests on gLite with CREAM mCE. New measurements focused on repetition of the tests with same configuration for 10 times distributed over 24 hours of the day. During our tests several stability problems were spotted and reported to SZTAKI developers. While running tests on EDGI Demo site, we have installed 3G Bridge and BOINC in local environment on Virtual Machines, performing independent performance tests of 3G Bridge. These tests were run with different submission rates, repeating the measurement with same configuration 10 times over 24 hour period of day. After finishing the tests on gLite, we proceeded to ARC mCE measurements. Since ARC site was connected to EDGI Demo 3G Bridge, we could not execute the tests on different Computing Elements at the same time, introducing noise in results and creating unequal test conditions. We were able to execute some of the planned measurements on ARC but lack of stability of the ARC mCE did not allow us to continue with the planned measurements. Finally we started UNICORE mCE test, managing to execute some of the planned experiments before detection of performance problems on this middleware. After finishing all measurements we started analysis of the obtained data and description of results.

To simplify infrastructure configuration, we have developed EBench application, which after simple configuration performs benchmarking tests and produces graphical results using Google Charts.

1st Semester
X509 certificate attainment
Study and definition of metrics and respective calculation methods
GLite (with CREAM as Computing Element) architecture study
ARC architecture study
UNICORE architecture study
3G Bridge architecture study
Study of the data collection methods necessary for metric calculations
Submitter Script architecture definition
Submitter Script implementation
DataManager Application architecture definition
DataManager Application implementation
Unit tests of Submitter Script and DataManager Application
Test execution on gLite (with CREAM as Computing Element) with variable job submission rate
Intership Interim Report elaboration
2nd Semester
Submitter Script improvements and modifications - Thread Pools implementation for job submission and status check - Implementation of more informative and extensive logging - Scalabale configuration of new Computing Elements implementation
DataManager Application improvements and modifications - Autonomous modules for metric calculation implementation - Resource consumption and stability improvements
Test of Submitter Script and DataManager Application
Test execution on gLite (with CREAM as Computing Element) with variable job submission rate
Test execution on ARC with variable job submission rate
3G Bridge and BOINC installation of local environment
Test execution on 3G Bridge with variable job submission rate
Test execution on UNICORE with variable job submission rate
EDGI project deliverable D6.3 elaboration
EBench application architecture definition
EBench application implementation
Preparation and presentation of the paper "Monitoring UNICORE jobs executed on Desktop Grid resources" at MIPRO 2012
Elaboration of the paper "Benchmarking the EDGI Infrastructure" for INForum 2012

Table 1: Tasks performed during internship period

1.4. Results Achieved

After performing the tasks described in the previous section we can state that we successfully achieved the defined goals.

One of the first achieved results was successful definition and implementation of benchmarking methodology. EDGI infrastructure architecture study allowed us to identify mathematically calculated metrics: latency and throughput, which successfully reflect system's performance. Using the latency metric we were able to highlight the behaviour of infrastructure components with different job loads. Latency also provided information about delays introduced by different components of EDGI infrastructure in job flow process..Using throughput metric, we were able to explore the limits of different middleware connected to the EDGI system and describe its behaviour when these limits are reached.

We developed different applications to perform planned measuments: Submitter Script and DataManager Application. Using developed tools we managed to perform benchmarking tests on different components of EDGI infrastructure: 3G Bridge, gLite (with CREAM Modified Computing Element), ARC and UNICORE. All planned tests were ran on 3G Bridge and gLite, while on ARC and UNICORE only part of desired measurements was performed, due to stability problems of these Computing Elements.

The data we collected from benchmarking tests allowed us to analyse specific middleware performance and provide performance comparison of different components of the EDGI infrastructure. We managed to identify job finalization and job submission (with higher load) bottlenecks on gLite. We spotted the architectural problem of UNICORE UI tool, related to the high usage of machine resources. During our measurements, several stability issues on different Computing Elements were revealed and reported to middleware developers. We managed to prove that 3G Bridge was definitely not a bottleneck in the system.

Our work also resulted in a paper [17], which was elaborated in the end of internship. The paper "Benchmarking the EDGI Infrastructure" was accepted at the INForum 2012 symposium (INForum 2012 – 4º Simpósio de Informática: http://inforum.org.pt/INForum2012/).

1.5. Outline

The remainder of this document is organized as follows. Chapter 2 describes and proposes the benchmarking methodology, including the metric definition, test description and configuration. Chapter 3 describes the architecture of the EDGI infrastructure, as well as the architecture of Service Grids and 3G Bridge. Chapter 4 presents and discusses the results of our measurements. Chapter 5 concludes this report.

2. The EDGI Architecture

The purpose of the infrastructure shown in Figure 2 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 [18]. ATTIC is a peer-to-peer file system aiming to reduce the bandwidth usage for frequently used job input files. After 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, to provide additional nodes capable of ensuring timely execution of batches of jobs.

2.1. The CREAM mCE

In the previous EDGeS project, the bridge developers created the bridge between gLite and the DGs, using the LCG-CE component. There were different reasons in EDGI to migrate from the traditional gLite to CREAM CE. This mCE has a cleaner architecture for job management. It also has support for different kinds of services, which are not supported by LCG-CE (for example batch jobs, which make possible to submit a high number of jobs with one single command). In CREAM CE, a new connector (EDGI Executor) was created. Its goal is gLite job interception and forwarding to 3G Bridge services. The new connector behaves like a batch system implementation with the difference that the job is not run on a Worker Node, but is sent to the 3G Bridge for execution on the grid.

2.1.1. CREAM mCE Architecture

The EDGI Executor consists of a number of components as shown in Figure 3: ConfigReader, EDGIExecutor, ARWrapper, BridgeSubmitter, EventLogger and UpdateManager.

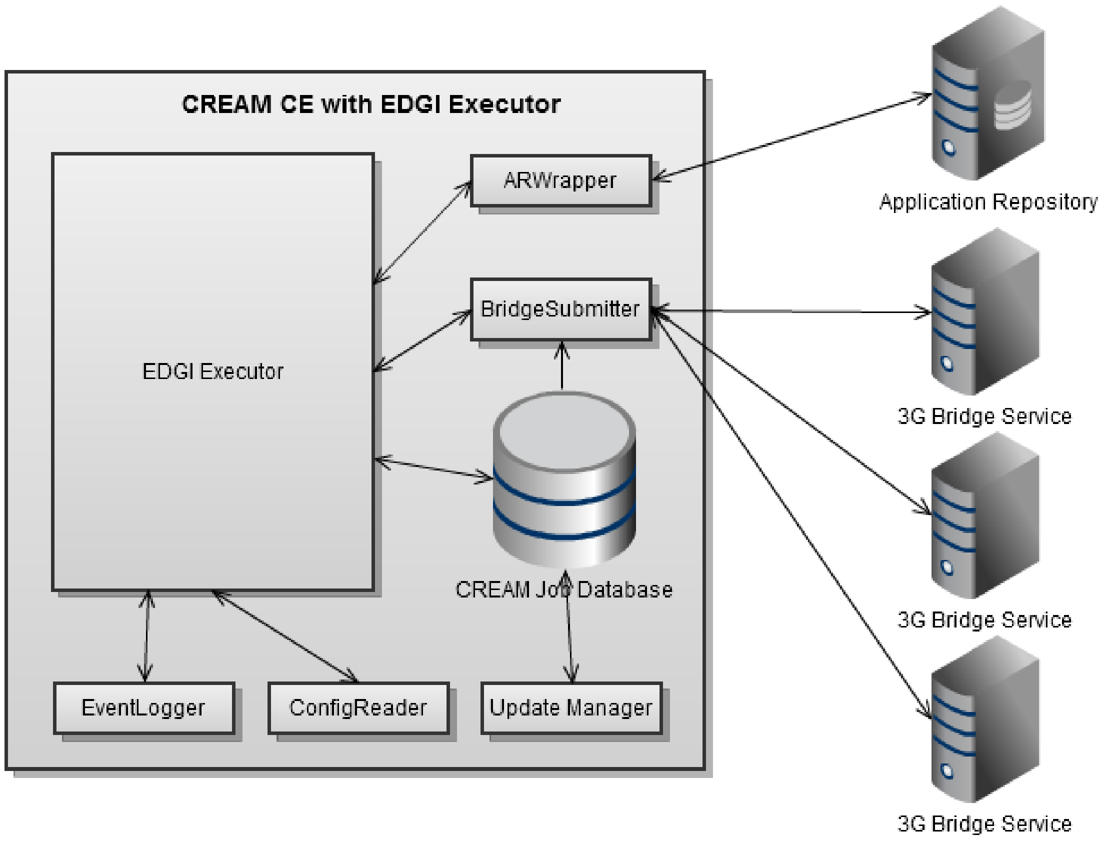


Figure 3: CREAM CE architecture

EDGIExecutor is the component responsible for job execution. Main functionalities of CREAM's executor are job submission, job status management, job cancelation and handling of input/output files. EDGIExecutor mainly makes use of ARWrapper (explained below) module when job validations on job submission event are performed and BridgeSubmitter during the whole job flow process. ARWrapper communicates with EDGI Application Repository in order to validate job's application is supported by 3G Bridge. BridgeSubmitter used for translation of gLite to 3G Bridge job descriptions and communication with bridge's web service (WSSubmitter), performing all necessary interaction in job flow process. ConfigReader and EventLogger are another components used by EDGIExecutor. ConfigReader component is used on start-up for reading and parsing its configuration files (in XML format). EventLogger is used for logging events related with job flow process. Finally, UpdateManager is the component responsible for periodical (10 seconds) job status update, which is performed by calling EDGIExecutor jobStatus command for every job existing in Job Database.

2.1.2. CREAM mCE Job Life Cycle

The following figure (Figure 4) shows general overview of job states (statuses) in CREAM CE, which is helpful in understanding of job flow process.

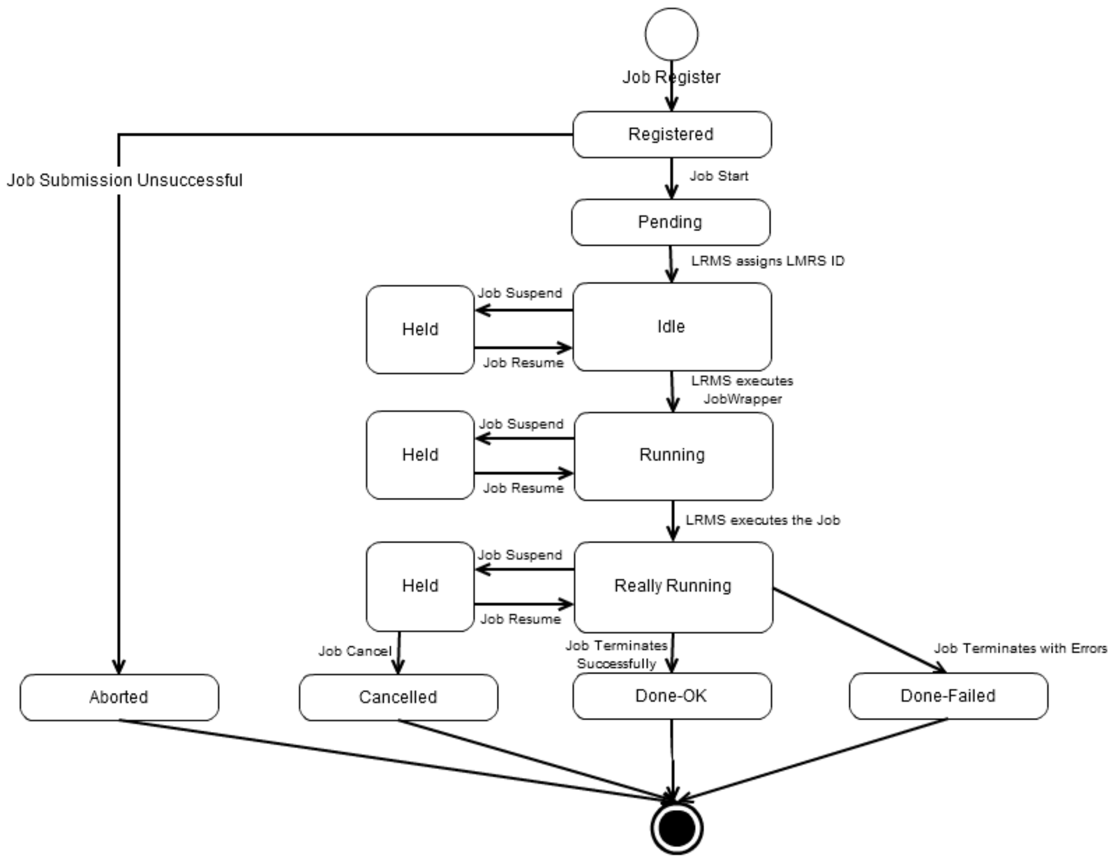


Figure 4: Job flow statuses on CREAM CE

The first step on CREAM CE job flow is job submission through EDGIExecutor. Before job is forwarded to 3G Bridge it is validated with internal and external mechanisms (ARWrapper application support check). If validity check is performed with success, input files are handled by EDGIExecutor; BridgeSubmitter translates gLite to bridge job descriptions and forwards task to the 3G Bridge, recording ID returned by the Web Service.

The next step is job status monitoring. UpdateManager periodically instructs EDGIExecutor to check the status of the jobs existing in database. According to the status, different actions are taken in different scenarios. The advance to the next step is only performed if the job status retrieved is terminal (check Figure 4).

The last step is job result management. As mentioned in previous section, different actions are taken depending on the results of job status check. In case of successful execution, job results are handled by EDGIExecutor (getResults function is called) and job is reported as finished. In case of existence of some error job is reported as failed, but attempt to get results still performed. In other cases job is reported as aborted and clear operation is performed.

2.1.3. CREAM mCE Benchmarking

To perform benchmarking, we used the EDGI Demo site, with CREAM CE as source of submitted jobs. Executing tests on this environment, allowed us to measure throughput and latency introduced by CREAM CE, as well as studying overloaded system behaviour to enable detection of problems.

2.2. The ARC mCE

One of the objectives of EDGI project is creating the bridge between middleware running on NorduGrid and existing DG. ARC mCE, similarly to the CREAM mCE, provides automatic job forwarding (through the 3G Bridge) to the Desktop Grid performing the execution of jobs. To allow the integration with the EDGI infrastructure, a DGBridge Back End was developed. This new component transforms jobs to 3G Bridge format, allowing jobs to be handled by the DG. In this way, the DG server appears as a standard ARC resource. The DGBridge back end differs from standard ARC back ends in data forwarding and by having a more complex user authentication. There were two major versions of ARC: ARC Classic and the new ARC developed within the KnowArc project, called ARC-NOX. These two versions were merged in the end of 2011, and now are called A-Rex.

2.2.1. ARC mCE Architecture

The ARC-NOX server consists of the following main components: A-Rex, Up/Downloaders, Info Service, Jobcontrol Provider and several scripted plug-ins (Figure 5).

A-Rex is the component responsible for job execution e.g. executor. In similarity to CREAM CE executor, A-Rex handles preparation of submitted jobs, creation of all necessary files for communication between different components and job execution result retrieval. When the job is being prepared, Info Service is informed by executor about the job. Once the job staging is finished A-Rex launches appropriate Jobcontrol Provider (corresponding to DGBridge Back End). Jobcontrol Provider communicates with 3G Bridge through 3G WS Client, which was developed to handle user authentication and support new I/O File methods (such as ATTIC). Whenever the job is finished, A-Rex triggers result retrieval plug-ins and cleans information related to the finished job.

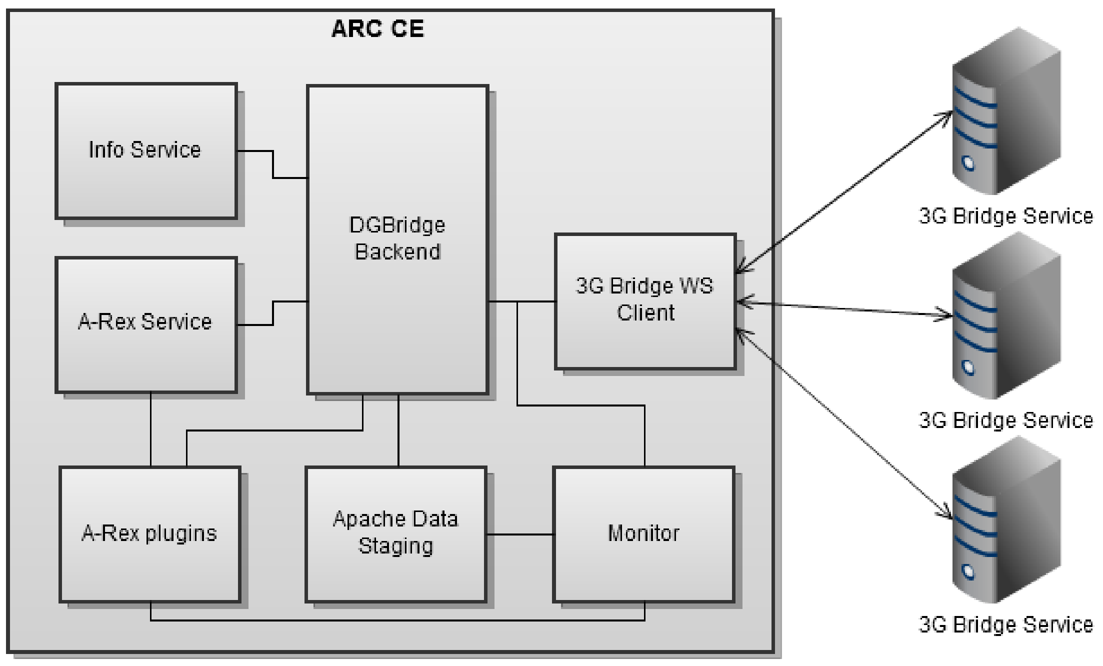


Figure 5: ARC CE architecture

2.2.2. ARC mCE Job Life Cycle

The first step in ARC mCE job flow is the submission of a job to A-Rex using one of the available ARC clients. After user authorization (which is done right after submission), job staging is performed, by translating job description to an A-Rex acceptable format (XRSL or JSDL) and by writing metadata to configured directories. If input files are specified in job description, these files are transferred (or copied in case of their location being on local file system) to the temporary directory. Upon completion of the described tasks the job is submitted to 3GBridge.

The next step in ARC job flow process is monitoring and updating job status. Info System Provider scripts periodically check job status, updating control directory with new information. The advance to the following step is performed when the job execution is finished.

The last step of the flow is managing the results of finished jobs. When a job has finished, A-Rex upload the output files to the file storage or removes them in accordance with the job description. User can download result using ARC client tools.

2.2.3. ARC mCE Benchmarking

In order to achieve the goals and measure defined metrics, benchmarking on EDGI production infrastructure was performed, namely on the EDGI Demo site, with ARC CE as source of submitted jobs. Executing tests on this environment allowed us to measure throughput and latency introduced by ARC CE, as well as studying overloaded system behaviour and detection of possible problems.

2.3. The UNICORE

UNICORE is one of the Service Grid technologies that the EDGI project intends to extend with Desktop Grid resources. UNICORE was developed in two projects funded by the German Ministry of Education and Research. Over the years, in different European projects, this technology was improved to reach the state of a solid, well-tested Grid Middleware. Nowadays, UNICORE is widely used in several supercomputer centres worldwide. This technology is open-source under the BSD licence, with its source available on SourceForge.

2.3.1. UNICORE Architecture

UNICORE is divided into three layers: client, service and system. The client layer is composed by a variety of tools to let users and their applications access Computing Elements. Different services accessed by users are part of the service layer of the UNICORE. These services are loosely coupled Java Web Services, which must be registered on the Service Registry, a crucial component of the UNICORE Computing Element. The system layer is composed by the UNICORE target systems, which are normally clusters, storage systems or some other concrete resources. All described layers make part of UNICORE Ecosystem (Figure 6).

The single point of connection of the client layer to the service layer is the Gateway. The main functionalities of this component are user the request authentication and request/response forwarding. User authentication and authorization performed by Gateway involves querying the XUUDB (UNICORE User Database) and assigning the corresponding roles. Upon arrival and after being authenticated, the user request is forwarded to the corresponding service. Whenever the service has completed the user request and the response is available it is sent to the client by the Gateway.

XUUDB is user database, which contains the mapping of the user names, certificates and roles. This database provides the system with information about user privileges on determined resource (like what services the user can access, permissions for applications, etc).

Once the user is successfully authenticated, he or she can indirectly access job and file management services, known as UAS (UNICORE Atomic Services). These services must be registered in the Service Registry. The Service Registry is responsible for the maintenance of the information about available UNICORE site functionalities. These services are hosted by the UNICORE/X component, also known as core of the Computing Element. Some of the main services are described below:

- Target System Service (TSS) is the service which provides access to native resources. It uses the eXtended Network Job Supervisor (XNJS) for submission and job management.
- Job Management Service (JMS) is responsible for single job flow control: submission, abort, pause and monitoring. It uses XNJS for request execution.
- The Storage Management Service (SMS) provides unified access in a transparent way for different available file systems.
- File Transfer Service (FTS) enables concurrent file transfer operations between generic SMSs. It supports different protocols like HTTP, UDP, GridFTP, etc.

The eXtended Network Job Supervisor (XNJS), UNICORE engine, provides functionalities which are used in implementation of User Atomic Services. This component makes use of the Target System Interfaces (TSI), enabling unified access for target systems, for resource and storage management. XNJS includes functionalities for job staging to and from local storage systems and job flow management. The Target System Interface, used by XNJS, implements access to local systems.

To make job submission to available resources transparent several modifications to the UNICORE server and service layers were made. For successful integration into the EDGI infrastructure a new Target System Interface (TSI) component was developed. There are several differences from traditional UNICORE TSIs: capability of job transformation from UNICORE format to the 3G Bridge format and new staging functionalities that allow job input/output file transfer to and from Desktop Grids. The Incarnation Database (IDB) ensures application availability on source and target systems. This is, another new UNICORE component, developed for integration with the EDGI Application Repository. A few changes in UNICORE/X were made to provide unified access the file transfer technologies (like HTTP and ATTIC). The changes made to the Computing Element allowed the successful integration of UNICORE into the existing project's infrastructure.

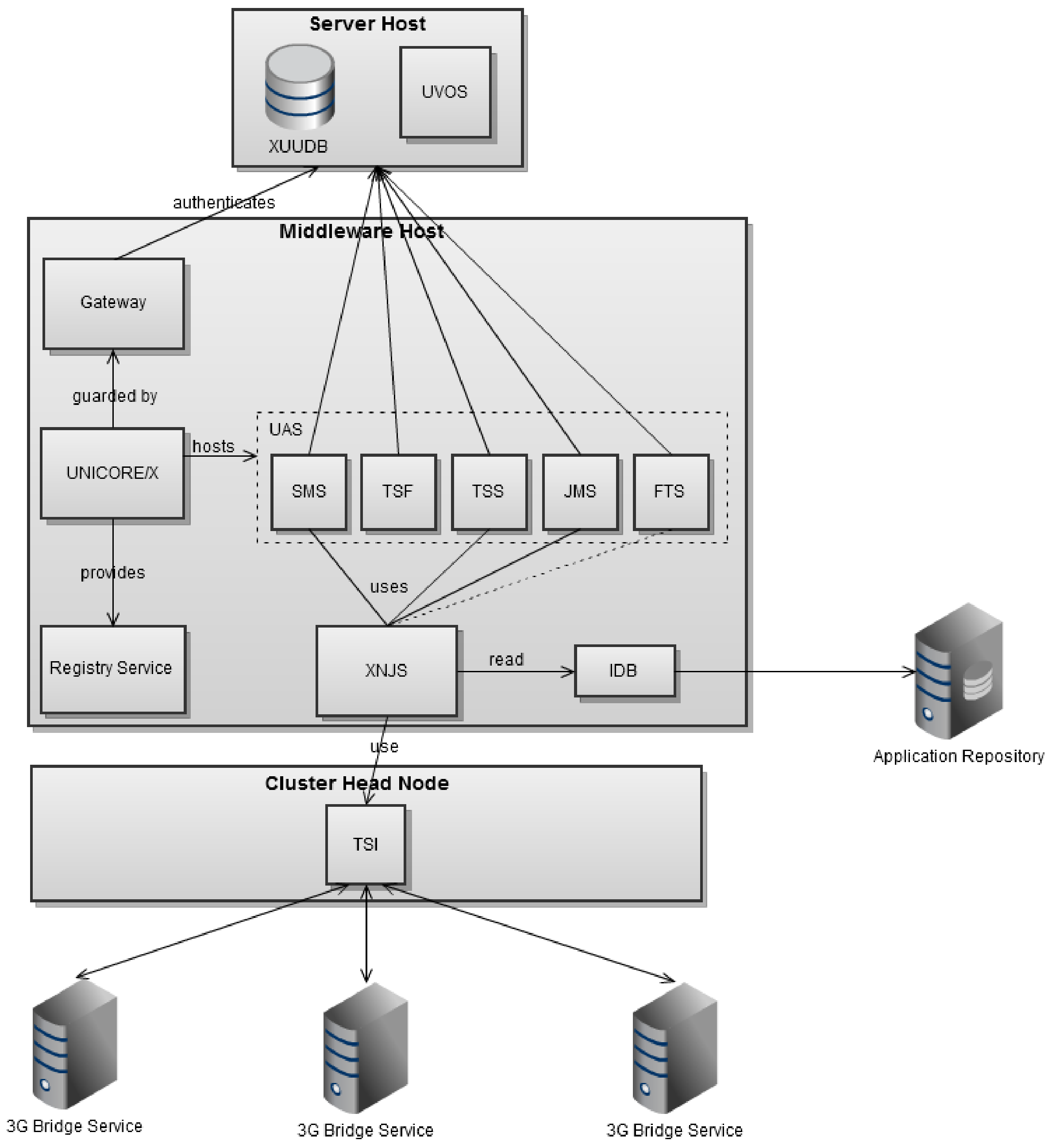


Figure 6: UNICORE CE Architecture

2.3.2. UNICORE Job Life Cycle

The job life cycle in a UNICORE site is quite similar to the already described job flows in other Computing Elements. After authentication of job submission requests by the Gateway, the job flow process starts. The XNJS, through TSI, initiates the staging in process. A working directory for the job is created. UNICORE, using its generic file transfer services, downloads all input files required by the job to this working directory.

After staging the process is finished, XNJS triggers job submission. In this phase TSI transforms the job request from UNICORE format to the 3G Bridge format. The input data is passed to the 3G Bridge using ULR pass-though technique, which consists of providing the URL pointer to the data of interest instead of including that data in the job submission request. For the 3G Bridge to be able to fetch necessary input data, the working directory of the job is being exposed via HTTP server on the UNICORE site.

If the request is successfully managed on UNICORE and 3G Bridge sites, the job status is changed from PENDING to RUNNING. The TSI periodically polls the 3G Bridge for information about job status. After successful completion of the job, a FINISHED job status is reported and the 3G Bridge provides the list of URLs to results files. The output data is downloaded by UNICORE to the job directory. After all files are successfully transferred, UNICORE removes result files from the DG site and from the 3G Bridge, minimizing the storage consumption in the infrastructure.

2.3.3. UNICORE Benchmarking

UNICORE benchmarking was performed on the infrastructure provided by the University of Paderborn, connected to the EDGI Demo site. No concurrent submissions from other Computing Elements were performed while we ran our experiments, so we could ensure that the same test conditions for different Computing Elements technologies were met. Similarly to previously executed tests on ARC and CREAM CE, our goal was measuring throughput, latency, and also observing and describing the overloaded system behaviour, as well as finding existing problems and bottlenecks in the UNICORE site.

2.4. The 3G Bridge

The 3G Bridge is the job-bridging component that has the role of gateway on the DG side server. It was developed during the previous phase of the project (EDGeS). The 3G Bridge generic interface allows different Service Grids to submit jobs to different Desktop Grids. In the EDGI project, several modifications were added to the infrastructure. Hence, this component also had several changes. The previous version of the 3G Bridge was unable to handle file references, so it had to fetch and store every input and output file locally, even if the target plug-in was able to handle the file reference. As a consequence, the 3G Bridge transferred all files twice: when the 3G Bridge has accepted the job and when the job has been sent to the destination grid.

2.4.1. 3G Bridge Architecture

The main components of the 3G Bridge are displayed on Figure 7. The first section shows WSMonitor with its Web interfaces. This service allows basic query information of the grids supported by grid plug-ins. Such data as running jobs, waiting jobs and CPU cores available can be queried using WSMonitor web service.

In the second section displayed, the component offers Web Service interface for job management. WSSubmitter is WSDL web service offering job submission, job cancel/removal, status queries and file listing functions. Download Manager is part of WSSubmitter enhancing it with the ability to fetch jobs' input files.

The third section shows core 3G Bridge elements. This block is responsible for job management on Destination Grids using grid plugging. One of the main components of this section is the QueueManager, which periodically queries plug-ins performing some actions combining it with the data received from users. Destination plug-ins are created to interact with destination grid, mainly for job management processes.

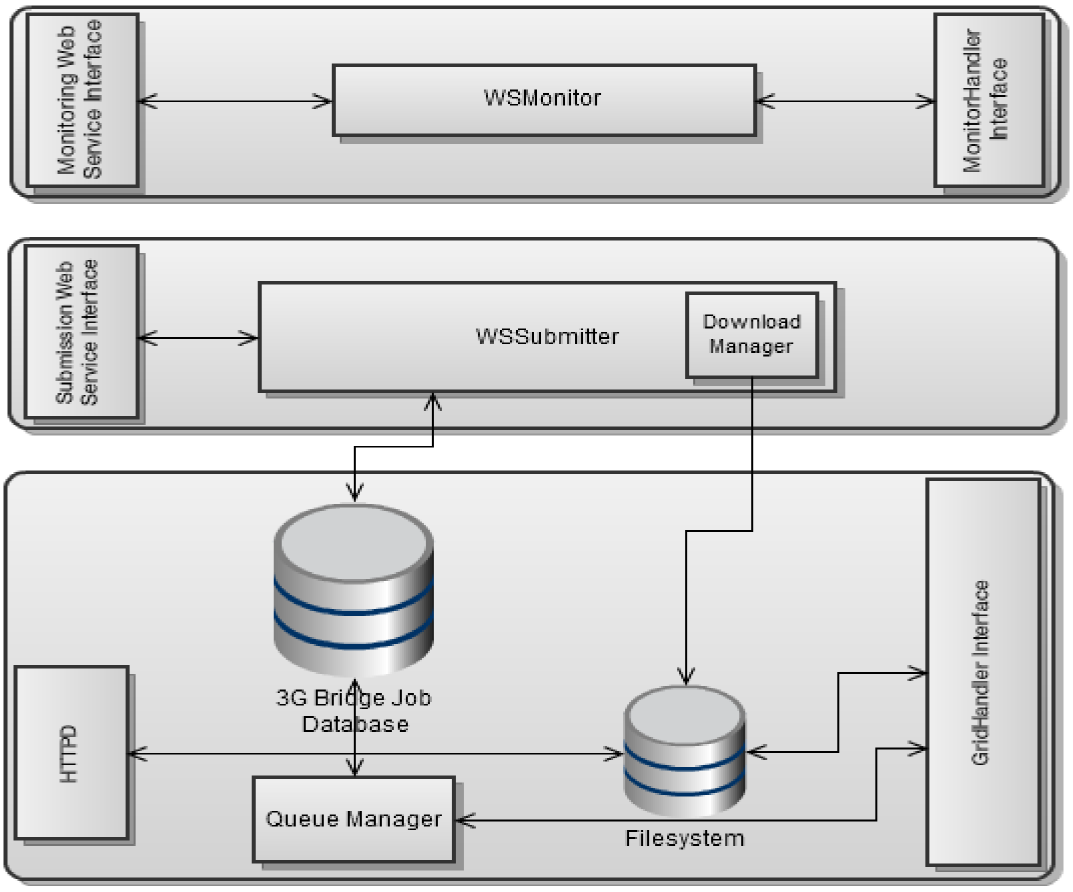


Figure 7: 3G Bridge architecture

2.4.2. 3G Bridge Job Life Cycle

The 3G Bridge has one single core 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 sources. Afterwards the job is submitted to any Desktop Grid, as long as an appropriate plug-in for that destination grid exists.

2.4.3. 3G Bridge Benchmarking

To achieve the benchmarking goals and measure defined metrics, we performed measurements with different configuration of the 3G Bridge infrastructure:

- Production infrastructure benchmarking - this configuration will be used mainly to measure throughput of 3G Bridge and analysis of the overall system performance, with detection of possible problems and bottlenecks.
- 3G Bridge isolated benchmarking - this configuration will be used to measure the latency introduced by the 3G Bridge in the overall architecture. Existing 3G Bridge configurations allow creation of an isolated environment, where the jobs will not be submitted to a DG, but rather marked as Finished as soon as job status query from some entity arrives.

3. Benchmarking Methodology

Central to the benchmarking process are the metrics we decided to take. We believe that latency and throughput are the most important criteria for grid users. The existing EDGI monitoring system, allowed us to collect all necessary data for every particular component involved in the flow of jobs. Moreover, in our experiments, we also tried to overload the system, to find EDGI's bottlenecks. This test serves to understand the impact of heavy job traffic on the whole EDGI infrastructure, as well as the impact on each particular middleware component. To understand if the system is overloaded, we used throughput and latency measurements. Moreover, with these results, we were able to tell which of the EDGI components is limiting the performance.

3.1. Metrics

As mentioned in previous section, definition of metrics was one of the crucial steps to measure EDGI infrastructure performance. These metrics are representing mathematically calculated indicators of system behaviour, related to external factors, like job submission rate.

3.1.1. Latency

Latency is a timeframe between two events of interest in the system. Data is collected in all accessible components of EDGI infrastructure providing information not only of job completion latency, but also of different components connected to the system. Timestamps of events related to job flows are collected and processed. The goal of measuring latency is to identify the delays introduced by the different middleware components of the EDGI infrastructure. We consider the following latencies:

- DG Latency – time difference between job submission detected on 3G Bridge and job completion detected on 3G Bridge.
- 3G Bridge Latency – time difference between job entry to the 3G Bridge and job completion (including the time job is waiting to enter DG).
- CE Latency – time difference between job entry to the Computing Element and job completion

3.1.2. Throughput

Throughput is the number of processed jobs per time unit. Data is collected in all points of the infrastructure. Full system information allows analysis of possible problems and bottlenecks, not only for a particular middleware, but also in the whole process. We consider the following scopes for the jobs:

- A job processed on the DG is a job entered though the 3G Bridge and that was executed and completed on the DG.

- A job processed on a 3G Bridge is a job, which entered the bridge, then was submitted to the DG and results were collected after its execution completed on the DG.

- A job processed on the CE is a job, which entered VO's queue, was submitted to 3G Bridge and results were collected after detection of its completion.

In our measurements, we also consider idle, waiting and running time of the jobs, which we analyze together with response times (latency) of the Computing Element UI commands.

3.1.3. Overloaded Behaviour

In our experiments, we also tried to overload the system, to better determine EDGI's bottlenecks. This test serves to understand the impact of heavy job traffic on the whole EDGI infrastructure, as well as the impact on each particular middleware component, involved in the job flow process. To understand if the system is overloaded, we are using results of throughput and latency measurements. With these results, we can evaluate which of the EDGI components is limiting the performance.

3.2. Test Description

There are two applications to collect all necessary data for infrastructure benchmarking (refer to Figure 8). One of the applications (Submitter Script) interacts directly with the UI provided by EDGI infrastructure component (gLiteUI or WSClient for example). The goals of Submitter Script are execution of commands related to job flow management, and collection of data available at the endpoint. Another application, Data Manager Application, interacts with system's components involved in the job flow, remotely collecting data from monitoring [19]. These reports are periodically generated and saved by the EDGI infrastructure elements in XML files, containing the data about events in each component. These events are related to the entry of the 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 report files. Besides remote data collection, the Data Manager Application merges available information, including information from the Submitter Script, and performs calculations required for each particular metric.

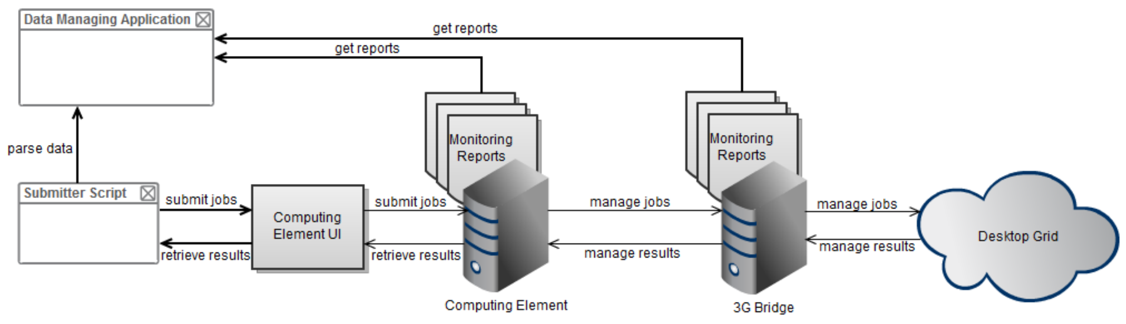


Figure 8: Developed applications' data collection scenario

3.2.1. Submitter Script

We implemented the Submitter Script in Python, because Python has powerful libraries for the work with Linux Shell. This script provides several functionalities that are crucial for the benchmarking process, related to the user authorization (which is based on validation of X509 certificate issued by Certification Authorities as LIPCA) and job flow management. During the experiment, the script is tracking data from all the operations performed for every job, saving information to the CSV file. The following high-level description (Figure 9) provides a general overview of the operation of "Submitter Script".

Several configuration parameters should be set before starting the tests. The most important are: number of tests, submission rate, number of jobs to submit and number of submitter threads in the pool. The script starts by authenticating the user with its certificate and proceeding to the main loop. The main loop is executed several times according to the configured number of tests. Before the start of every test, counters and lists are initialized to make sure that there is no interference with previously running tests, and a submitter thread pool is created. Submitter thread pool consists of configured number of threads which are waiting for data to be added to the synchronized queue (classical Producer-Worker environment). The next step in the execution of the script is the job submission process. Jobs are being added at the configured submission rate to the previously indicated synchronized queue. The submitter thread gets the job to submit from the queue and performs UI submission command adding the results of submission to the job running list. Since job submission consumes some time, to guarantee that the job submission rate does not drop below the value requested at configuration time, we recompute the job submission interval after submitting every job. The job submission rhythm is dictated by job submission interval, which is calculated dividing the time left for estimated submission end and number of jobs still to be submitted. It is important to note the size of the submitter thread pool play crucial role in successfully performed test with given submission rate. Submission process may take some time and while one submitter thread is submitting another thread can start new submission in parallel. During job submission process another application component for checking the job status is started. This component consists of the thread, which periodically queries the infrastructure component for the status of submitted jobs. The test is only concluded when submission of all jobs is finished, and all jobs are executed and finalized by the infrastructure. The results are written in CSV format into result files.

During benchmarking task "Submitter Script" was improved in several ways to meet requirements necessary to carry out this task. Major improvements were done on the assurance of the job submission rate and parallelization of the submission task by adding thread pool capability for the application with Producer-Worker scenario. In the latest updates new logging system was integrated allowing this process to be more standardized and maintainable.

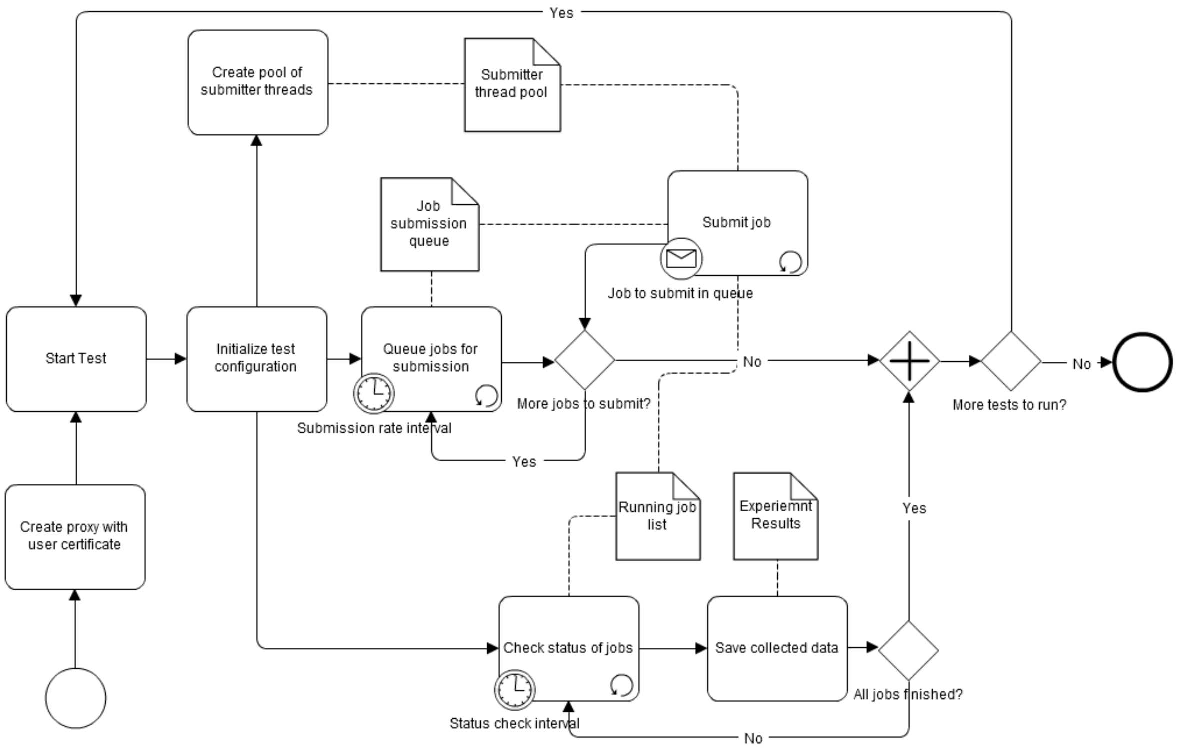


Figure 9: High-level description of operating principles of Submitter Script

There are two important result files produced by the Submitter Script:

- **res_njpt<number_of_jobs_to_submit>_njpm<number_of_jobs_per_minute>_ntest<number_of_test>.perf** (performance and monitoring file): timestamp of job submission, actual submission rate, job submission failure rate and time which script took to submit a job. This file is mainly used for analysis of current status of the tests and additional information on job submission process.
- **res_njpt<number_of_jobs_to_submit>_njpm<number_of_jobs_per_minute>_ntest<number_of_test>.txt** (job execution result file): this file contains detailed historical information about job flow for every submitted job: job ID, timestamp of job submission, timestamp of job completion, last reported status, error code, list of job statuses which it passed through with corresponding timestamp of when

status was acquired and when it was changed and completed job result download start and end timestamp.

3.2.2. Data Manager Application

We use the Data Manager Application (Figure 10) to collect data from the monitoring reports of the EDGI infrastructure components. We merge these data with information collected by the Submitter Script. The components involved in job flow process are known, since every time the job submission event appears in monitoring reports its destination component is recorded. After retrieval of monitoring information (reports are in XML format) the data is parsed and stored into database. At the end collected timestamps are merged and calculations are performed depending on the metric being measured.

Data Manager Application takes three configuration parameters: name of the file created by the submitter script, URL of the monitoring reports' directory of the closest architecturally located infrastructure component and URL of the monitoring reports' directory of the longest architecturally located infrastructure component (this parameter is optional in case for example when infrastructure is only composed by 3G Bridge and Desktop Grid it's not necessary to indicate this parameter). The first step performed by the Data Manager Application is data retrieval and aggregation from the indicated components' URL(s). Timestamps from events like job entry, job submission and job status are collected and saved into the database. After data from monitoring system is collected, the application parses the data provided by the Submitter Script, complementing the previously collected information. After all data is gathered calculations are made differing for different metrics measurements. Calculated values are saved in the results file, using CSV format.

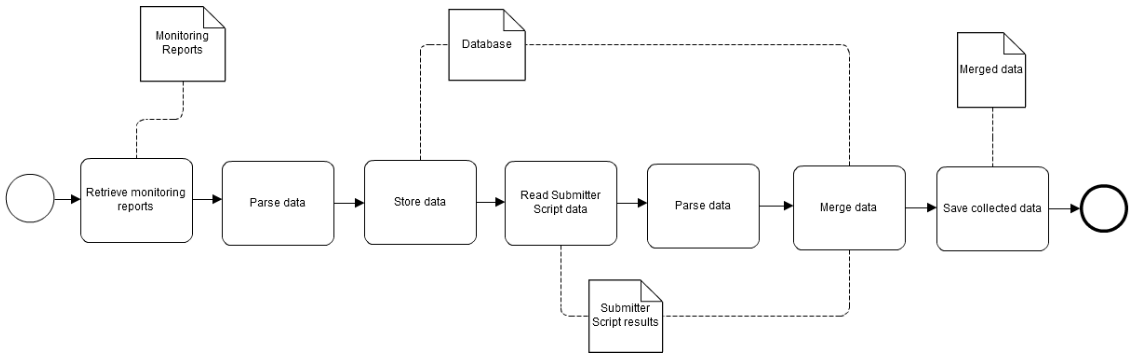


Figure 10: High-level description of operating principles of Data Manager Application

Below, we present a small part of a monitoring report file. Every event tracked by the EDGI infrastructure monitoring system starts with start-tag <metric_data> and ends with the end-tag </metric_data>. Depending on the event, different fields are reported. For example, when a job entry event is detected on a Computing Element, these elements are: the date and time of the occurred event, event type, ID assigned by middleware, application which job belongs to and source Grid. Although some of these details do not represent any interest for the DataManager application, these make part of the monitoring protocol [20], providing important information for EDGI monitoring system. In the XML presented in Figure 11, we can see all possible events related to some job: job entry to the grid, job submission from the grid and job status update[1]. For our measurements, we use the following fields: date and time of occurred event, event type, job ID and the status. By merging monitoring report data with information collected by the Submitter Script, we are able to extract performance information on every component we are interested in.

```
<report timestamp="1309257007159" timezone="GMT" version="1.1">
  <metric_data>
    <dt>2011-06-28 12:30:06</dt>
    <event>job_entry</event>
    <job_id>https://wms.ipb.ac.rs:9000/udplmiSPPWIvAM00y0dssg</job_id>
    <application>dsp</application>
    <input_grid_name>gLite/seegrid</input_grid_name>
  </metric_data>
  <metric_data>
    <dt>2011-06-28 12:30:06</dt>
    <event>job_submission</event>
    <job_id>https://wms.ipb.ac.rs:9000/udplmiSPPWIvAM00y0dssg</job_id>
    <job_id_bridge>25c34ece-72e1-4d1d-ba67-27bd97be241c</job_id_bridge>
    <status>Submitted</status>
    <output_grid_name>http://mishra.lpds.sztaki.hu:9091</output_grid_name>
  </metric_data>
  <metric_data>
    <dt>2011-06-28 12:30:06</dt>
    <event>job_status</event>
    <job_id>https://wms.ipb.ac.rs:9000/udplmiSPPWIvAM00y0dssg</job_id>
    <status>Running</status>
  </metric_data>
  <metric_data>
    <dt>2011-06-28 12:32:13</dt>
    <event>job_status</event>
```

[1] In this example we only can see Running and Finished job status, but there are many other different status depending on job execution result and Service Grid type, most of them presented in Figure 4.

```
        <job_id>https://wms.ipb.ac.rs:9000/udplmiSPPWIvAM00y0dssg</job_id>
        <status>Finished</status>
    </metric_data>
</report>
```

Figure 11: Example extract from EDGI monitoring report

3.2.3. Configuration

In all the tests, we ran the DSP application to measure the infrastructure performance. DSP is one of the applications ported to available Service Grids. The DSP job parameters were set to the minimum available values, to ensure a very fast execution of DSP, because DG performance was not the main goal of benchmarking. In the following table, we identified the main configurations that are going to be used during benchmarking tests.

		Job Complexity	Job Submission Rate
ARC	**Latency**	**Minimum Possible DSP Complexity**	**10-40 Jobs/Minute**
	Throughput	**Minimum Possible DSP Complexity**	**10-40 Jobs/Minute**
	Number of Tests per Experiment	**10**	
CREAM CE	**Latency**	**Minimum Possible DSP Complexity**	**20-100 Jobs/Minute**
	Throughput	**Minimum Possible DSP Complexity**	**20-100 Jobs/Minute**
	Number of Tests per Experiment	**10**	
UNICORE	**Latency**	**Minimum Possible DSP Complexity**	**10-40 Jobs/Minute**
	Throughput	**Minimum Possible DSP Complexity**	**10-40 Jobs/Minute**
	Number of Tests per Experiment	**10**	
3G Bridge	**Latency**	**Minimum Possible DPS Complexity**	**20-100 Jobs/Minute**
	Throughput	**Minimum Possible DPS Complexity**	**20-100 Jobs/Minute**
	Number of Tests per Experiment	**10**	

Table 2: Experiment configurations

3.3. EBench Application

To simplify the configuration of the EDGI infrastructure for DG and SG administrators we developed the EBench Application (**Error! Reference source not found.**). This web-based application was implemented in Java and requires the Glassfish Application Server[2] and MySQL[3] RDBMS installed on the machine where it runs. The application front-end was implemented using Java Server Pages technology, Twitter Bootstrap[4], Google Charts[5] and jQuery[6]. EBench allows, after simple configuration, to execute benchmarking measurements of the EDGI infrastructure components, monitoring the current status of the experiment and view the results of the execution (including latency and throughput metrics).

The EBench application makes use of the Submitter Script and of the DataManager Application. While the Submitter Script is used entirely, only some classes were taken from the DataManager Application. After the user fills the required fields on the configuration web page, script configuration and command files are generated. All are uploaded to the configured remote hosts and the experiment is started automatically by our application using the JSCH Java library[7]. The experiment status can be monitored on the status Web page, which presents the overall experiment progress and the progress of every test belonging to the experiment. After the experiment is finished, its results are presented in a results page. It is important to note that only one experiment can run for one instance of EBench application.

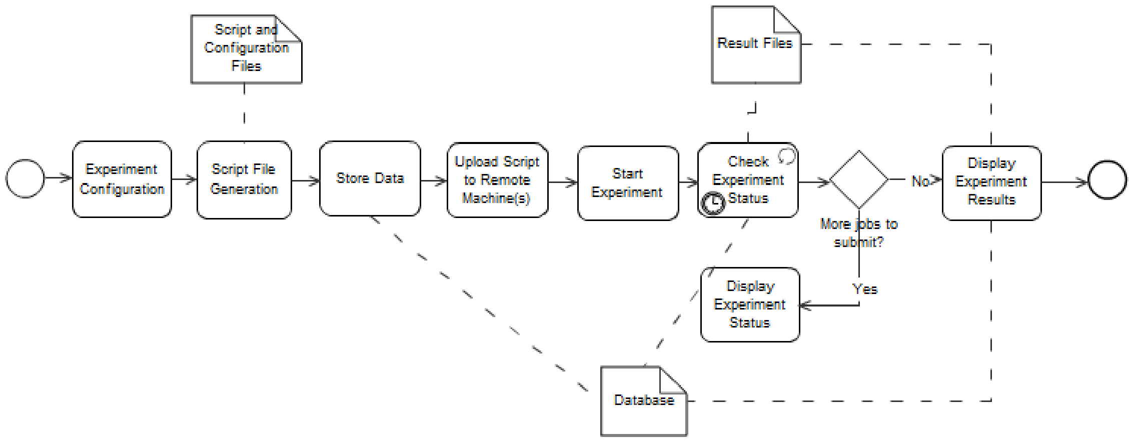


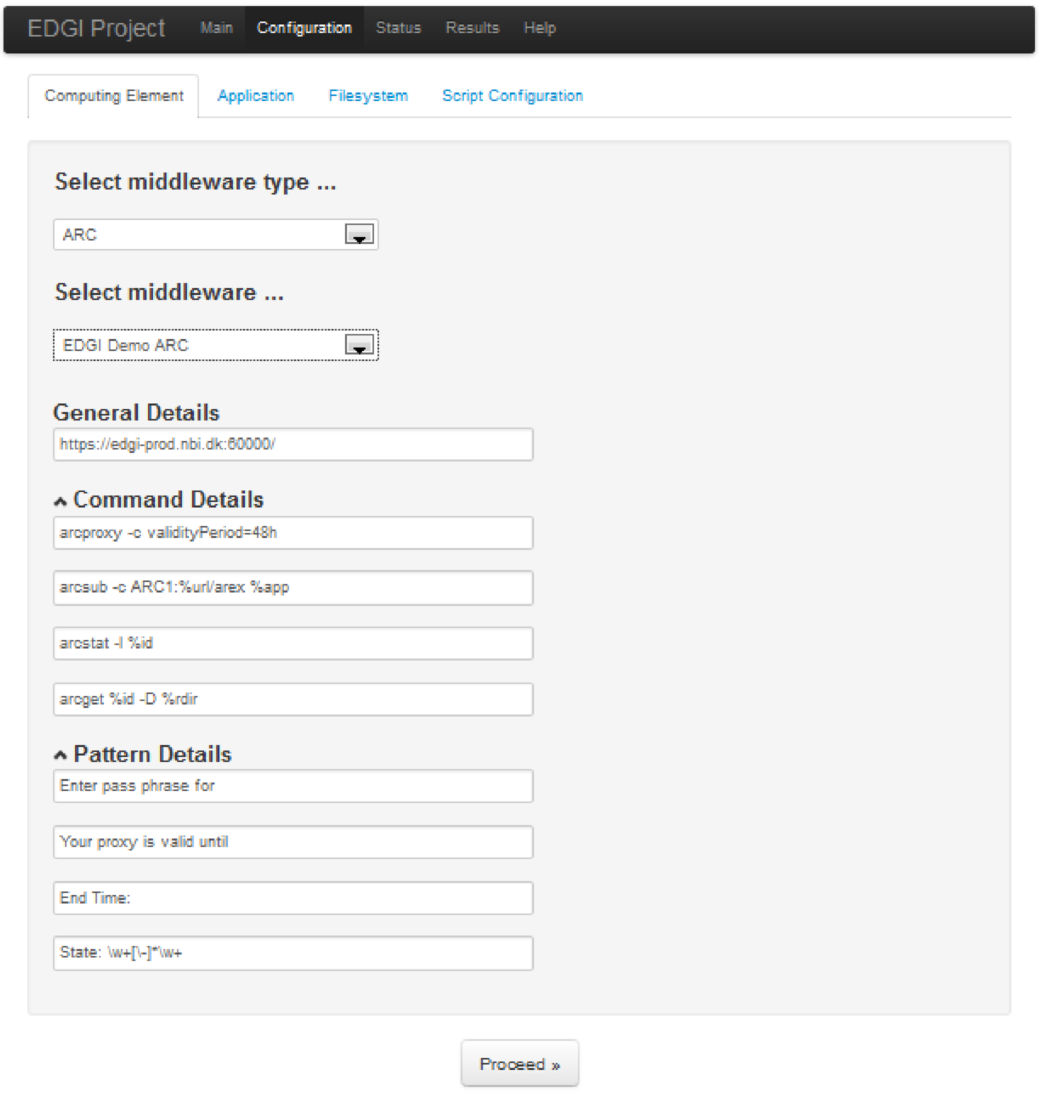
Figure 12: High-level description of operating principles of EBench Application

[2] Glassfish, online at http://glassfish.java.net/ on July 11th, 2012, Java Application Server
[3] MySQL, online at http://www.mysql.com/ on July 11th, 2012, Relational DataBase Management System
[4] Twitter Bootstrap, online at http://twitter.github.com/bootstrap/ on July 11th, 2012, HTML, CSS and Javascript tool
[5] Google Charts, online at https://developers.google.com/chart/ on July 11th, 2012, Chart Tool
[6] jQuery, online at http://jquery.com/ on July 11th, 2012, Javascript library
[7] JSCH, online at http://www.jcraft.com/jsch/ on July 11th, 2012, is a Java library, which implements SSH2 functionalities.

3.3.1. Configuration Web Page

The configuration Web page is used for the benchmarking experiment setup (Figure 13). There are several sections on this page, which must be filled to run the measurements. The Computing Element configuration uses some already pre-defined configurations (which can be changed and adjusted according to user needs), loading all necessary field values from database. Pre-defined field values would allow the user faster test configuration.

The next section is related to application configuration. Some pre-defined applications are available for the user to choose. The application configuration file (either JDL or XRSL) must be created and uploaded to the server by the user. We do not generate application configuration files for several reasons. First of all there is a very wide range of applications, which make it very hard to generalize the set of fields to be filled by the user.



Select middleware type ...

ARC

Select middleware ...

EDGI Demo ARC

General Details

```
https://edgi-prod.nbi.dk:60000/
```

Command Details

```
arcproxy -c validityPeriod=48h
arcsub -c ARC1:%url/arex %app
arcstat -l %id
arcget %id -D %rdir
```

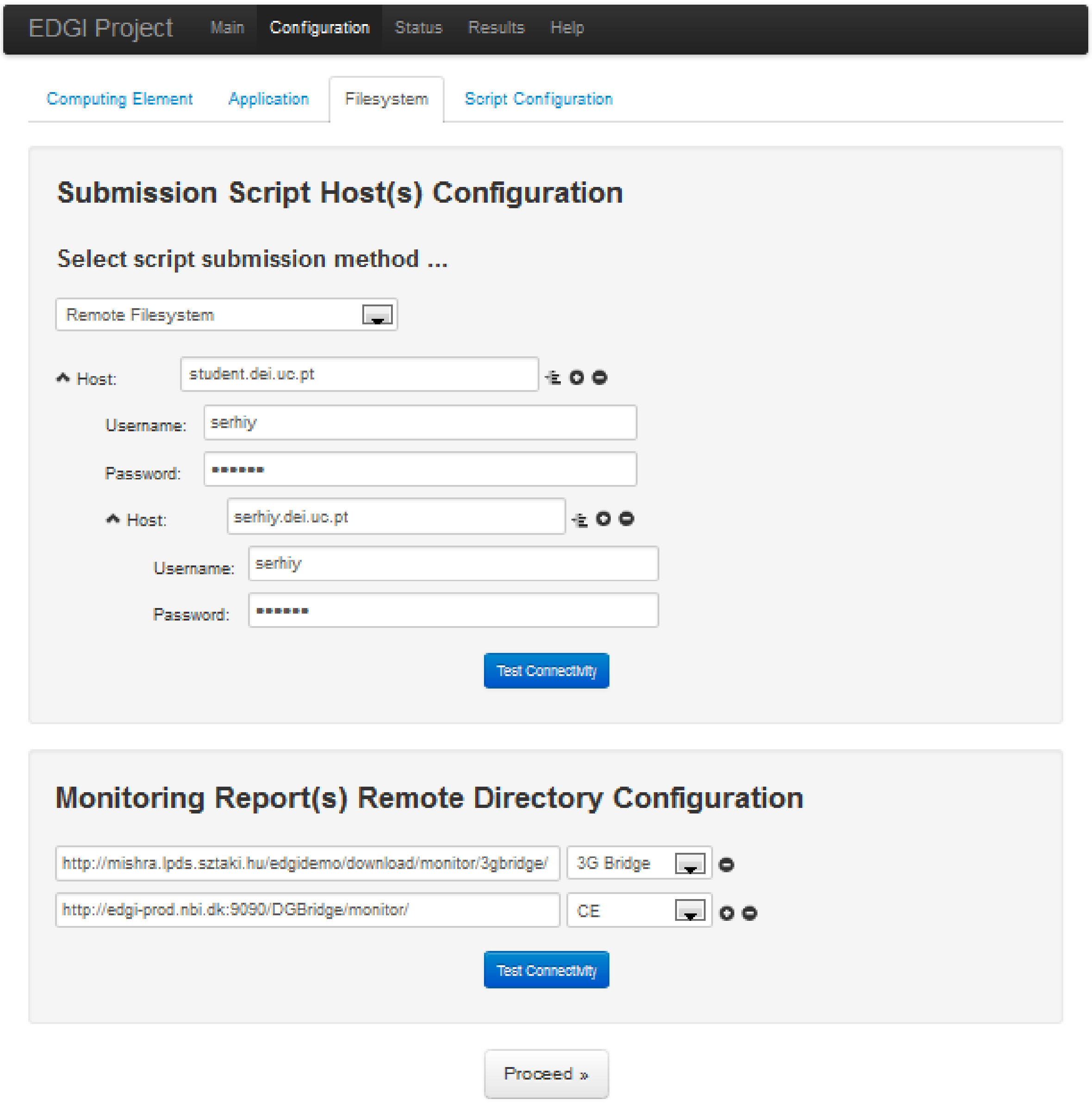
Pattern Details

```
Enter pass phrase for
Your proxy is valid until
End Time:
State: \w+[\-]*\w+
```

Figure 13: EBench Computing Element configuration section.

Application configuration is followed by the test environment configuration (Figure 14). In this section the user must specify the remote host(s) with installed target Computing Element UI to upload and run Submitter Script. It is possible to configure port forwarding in case the target host is behind a firewall. This section also includes the configuration of monitoring report directories for result extraction when the experiment is completed.

In the last section we may configure the Submitter Script parameters. User fields are used for generation of the script configuration file. Main characteristics of test are configured on this page.


EDGI Project Main Configuration Status Results Help

Computing Element Application Filesystem Script Configuration

Submission Script Host(s) Configuration

Select script submission method ...

Remote Filesystem

Host: student.dei.uc.pt

Username: serhiy

Password: ••••••

Host: serhiy.dei.uc.pt

Username: serhiy

Password: ••••••

Test Connectivity

Monitoring Report(s) Remote Directory Configuration

http://mishra.lpds.sztaki.hu/edgidemo/download/monitor/3gbridge/ 3G Bridge

http://edgi-prod.nbi.dk:9090/DGBridge/monitor/ CE

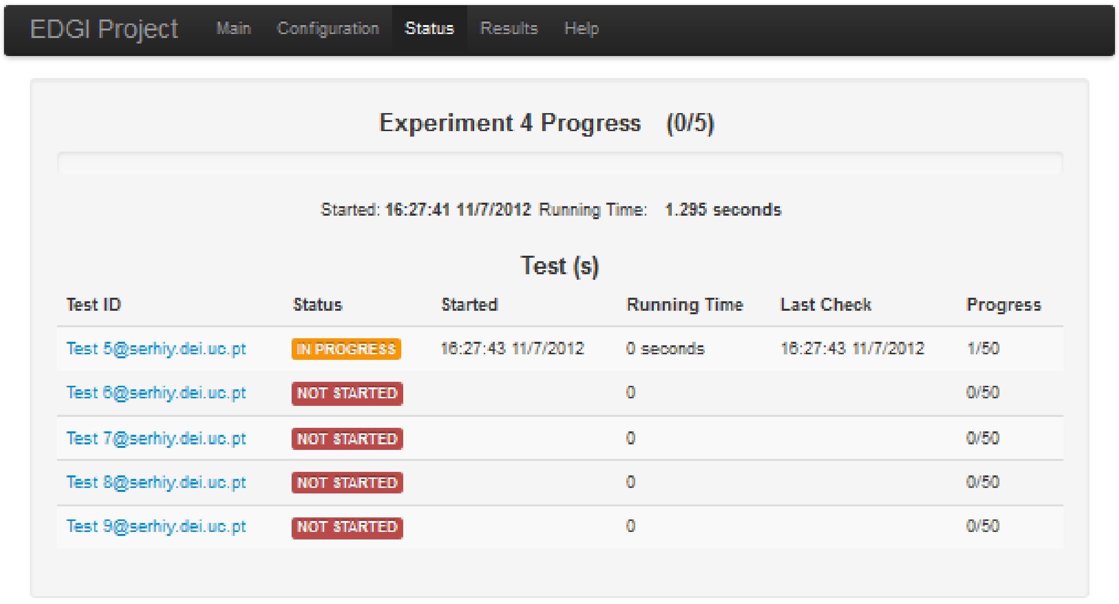
Test Connectivity

Proceed »

Figure 14: EBench remote hosts configuration section.

3.3.2. Status Web Page

The Status Web Page is used to monitor experiment and test progress. On the top of the page experiment progress is displayed (Figure 15). Experiment status information includes the number of completed tests, experiment start and running time. Below follows test progress data. Test progress information includes the test status, the number of already submitted jobs, test start and running time. The data on this page is updated in real-time.



Experiment 4 Progress (0/5)

Started: 16:27:41 11/7/2012 Running Time: 1.295 seconds

Test (s)

Test ID	Status	Started	Running Time	Last Check	Progress
Test 5@serhiy.dei.uc.pt	IN PROGRESS	16:27:43 11/7/2012	0 seconds	16:27:43 11/7/2012	1/50
Test 6@serhiy.dei.uc.pt	NOT STARTED		0		0/50
Test 7@serhiy.dei.uc.pt	NOT STARTED		0		0/50
Test 8@serhiy.dei.uc.pt	NOT STARTED		0		0/50
Test 9@serhiy.dei.uc.pt	NOT STARTED		0		0/50

Figure 15: EBench experiment progress monitoring page.

The user can also check more detailed information about running tests by clicking on test ID (Figure 16). Test status page displays graphical information about job submission rate, job failure rate and time, which jobs take to be submitted. The graphs on this page are updated in real-time.

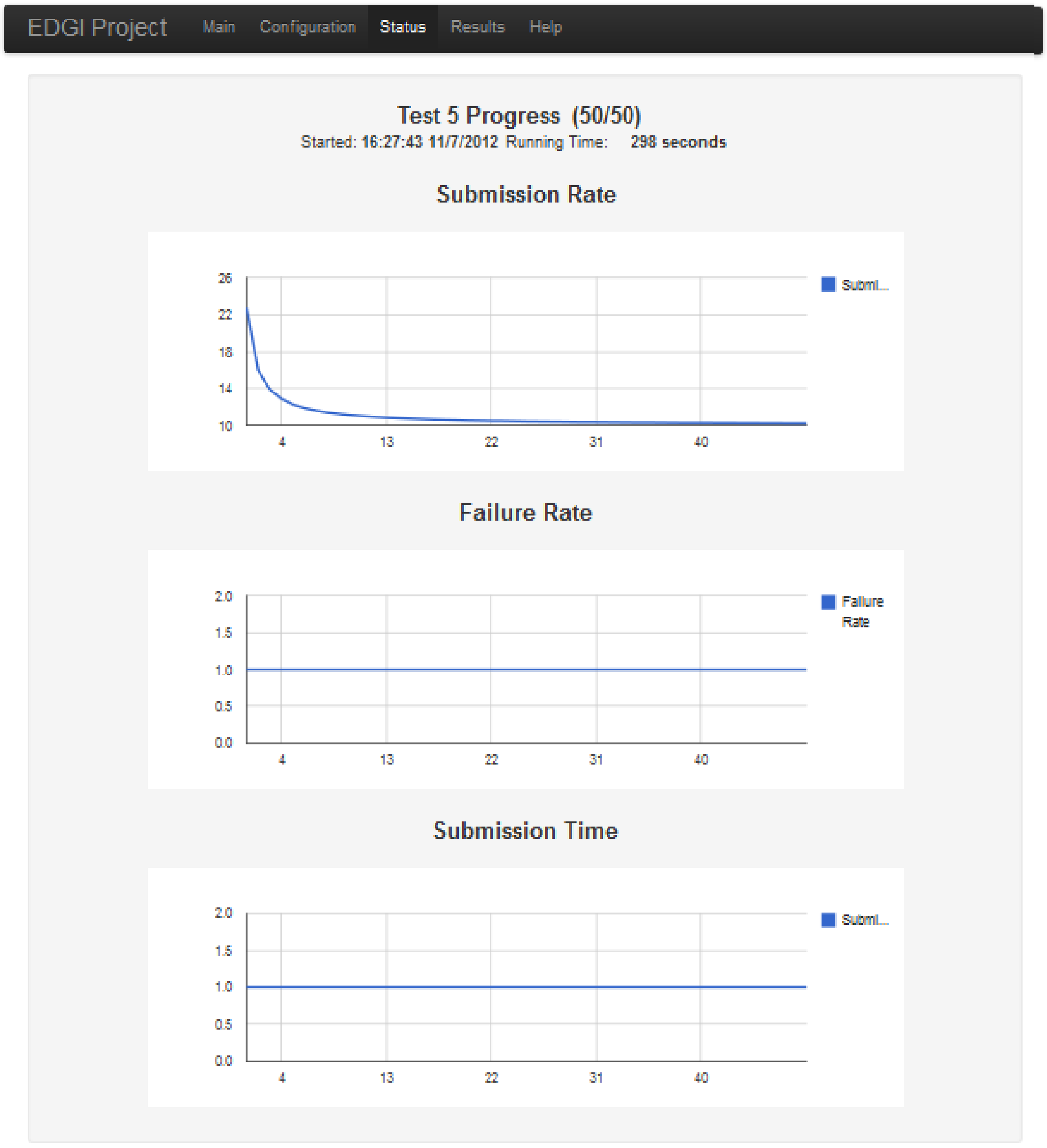
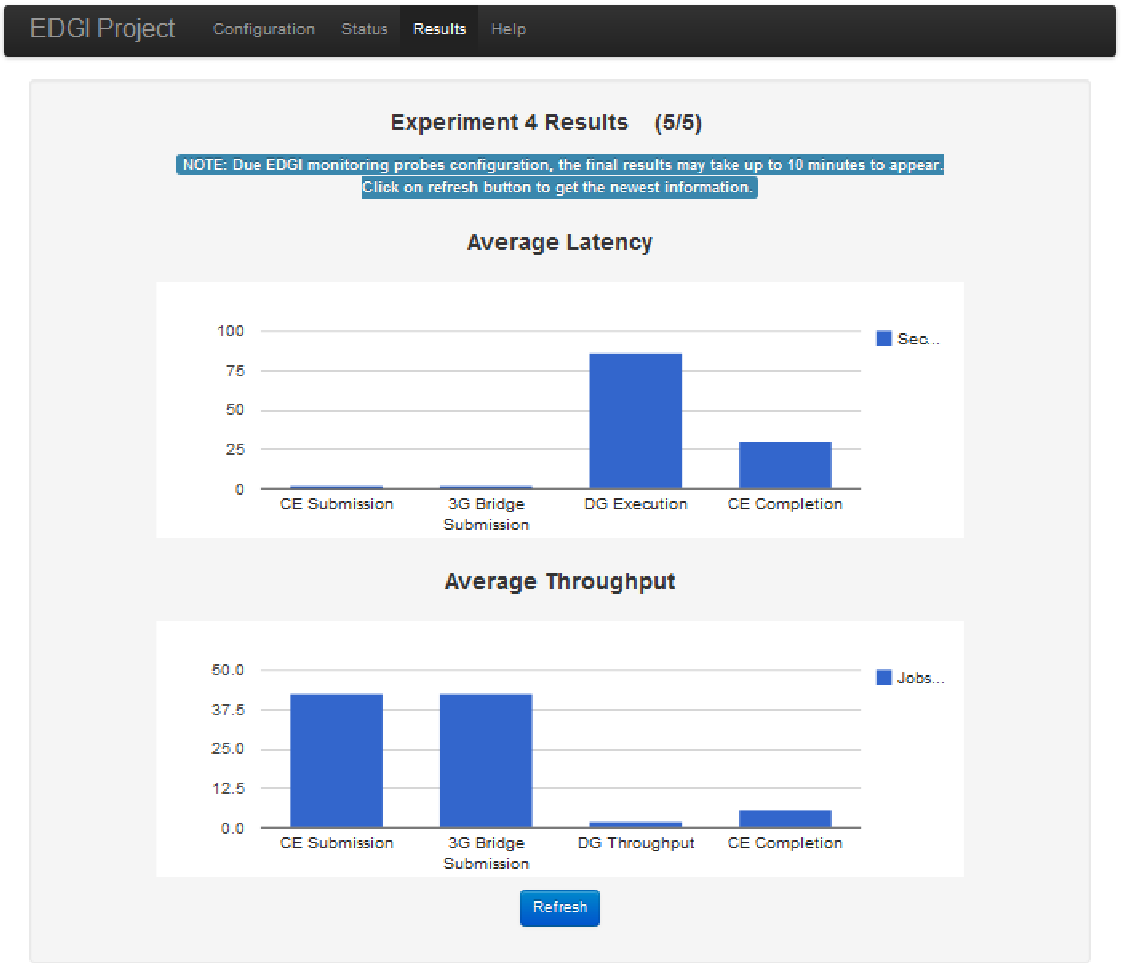


Figure 16: EBench test progress monitoring page.

3.3.3. Results Web Page

Results Web Page is used to check the final results of experiment (Figure 17). Average test latency and test throughput is calculated and graphically displayed to the user. Due the configuration of EDGI monitoring probes, it may take up to 10 minutes to obtain complete results of the tests.



Experiment 4 Results (5/5)

NOTE: Due EDGI monitoring probes configuration, the final results may take up to 10 minutes to appear.
Click on refresh button to get the newest information.

Figure 17: EBench experiment results page.

4. Benchmarking Results

4.1. Scenario

To measure latencies and throughputs, we collected, aggregated and processed many timestamps as indicated in Figure 18.

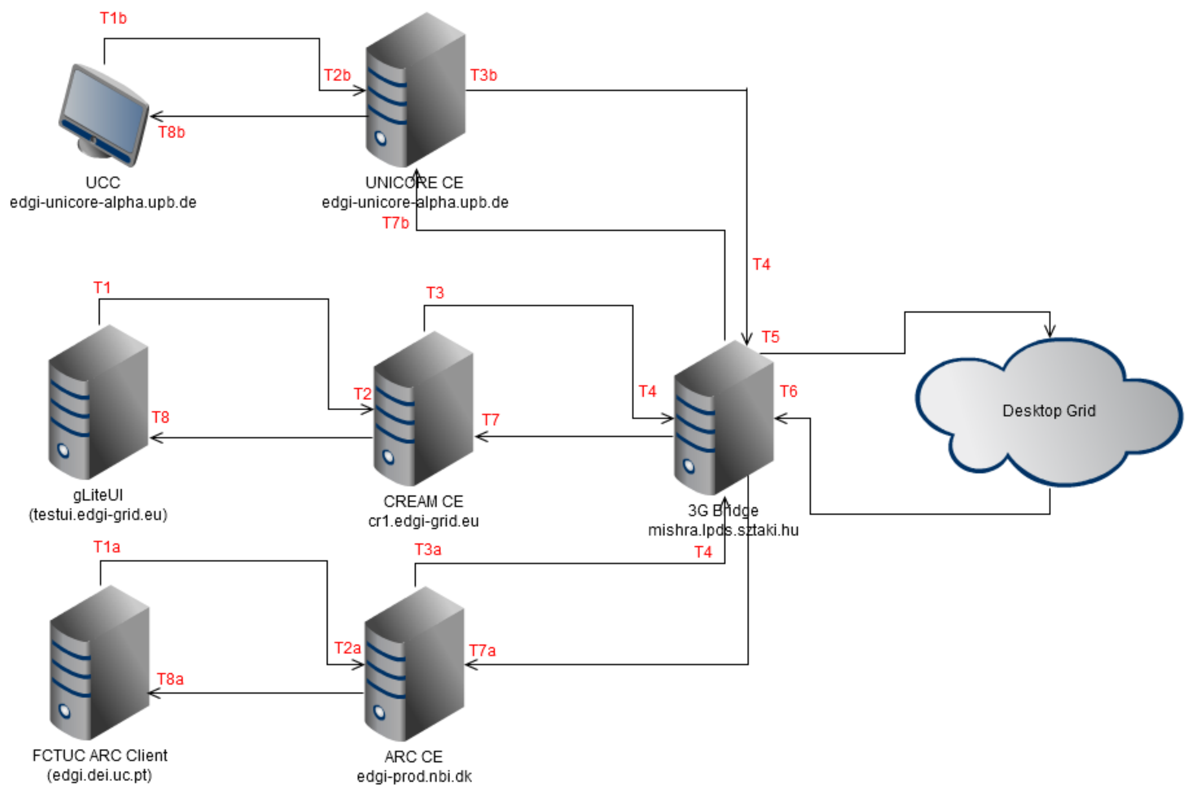


Figure 18: Timestamps collected for throughput measurement

The meaning of the timestamps collected is:

T1, T1a, T1b: timestamp of the job submission on UI (CREAM, ARC, UNICORE)

T2, T2a, T2b: timestamp of the job entry to the Computing Element (CREAM, ARC, UNICORE)

T3, T3a, T3b: timestamp of the job submission to the 3G Bridge (on CREAM, ARC, UNICORE)

T4: timestamp of the job entry to the 3G Bridge

T5: timestamp of the job submission to the Desktop Grid

T6: timestamp of the job completion on 3G Bridge

T7, T7a, T7b: timestamp of the job completion on Computing Element (CREAM, ARC, UNICORE)

T8, T8a, T8b: timestamp of the job completion on UI (CREAM, ARC)

Timestamps T1 and T8 are collected on the submitter endpoint of the infrastructure, by the Submitter application. The remaining timestamps are collected by the Data Manager Application, recurring to the monitoring data generated in agreement with the protocol defined in the first deliverable of the EDGI project, by the infrastructures components involved in the job flow process. After collecting all the previous timestamps, data processing extracts useful intermediate metrics necessary for posterior performance analysis.

4.2. CREAM CE Results

The tests for measuring throughput and latency were run on a production infrastructure called EDGI Demo, installed at SZTAKI, Budapest, Hungary. This means that we ran the tests on a configured system, without changing any values. In order to ensure correctness of results, we repeated the execution of each experiment ten 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 would allow submission of the jobs during one hour, with the defined submission rate. The submission rate varied from 20 to 100 jobs per minute, incrementing 20 for each new batch of tests. After test completion, results were collected and processed, according to the procedure described in the previous section. Average values and standard deviation of test running times were calculated. Figure 19 illustrates the first results of the average experiment execution times.

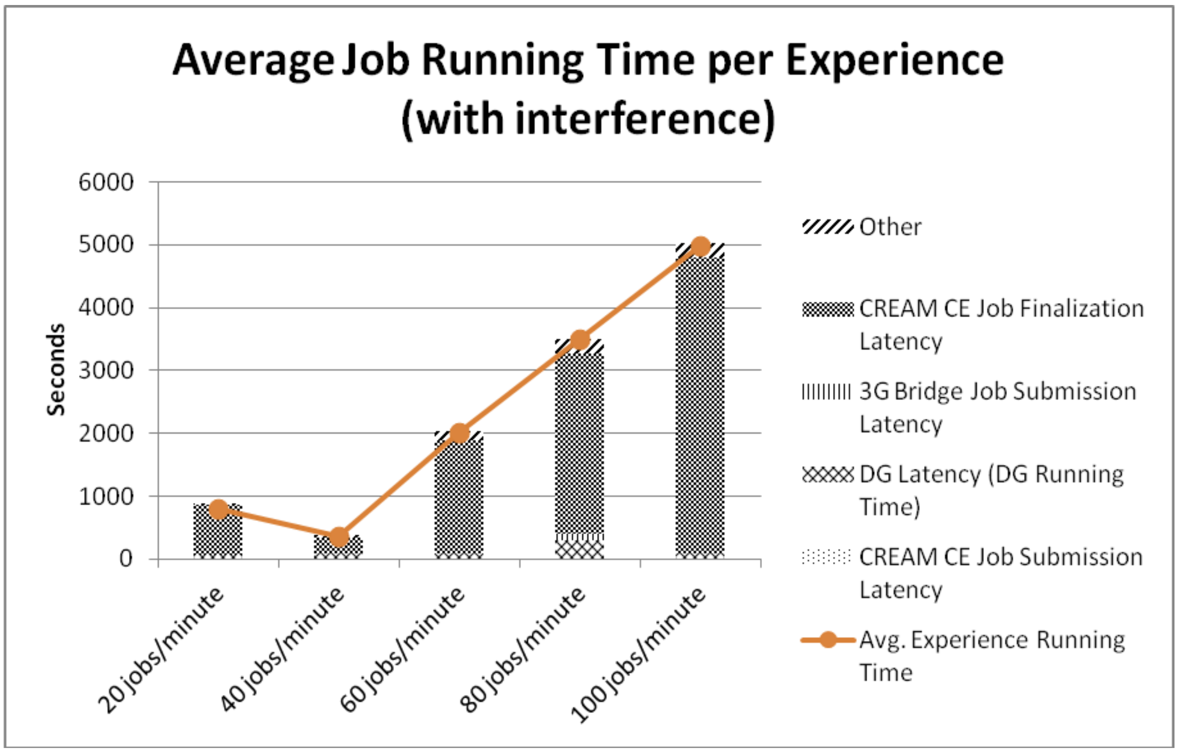


Figure 19: Average Job Running Time per Experiment (with interference)

The results of the experiment with 20 jobs/minute were highly penalized by the first of ten tests. During the first test run, the CREAM CE performance was considerably lower than normal, comparing to the following tests. Analysis of the monitoring reports has shown that in this time there were additional jobs running on the EDGI Demo site, other than those we submitted in this test (these jobs were submitted by our own script at an earlier time). We decided to remove this experiment from the infrastructure throughput analysis.

Another problem was spotted in the tests number 7 and number 8, in the experiment with submission rate of 80 jobs/minute. In this case the source of the problem was the usage of Desktop Grid resources by another application. After analysing monitoring reports of the BOINC, we were able to spot the workunits of "fusion" application running in the same time with the tests number 7 and number 8. Due to the nature of this problem it was decided to keep the results of the test number 7 and number 8 in following data analysis. A new figure was created without the interference introduced by test number 1 in the 20 jobs/minute (Figure 20). This figure allows us to observe a drastic performance drop of the CREAM CE, when it is finalizing jobs.

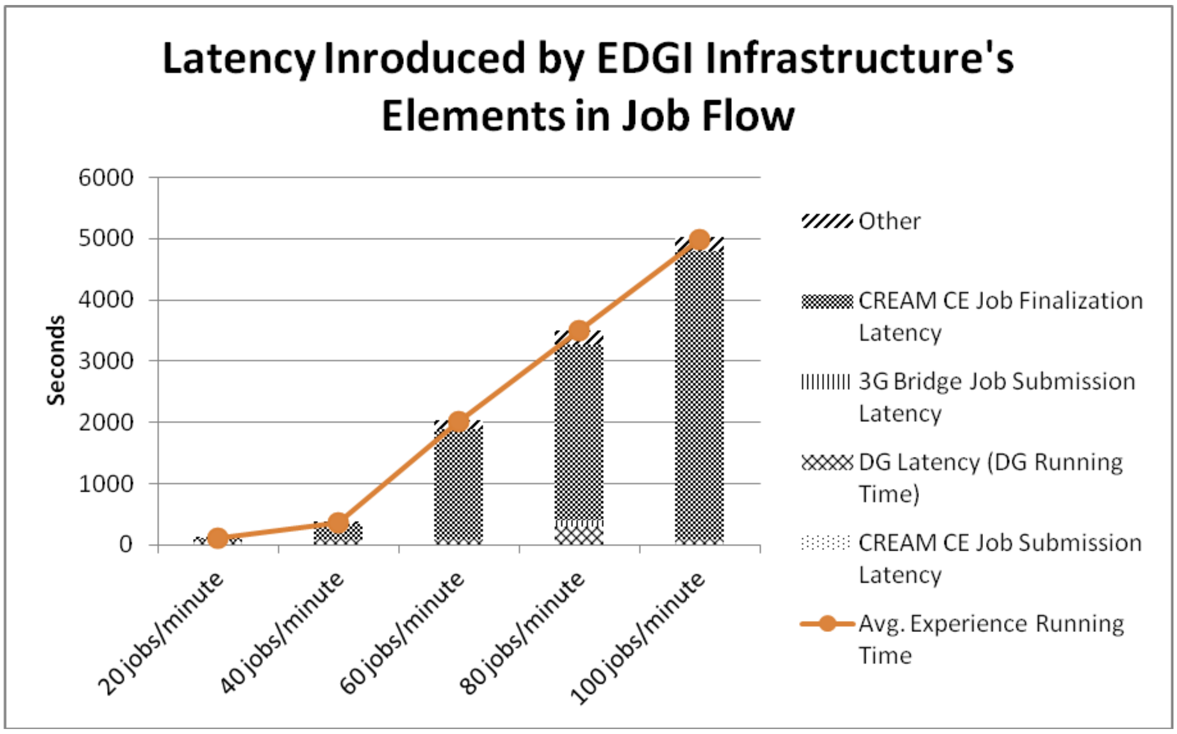


Figure 20: Average Job Running Time per Experiment (without interference)

4.2.1. CREAM CE Throughput

In the EDGI Demo site, we identified two different throughputs that are relevant for CREAM CE: job submission throughput, and job finalization throughput. These throughputs were measured using the data collected by the submitter application and data obtained from monitoring reports. To measure the overall throughput of the elements involved in the job flow process, we used the already collected gLite UI's submission timestamp (T1) and gLite UI's finish timestamp (T8) of every successfully submitted job and applied the following formula:

$$\text{Throughput} = \frac{\text{Number of Finished Jobs}}{(\text{Timestamp of Finished Job} - \text{Timestamp of the First Submitted Job})}$$

In words, the overall throughput until the last job that finished (T8) is the number of jobs that were completed so far, divided by the current timestamp (T8) minus the timestamp of the first job submitted. To compute throughputs in several points of the EDGI infrastructure, we used the formulas of Table 3. The denominator of the fraction refers to the current timestamp minus the minimum timestamp. By considering different points of the architecture, we can have a more extensive overview of the system, to detect its bottlenecks.

Calculated Throughput	Formula	Units
Overall Throughput	$\frac{\text{Number of Completed Jobs}}{T8 - \text{MIN}(T1)}$	**Jobs/Second**
CREAM CE Preparation Throughput	$\frac{\text{Number of Submitted to 3G Bridge Jobs}}{T3 - \text{MIN}(T2)}$	**Jobs/Second**
3G Bridge Preparation Throughput	$\frac{\text{Number of Submitted to DG Jobs}}{T5 - \text{MIN}(T4)}$	**Jobs/Second**
Desktop Grid Throughput	$\frac{\text{Number of Completed Jobs}}{T6 - \text{MIN}(T5)}$	**Jobs/Second**
CREAM CE Finishing Throughput	$\frac{\text{Number of Completed Jobs}}{T7 - \text{MIN}(T6)}$	**Jobs/Second**

Table 3: EDGI Demo Components' Throughputs and corresponding Calculation Methods

Using these formulas, we were able to calculate the average throughput of the system. It is important to note while calculating average throughput we omitted 15% of the initial results considering such period as warm up. Figure 21 shows the evolution of the overall throughput, thus showing the need to exclude a warm up period from the analysis we do in this document.

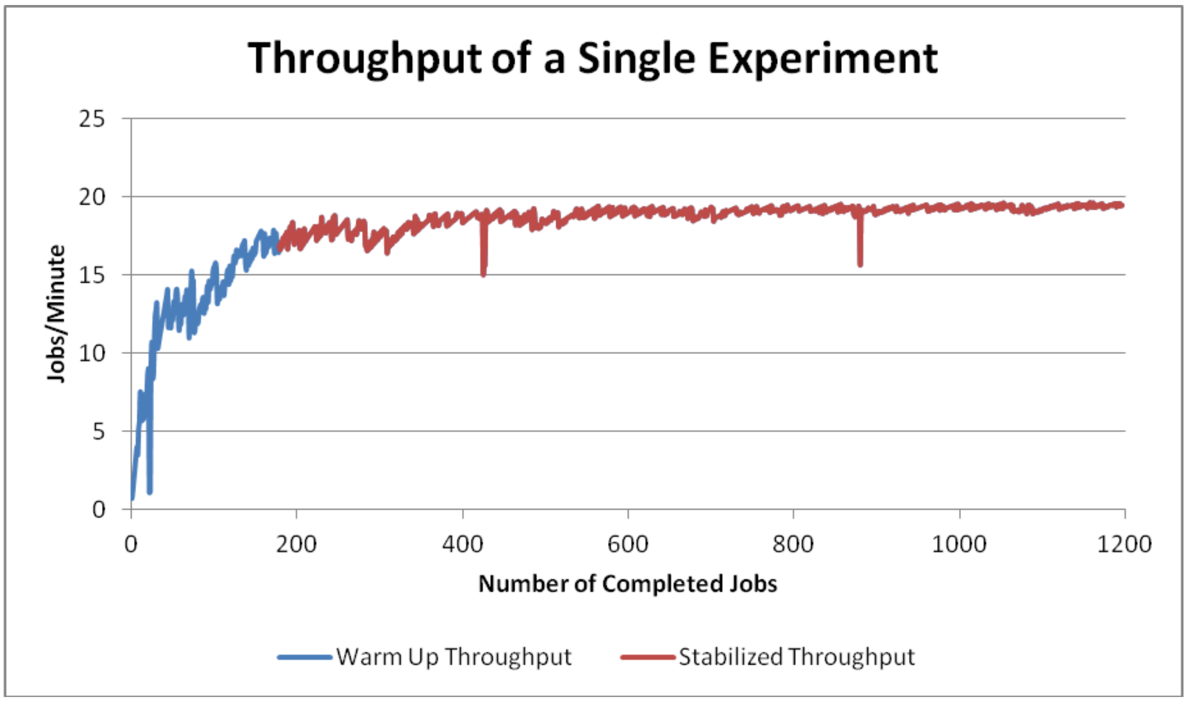


Figure 21: Punctual Throughput of Random Experiment

In Figure 22, we illustrate the overall throughput of the system as a function of the job submission throughput.

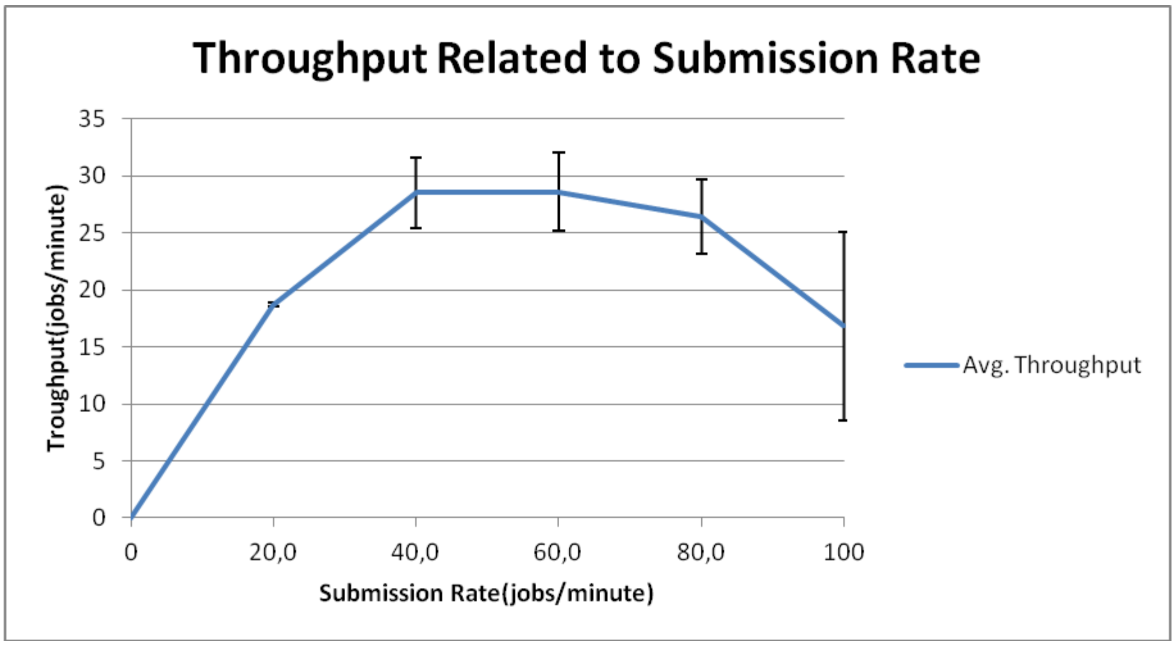


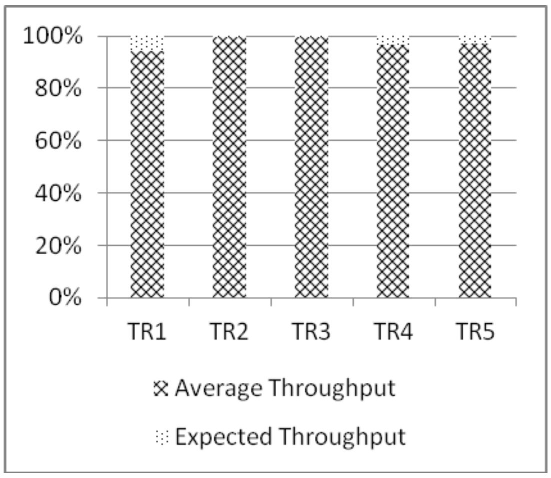
Figure 22: Average Throughput per Experiment

One should notice the loss in the throughput for higher submission rates. To understand what causes this effect, we proceeded to analyse the throughput of each particular element of the system. Data of interest was extracted from monitoring reports and processed accordingly to the formulas indicated in the Table 3. In the average throughput per experiment we ignored, once again, the 15% initial results, to ensure that the “warm up” will not influence the final results.

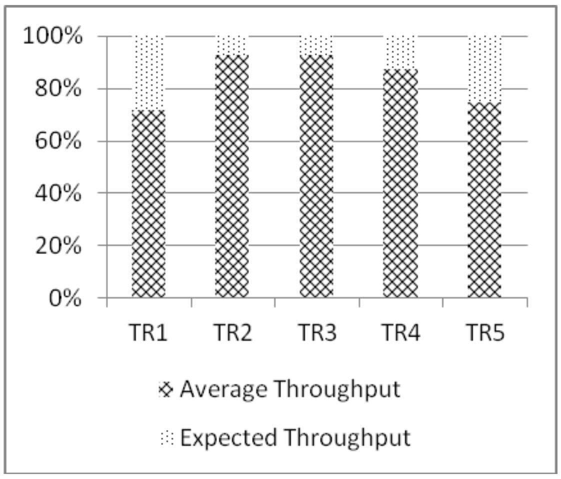
In the following graphs (see Figure 23), results of our work are shown (the meaning of X-Axis can be found in the Table 4). The “Expected throughput” is the throughput that we would expect from a system without job delays, i.e., it is the same as the submission throughput. After careful analysis of the EDGI Demo site elements throughput we were able to identify potential bottleneck of the system, which is TR5, throughput of the CREAM CE job finalization. Some of the graphs on Figure 23 allowed us to conclude that there is direct dependency of the overall job flow process throughput on the CREAM CE job finalization part.

TR1	**Average throughput measured on gLiteUI**
TR2	**Average throughput of CREAM CE job submission**
TR3	**Average throughput of 3G Bridge job submission**
TR4	**Average throughput of Desktop Grid job processing**
TR5	**Average throughput of CREAM CE job finalization**

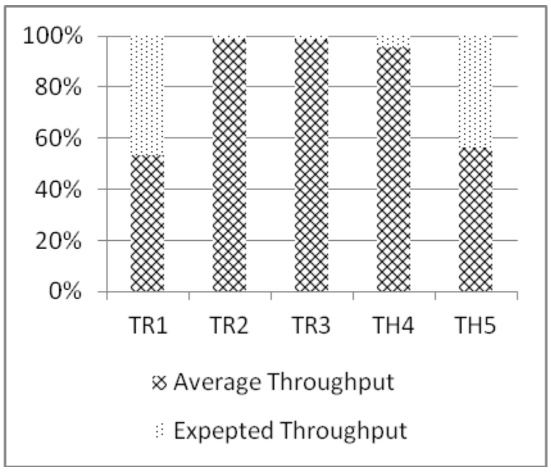
Table 4: The Meaning of the X-Axis of the “Average Current/Expected Throughput” Graphs



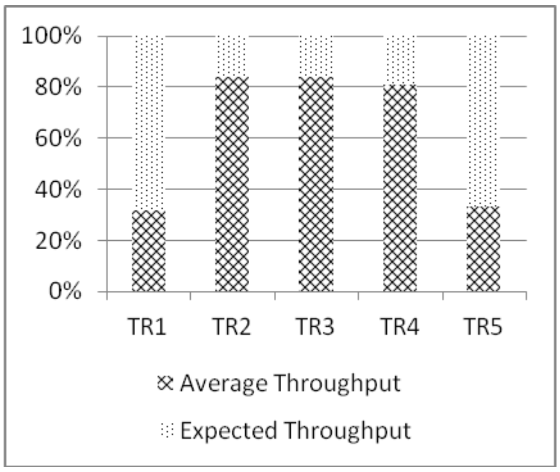
Experiment 1: Submission Rate 20 Jobs/Minute



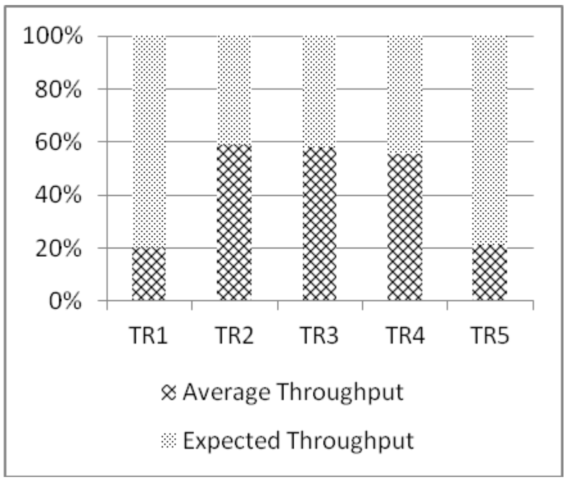
Experiment 2: Submission Rate 40 Jobs/Minute



Experiment 3: Submission Rate 60 Jobs/Minute



Experiment 4: Submission Rate 80 Jobs/Minute



Experiment 5: Submission Rate 100 Jobs/Minute

Figure 23: Average Current/Expected Through Relation in EDGI Infrastructure Components

To understand if CREAM CE finalization could be the bottleneck of the EDGI Demo site, we analysed the source code of the CREAM CE. From the architectural analysis of this computing element, we realized that there is an element responsible for job status update, called UpdateManager. UpdateManager is a Thread that retrieves from database jobs with configured statuses (IDLE, PENDING, RUNNING) and queries the 3G Bridge for the actual status of the job (a Web Service call is made). After getting response from the 3G Bridge the UpdateManager starts a status handling process, including updates of the database and event logging. When the number of jobs in the list is small, this model might be working very effectively, but when the number of jobs in the list increases, the job status check processes consumes much more resources. The following graph (Figure 24) allowed us to understand that the number of jobs for status check in CREAM CE grows with higher submission rate and only starts to decrease when the submission is over (after 3600 seconds). The results and code analysis allowed us to conclude that the CREAM CE job status check is currently the source of the scalability limitations on CREAM CE and the bottleneck of the EDGI Demo site.

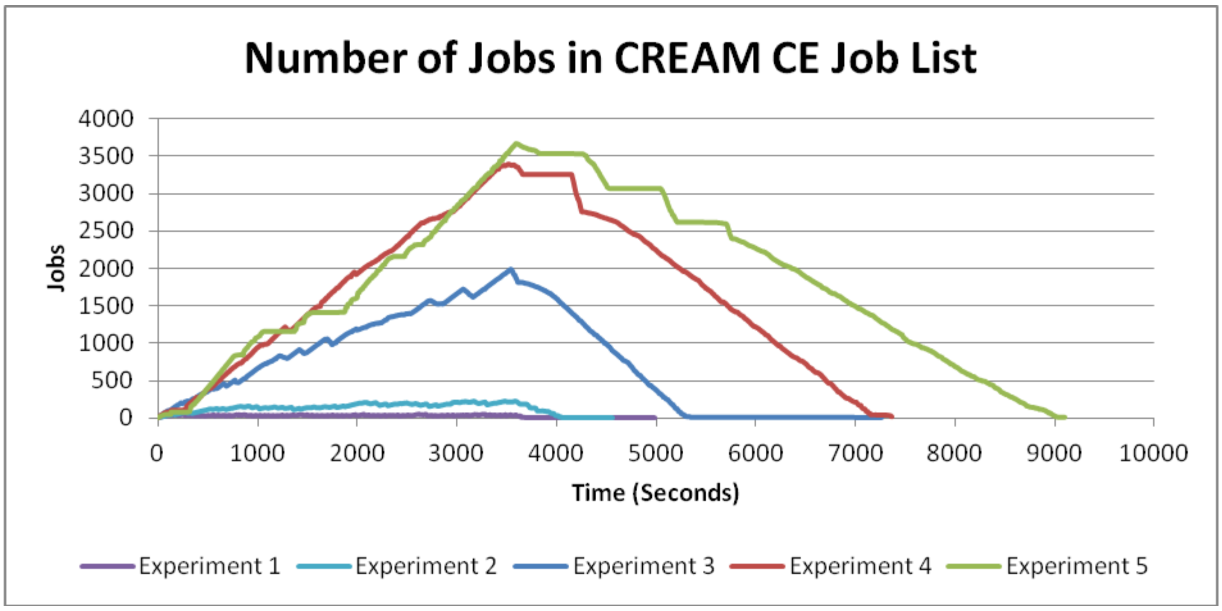


Figure 24: Number of Jobs in CREAM CE Job List (for status check)

The CREAM CE job finalization is not the only bottleneck identified with benchmarking tests. Analysing results of the average throughput of the CREAM CE job submission (TH2) we identified that there was a throughput break in experiment number four. The experiment number five confirmed also suffers from this reduction. In both cases, CREAM CE is unable to keep up with job submission rate. During execution of these two experiments, we were able to note that the number of failed jobs was increasing with job submission rate. The main reason for job failure was connection timeout on the execution of the job submission command. In Figure 25, we provide an overview of the jobs that the script fails to submit against CREAM CE job throughput. The figure clearly displays the growth of failure rate when job submission rate is increased.

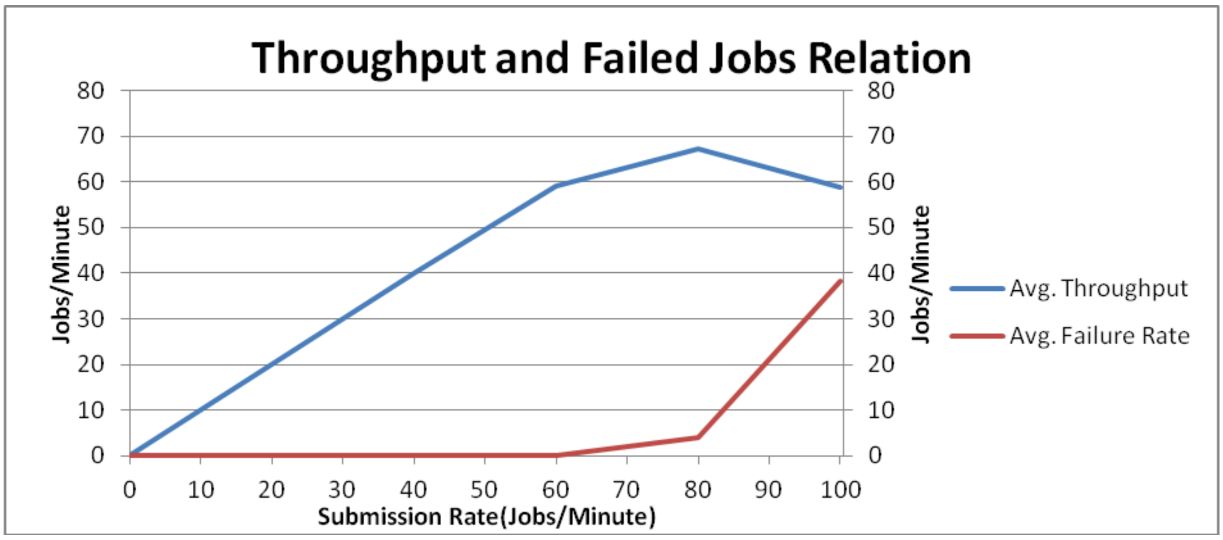


Figure 25: Throughput and Failed Job Relation

After analysing job failure rate and average throughput values we were able to conclude that throughput decreases in relation to job submission rate, due to jobs that the script fails to submit. It is important to note that submission failure rate is not constant in every point of the test having low and peak periods.

4.2.2. CREAM CE and Other Latencies in EDGI Demo

We identified two latencies introduced by CREAM CE: job submission latency and job finalization latency. The latency analysis of the other EDGI Demo elements (3G Bridge and Desktop Grid) was also performed for comparison purposes and possible infrastructure problems detection (Table 5).

Calculated Latency	Formula	Units
Job Running Time (Overall Latency)	**T8 – T1**	**Seconds**
CREAM CE Preparation Latency	**T3 – T2**	**Seconds**
3G Bridge Preparation Latency	**T5 – T4**	**Seconds**
Desktop Grid Running Time (DG Latency)	**T6 – T5**	**Seconds**
CREAM CE Finishing Latency	**T7 - T6**	**Seconds**
Other (Network Latencies, etc)	**Job Running Time – Components' Latencies**	**Seconds**

Table 5: EDGI Demo Components' Latencies and corresponding Calculation Methods

After processing collected data, we proceeded to analysis of the average latency introduced by the EDGI Demo components, during the execution of different experiments with different job submission rate.

Experiment 1: Submission Rate 20 Jobs/Minute				
	Average	Mininum	Maximum	Standart Deviation
Avg. Experiment Running Time	112,37	98,63	155,61	17,37
CREAM CE Job Submission Latency	0,40	0,31	0,84	0,17
3G Bridge Job Submission Latency	2,85	2,27	6,44	1,35
DG Latency (DG Running Time)	60,63	52,10	77,62	7,21
CREAM CE Job Finalization Latency	39,22	31,60	59,98	8,70
Other	9,27	8,78	10,73	0,59
Experiment 2: Submission Rate 40 Jobs/Minute				
	Average	Mininum	Maximum	Standart Deviation
Avg. Experiment Running Time	363,83	260,60	850,63	178,41
CREAM CE Job Submission Latency	0,75	0,37	3,50	0,97
3G Bridge Job Submission Latency	3,49	2,35	13,12	3,38
DG Latency (DG Running Time)	69,72	58,58	101,32	12,36
CREAM CE Job Finalization Latency	262,90	175,51	693,25	158,84
Other	26,97	21,29	39,45	6,35
Experiment 3: Submission Rate 60 Jobs/Minute				
	Average	Mininum	Maximum	Standart Deviation
Avg. Experiment Running Time	2018,62	1523,95	3204,90	527,82
CREAM CE Job Submission Latency	1,05	0,59	3,09	0,82
3G Bridge Job Submission Latency	7,13	2,42	43,35	12,78
DG Latency (DG Running Time)	68,72	57,29	111,25	15,90
CREAM CE Job Finalization Latency	1796,15	1332,64	2896,55	485,83
Other	145,58	119,99	237,67	33,99
Experiment 4: Submission Rate 80 Jobs/Minute				
	Average	Mininum	Maximum	Standart Deviation
Avg. Experiment Running Time	3490,07	2873,47	5000,61	646,74
CREAM CE Job Submission Latency	1,67	0,91	4,90	1,28
3G Bridge Job Submission Latency	98,81	2,90	675,69	221,04
DG Latency (DG Running Time)	287,30	61,87	1588,28	507,74
CREAM CE Job Finalization Latency	2861,87	2485,29	3354,96	327,00
Other	240,43	204,01	254,12	30,99
Experiment 5: Submission Rate 100 Jobs/Minute				
	Average	Mininum	Maximum	Standart Deviation
Avg. Experiment Running Time	4979,44	3977,65	6477,41	790,52
CREAM CE Job Submission Latency	1,67	2,28	4,46	0,67
3G Bridge Job Submission Latency	7,07	3,66	17,57	5,00
DG Latency (DG Running Time)	78,69	69,02	95,21	9,89
CREAM CE Job Finalization Latency	4696,42	3774,93	6437,37	872,52
Other	222,72	126,87	300,48	61,05

Table 6: Latency Results

The first thing we have noticed was the difference of job's average running time in the different experiments. On the one hand we can expect behaviour that a higher load will cause a higher completion time, but, on the other hand, we needed to study the completion time to identify problems on any of the EDGI Demo site components. To better understand the latency introduced by every component in the job flow process, we illustrate the latencies in Figure 26.

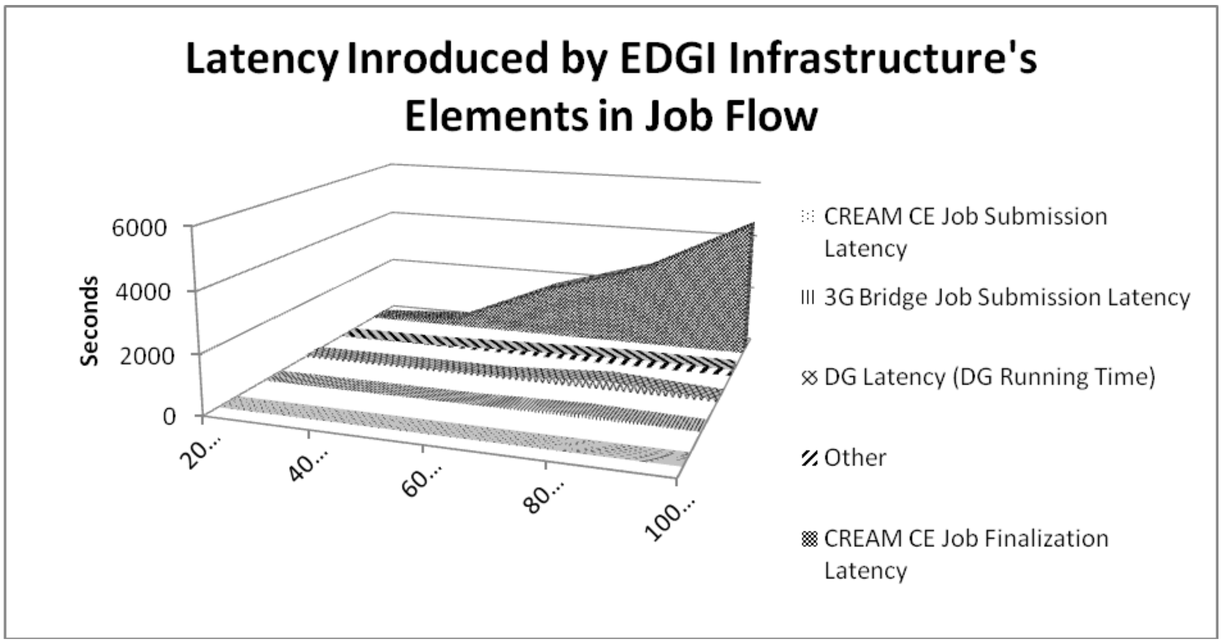


Figure 26: Latency of EDGI Demo Components in Job Flow Process

The first thing we noticed was the drastic increment of CREAM CE job finalization time in comparison with other calculated latencies. After submitting 40 jobs/minute, we increased load to 60 jobs/minute. This corresponds to a load increment of 150%. However, latency rose from 262.9 to 1796.1, a 685% increase.

To understand the impact of CREAM CE job finalization latency on job completion time, we generated the following graph, which highlights the proportion of the average time taken by each component of the EDGI infrastructure to process the job.

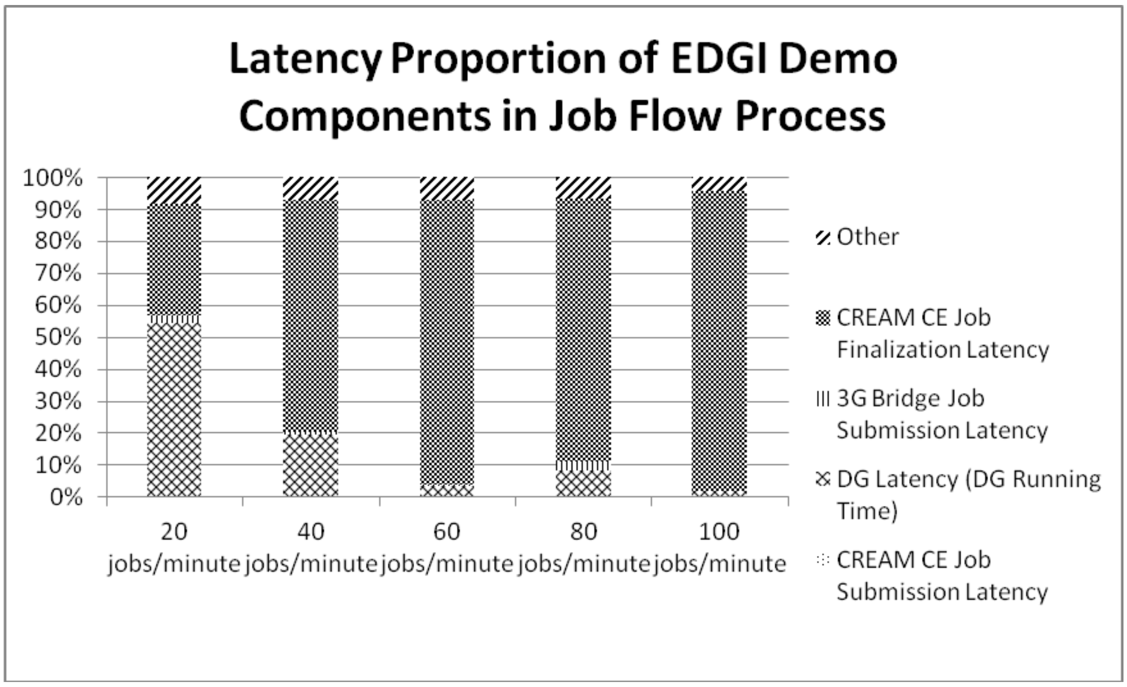


Figure 27: Latency Proportion of EDGI Demo Components in Job Flow Process

Analysis of the proportion of the EDGI Demo components in the job flow allowed us to confirm that CREAM CE finishing time directly influences overall performance of the job's 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.

4.3. ARC CE Results

The tests for measuring throughput and latency were run on an ARC Client installed at FCTUC, which connected to the production infrastructure installed at NorduGrid integrated with EDGI Demo site. This means that we ran the tests on a configured system without modifying any configuration values. To ensure correctness of results, each experiment with the same configuration was run ten times. The configuration of every experiment would allow submission of the jobs during one hour, with a fixed submission rate. The submission rate was 20-40 jobs per minute. After test completion, results were collected and processed accordingly to the procedure described in the first section on this chapter. We were unable to make tests with higher workloads, as we occasionally experienced some issues with this ARC Computing Element.

4.3.1. ARC CE Throughput

For the overall ARC CE throughput calculation, we used the same approach as in CREAM CE. To measure the overall throughput of the elements involved in the job flow process, we used the collected ARC Client's submission timestamp (T1a) and ARC Client's finish timestamp (T8a) of every successfully submitted job and applied the following formula:

$$\text{Throughput} = \frac{\text{Number of Finished Jobs}}{(\text{Timestamp of Finished Job} - \text{Timestamp of the First Submitted Job})}$$

After calculating throughput, we created the following graph showing results of ARC CE. We determined that Computing Element is able to keep up with 40 jobs/minute submission rate, without any notable throughput loss problems. The following Figure 28 shows the results of our experiment.

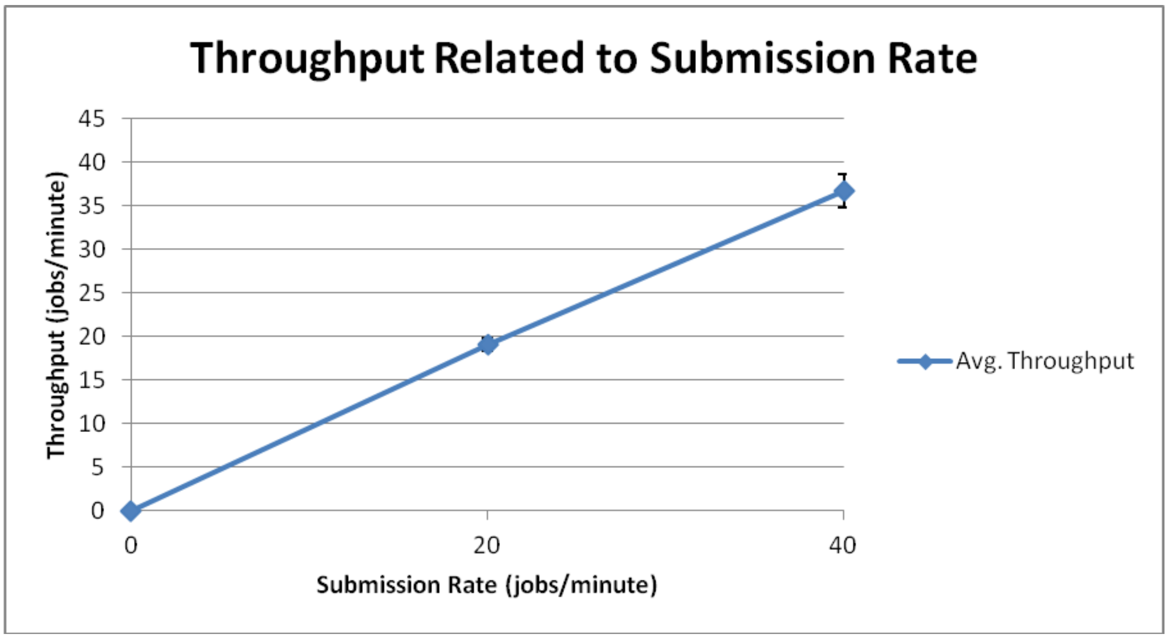


Figure 28: ARC Throughput related to Submission Rate

Standard deviation shows the stability of the throughput through different experiments run on ARC Computing Element. Calculated data could not reveal any possible bottleneck in the system with mentioned job submission load. Unfortunately experienced issues on ARC didn’t allow us to load the system to the planned 100 jobs/minute submission rate, where potential problems could appear.

4.3.2. ARC CE Latency

In the job flow process of the EDGI Demo site, with ARC as the submitter endpoint, we identified two latencies, just as we did for CREAM CE: job submission latency and job finalization latency. Similarly to throughput analysis, we used the data collected by the submitter application, and data collected from the monitoring reports of the infrastructure’s components. The latency analysis of the other EDGI Demo elements (3G Bridge and Desktop Grid) was also performed for comparison purposes and possible infrastructure problems detection (Table 7).

Calculated Latency	Formula	Units
Job Running Time (Overall Latency)	**T8a – T1a**	**Seconds**
ARC CE Preparation Latency	**T3a – T2a**	**Seconds**
3G Bridge Preparation Latency	**T5 – T4**	**Seconds**
Desktop Grid Running Time (DG Latency)	**T6 – T5**	**Seconds**
ARC CE Finishing Latency	**T7a - T6a**	**Seconds**

Table 7: ARC Latencies and corresponding Calculation Methods

To understand the impact of particular latencies in the job flow process, we have calculated and analysed particular latencies of every component involved in the job flow. The following graph (Figure 29) has identified the latencies of individual components.

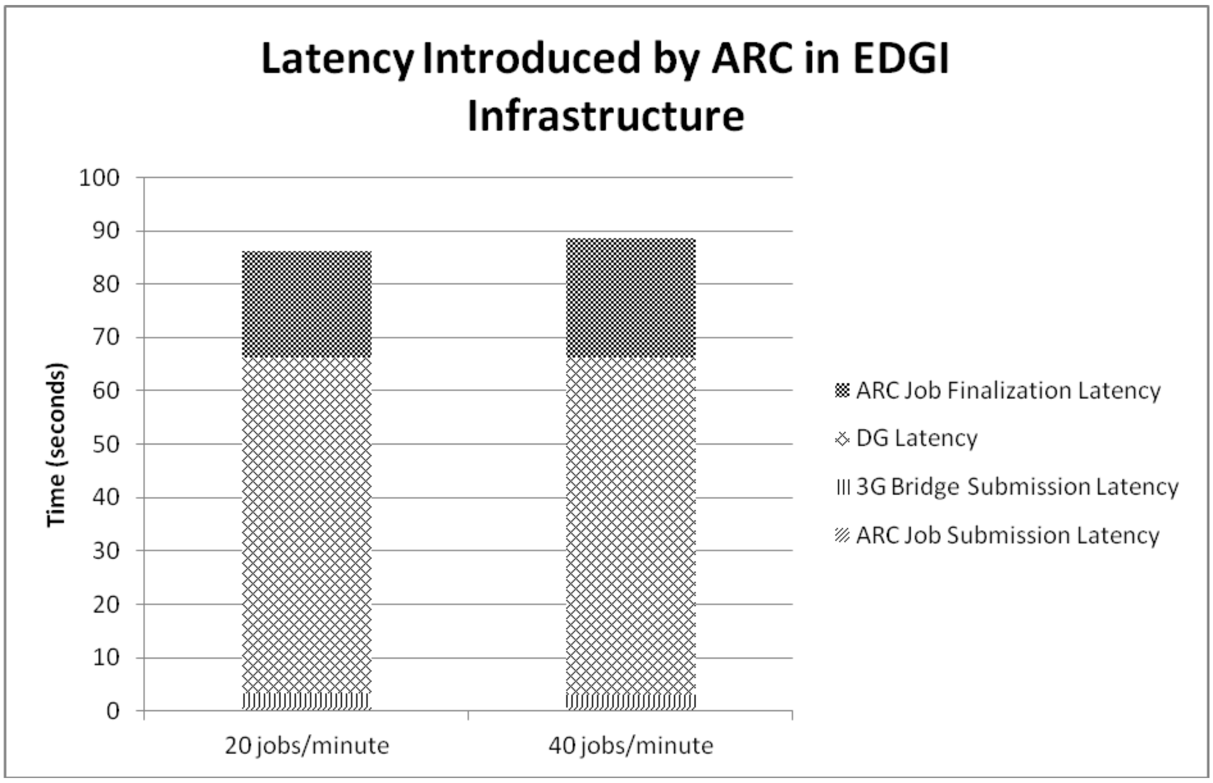


Figure 29: Latency Introduced by ARC in EDGI Infrastructure

After analysis of collected data we were able to observe that the stability of the ARC CE latency. Job submission and finalization latency measurement didn't reveal any significant problem. Job submission latency comparing with other latencies didn't introduce any overhead in job flow process and maintained constant through different experiments. We could observe some overhead introduced by job finalization process, but this can be explained by file download operations and configured time interval for job status check. Once again we observe very similar latencies introduced by Desktop Grid and 3G Bridge, proving that ARC technology integration with infrastructure does not influence performance of any of its components.

4.4. UNICORE CE Results

Benchmarking tests of UNICORE CE were run on the infrastructure installed and provided by the University of Paderborn. We have used UCC user interface to send commands to the server and service layers installed on the same machine. We performed the tests on UNICORE in several phases. The goal of the first batch of tests was helping UNICORE developers to tune configuration of the Computing Element. After configuration was tuned we advanced with benchmarking measurements of the UNICORE performance. In similarity with previously executed tests on CREAM and ARC experiments with same configuration were ran ten times. The configuration would allow the submission of jobs during one hour with different submission rate. The submission rate varied from 10 to 40 jobs per minute. Initially we planned to push UNICORE to 100 job per minute, but after significant performance loss and job submission failure growth, we stopped at 40 jobs per minute.

After spotting possible UNICORE bottleneck we have started third phase of tests to identify the source of the problem. While executing the tests we were also collecting the information about the CPU and Memory consumption by the client application, because one of the possible reasons of performance problems could be the architecture of the client.

4.4.1. UNICORE CE Throughput

UNICORE throughput measurements were performed according to the configuration described in Table 8. To collect the necessary data we recurred to analysis of the monitoring data of the EDGI Demo infrastructure. Namely the job entry to the UNICORE CE timestamp (T1b) and the job completion on UNICORE CE timestamp (T8b) were collected and processed accordingly to the following formula:

$$\text{Throughput} = \frac{\text{Number of Finished Jobs}}{(\text{Timestamp of Finished Job} - \text{Timestamp of the First Submitted Job})}$$

By applying the formula to collected data, we obtained the overall system throughput. In UNICORE measurements, we were especially focusing on UNICORE-only performance, since from our previous tests we concluded that other components following the Computing Elements (e.g., the 3G Bridge and the Desktop Grid) were not bottlenecks. To analyze the UNICORE-only performance we calculated the throughputs following in the Table 8.

Calculated Throughput	Formula	Units
Overall Throughput	$\frac{\text{Number of Completed Jobs}}{\text{T8b} - \text{MIN(T1b)}}$	Jobs/Second
UNICORE Preparation Throughput	$\frac{\text{Number of Submitted to 3G Bridge Jobs}}{\text{T3b} - \text{MIN(T2b)}}$	Jobs/Second
UNICORE Finishing Throughput	$\frac{\text{Number of Completed Jobs}}{\text{T7b} - \text{MIN(T6)}}$	Jobs/Second

Table 8: UNICORE Throughputs and corresponding Calculation Methods

Using these formulas we were able to calculate the average throughput of the system. As is in previous throughput measurements we omitted the initial 15% of results, considering this phase as a warm up period (Figure 21). In Figure 30, we illustrate the overall throughput of the system as a function of the job submission throughput.

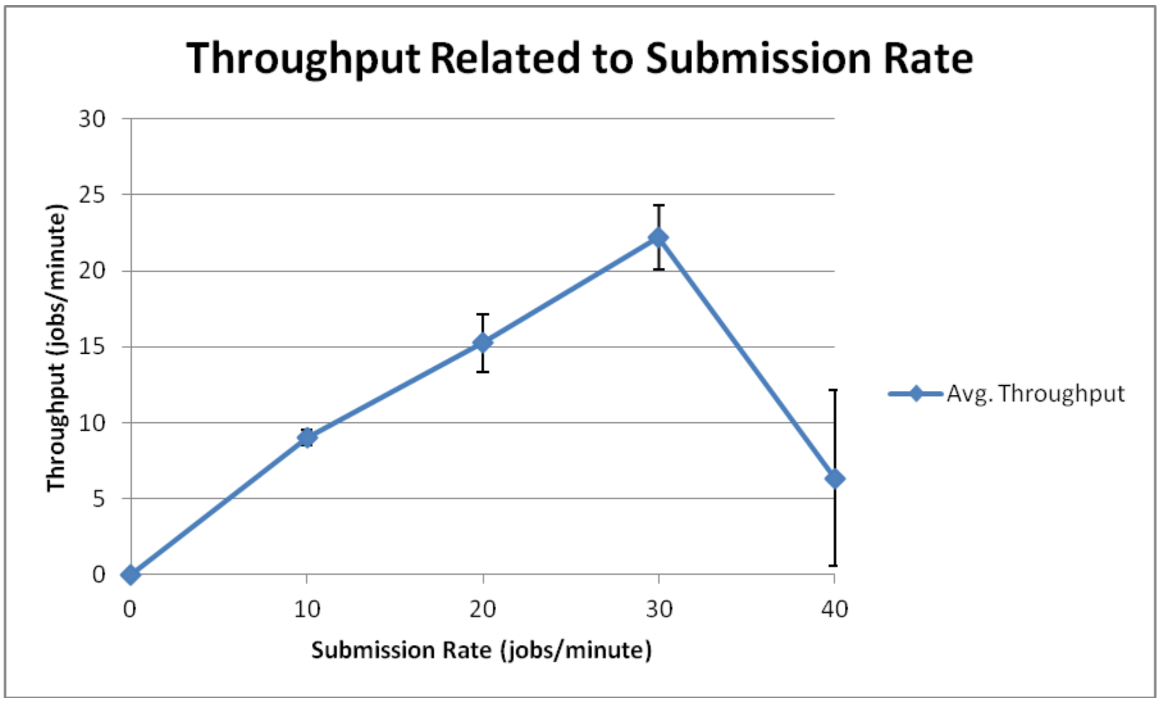


Figure 30: UNICORE Throughput related to Submission Rate

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 analyse the preparation throughput and finishing throughput of the UNICORE system. The following Figure 31 shows the results of our experiment.

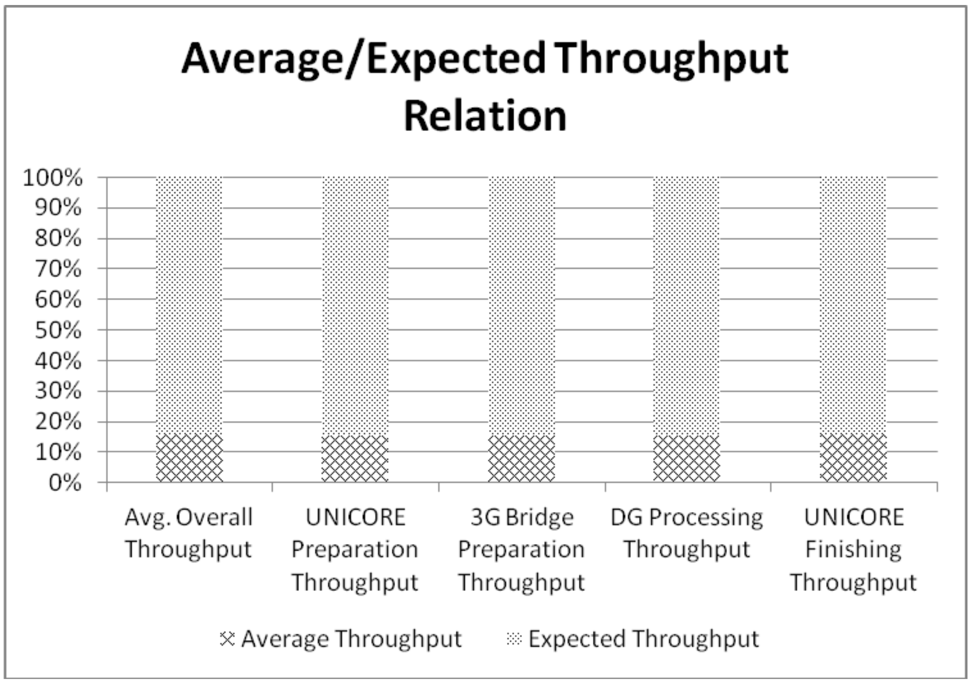


Figure 31: Average/Expected Throughput Relation

After identifying problems on the UNICORE preparation throughput, we proceeded to an in-depth analysis of the submission process. Merging the data from UI and monitoring reports, we were able to identify that the problem resided on the job submission from the client application (UCC) to the UNICORE server. The next Figure 32 shows the submission time analysis for different experiments.

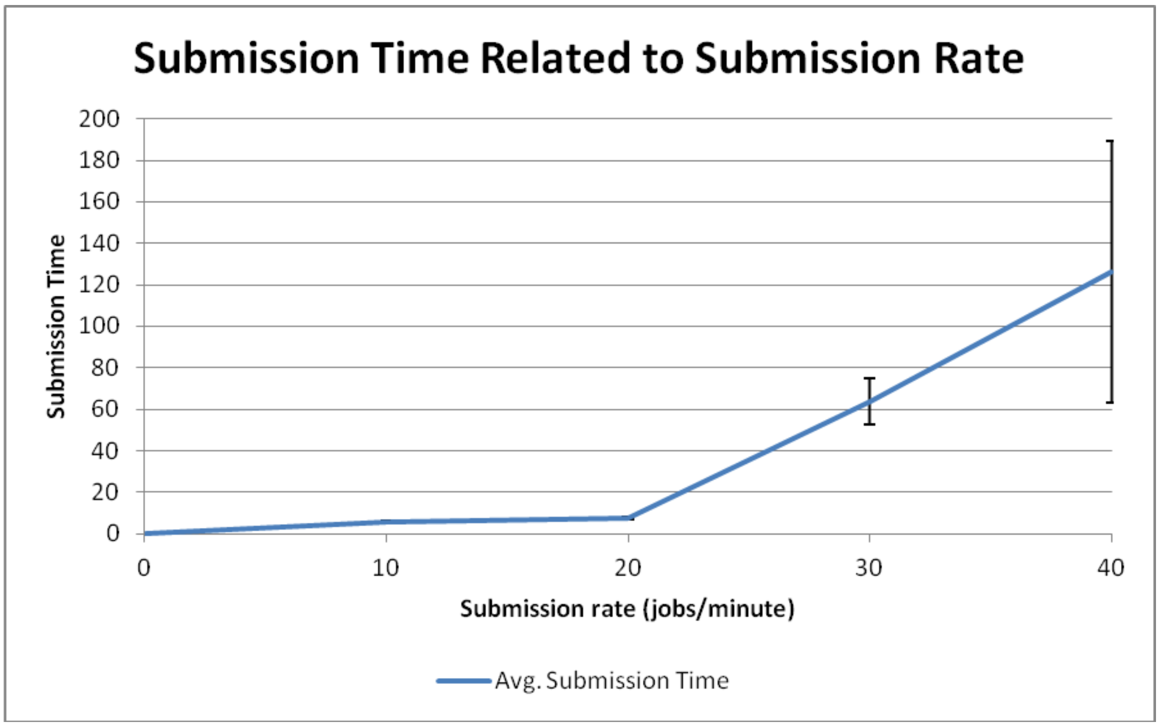


Figure 32: UNICORE Submission Time Related to Submission Rate

The submission time from UCC to UNICORE server grows quite fast relatively to job submission rate. When request takes too much time to submit, the client application reports an error and stops executing the request. There are several problems, which can contribute to this issue, creating a bottleneck on the request submission part.

First of all, after analysing of the the error reported by the client software, we could identify that the server request processing policy was either not configured or not able to handle a high number of requests. Processing of incoming requests is a very heavy process. First step is authentication of the request, by querying XUUDB. The next step after request was authorized is performing a Web Service call to the corresponding UAS. In case of job submission, the staging process is triggered on the UNICORE system. This task includes creation of a working directory, file I/O operations, communication with the Application Repository and database insertion operations. The last step is transformation of job request to a 3G Bridge compatible format and submission of the transformed job request to 3G Bridge. All these steps require communication with system services external to UNICORE, making them a possible cause of this bottleneck.

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 confirm the high demand of resources by UCC, especially CPU processing power. The results of our experiment are shown on the following Figure 33[8].

[8] There were 4 CPU cores on the machine where the tests were run, thus percentages vary from 0 to 400%

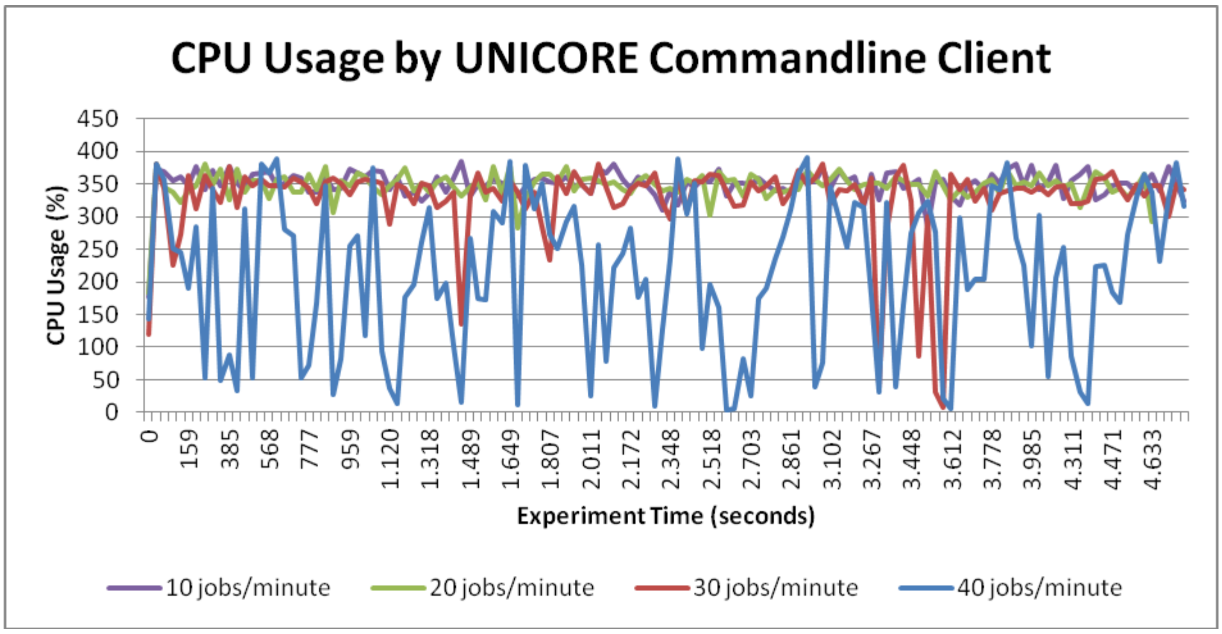


Figure 33: CPU Usage by UNICORE Commandline Client

Along with CPU Usage analysis, we also collected data about Virtual Memory Usage by the UNICORE Commandline Client. We have not found any noticeable problem with memory management by the application.

4.4.2. UNICORE CE Latency

In UNICORE latency measurements we identified two important latencies to work with: CE Preparation Latency and CE Finishing Latency. The data from submitter script and mainly the data from monitoring system were used to retrieve all necessary information to calculate this metric.

Calculated Latency	Formula	Units
Job Running Time (Overall Latency)	**T8 – T1**	**Seconds**
UNICORE CE Preparation Latency	**T3b – T2b**	**Seconds**
UNICORE CE Finishing Latency	**T7b - T6**	**Seconds**

Table 9: UNICORE Latencies and corresponding Calculation Methods

After collecting the data from the Submitter Script and Monitoring Reports we performed calculations according to the indicated in the formulas of Table 9. The following figure (Figure 34) shows the results of latency measurements made on the UNICORE system.

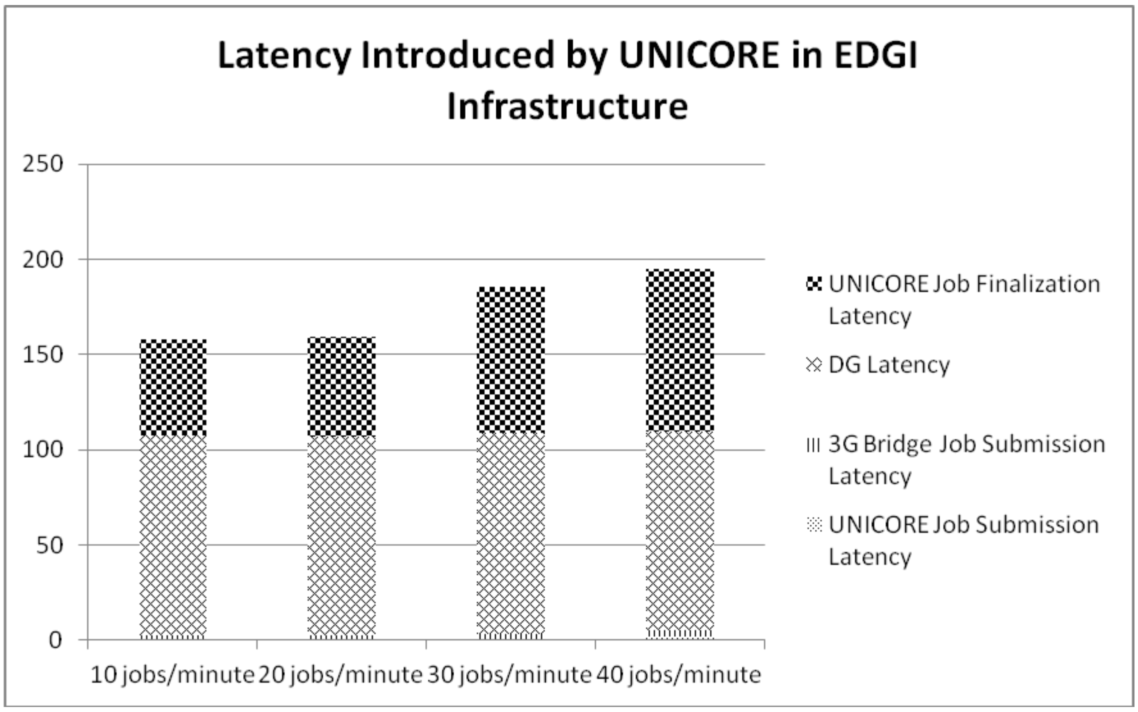


Figure 34: Latency introduced by UNICORE in EDGI Infrastructure

Once again, we could conclude that the latency of 3G Bridge remained constant independently of the technology used upstream. The Desktop Grid processing time confirmed the tendency of previously executed tests with different technologies and 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. In the staging out, the UNICORE system only reports jobs as finished when all data is downloaded and cleaned from the 3G Bridge and Desktop Grids.

The throughput limitation unfortunately did not allow us to increase the load on the system and confirm the latency behaviour with higher job submission rates. With the obtained results, we could not see any very significant delay introduced by UNICORE in the job flow process.

4.5. Computing Element Comparison

Collected results allowed us to make comparison of throughput and latency of ARC, CREAM and UNICORE Computing Elements. We have merged the data from different technology metric measurements, setting the 40 jobs/minute as upper limit. The following figure (Figure 35) shows the result of throughput for different Computing Elements.

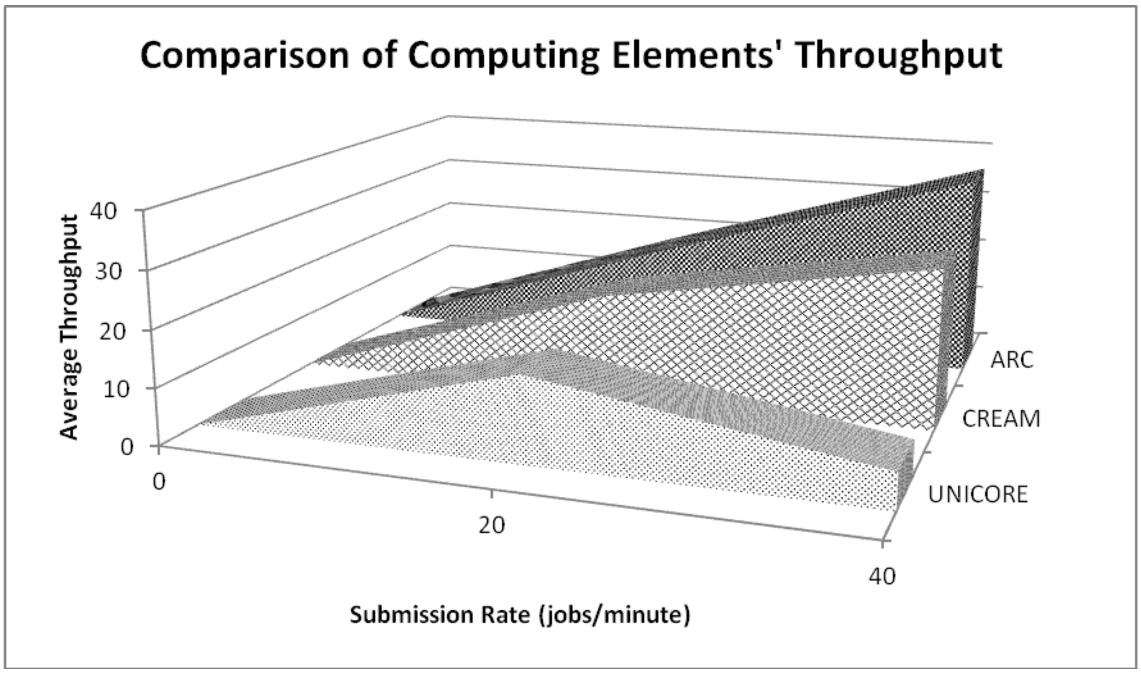


Figure 35: Comparison of Computing Elements' Throughput

We can conclude that efficiency of ARC CE architecture allows it to follow up with defined job submission rate, while other Computing Elements have throughput losses when load reaches 40 jobs/minute. The ARC CE analysis allowed us to conclude that it is also the element which introduces less latency in the system, which can be seen in the following latency comparison figure (Figure 36). Due UNICORE UI problem described in previous sections, we can conclude that architecture and implementation of the UNICORE client limits significantly its throughput, comparing to ARC and CREAM Computing Elements. The latency introduced by UNICORE Computing Element, comparing to gLite, was much higher, although it was more stable.

Delays introduced by different Computing Elements especially on job submission, does not influence much the overall job running time. The similarity of the latency on ARC, UNICORE and gLite leads us to affirm that all mentioned middleware manages to deal with submission efficiently. While the difference of latency results introduced on job finalization process provides the main distinction between performances of different Computing Elements. It is clear that gLite is unable to handle job completion process efficiently enough introducing the bottleneck in the system and keep up with another tested Computing Elements.

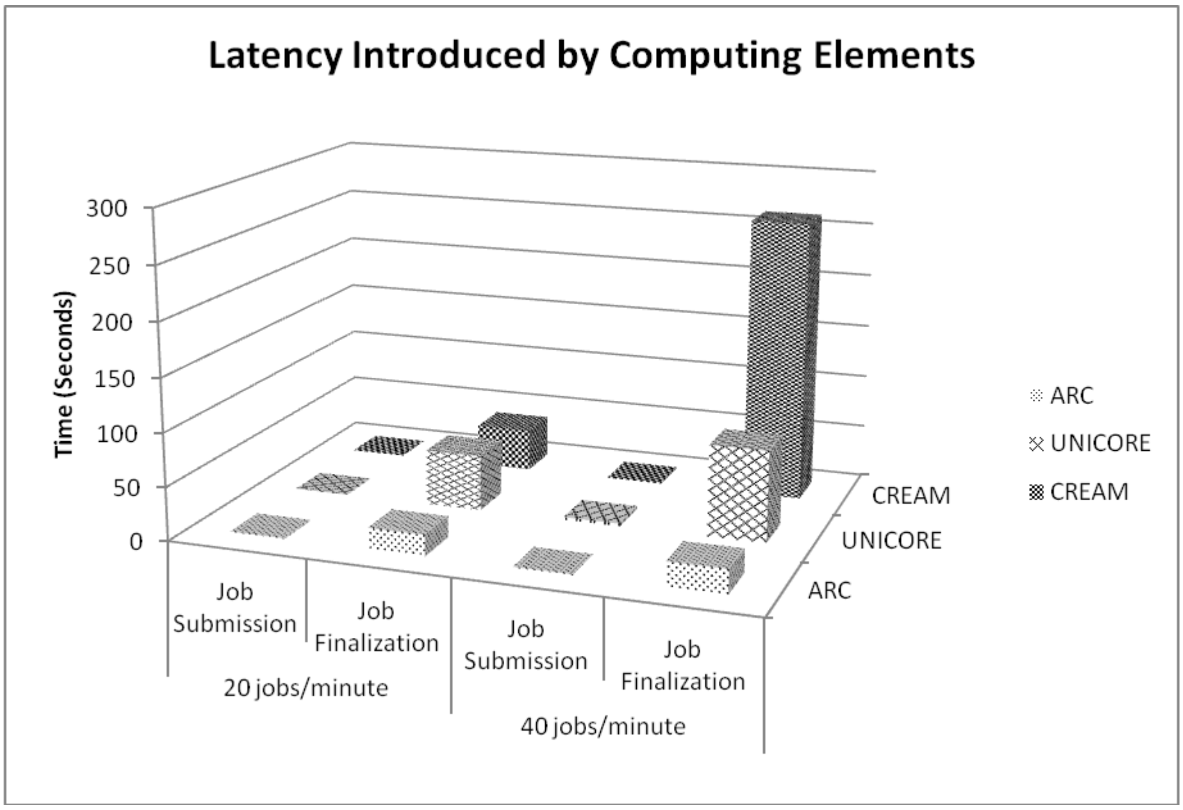


Figure 36: Latency introduced by Computing Elements comparison

4.6. 3G Bridge Results

Besides the test ran on the EDGI Demo site, we ran benchmarking tests on the 3G Bridge installed on the FCTUC infrastructure. We tested the newest version of 3G Bridge, newer then the version already tested before by FCTUC team [21]. The main goal of 3G Bridge benchmarking was to check if possible 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 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 Null_Handler plugin, which is marking jobs as finished as soon as it receives a job status query. After running the first batch of tests, we confirmed that the latency of the EDGI Demo is quite small in comparison with isolated 3G Bridge results (Figure 37)[9]. Such results can be explained by machine processing capacities difference and the fact that in isolated benchmarking scenario 3G Bridge was installed on Virtual Machine. It is also important to note that some results were omitted from this graph, since in the experiment with submission rate of 80 jobs/minute, 3G bridge performance was affected by simultaneous usage of Desktop Grid by another project.

[9] We only ran 5 tests instead of 10 per point of the graph. We believe this has negligible effect, because the standard deviation we observed was rather small.

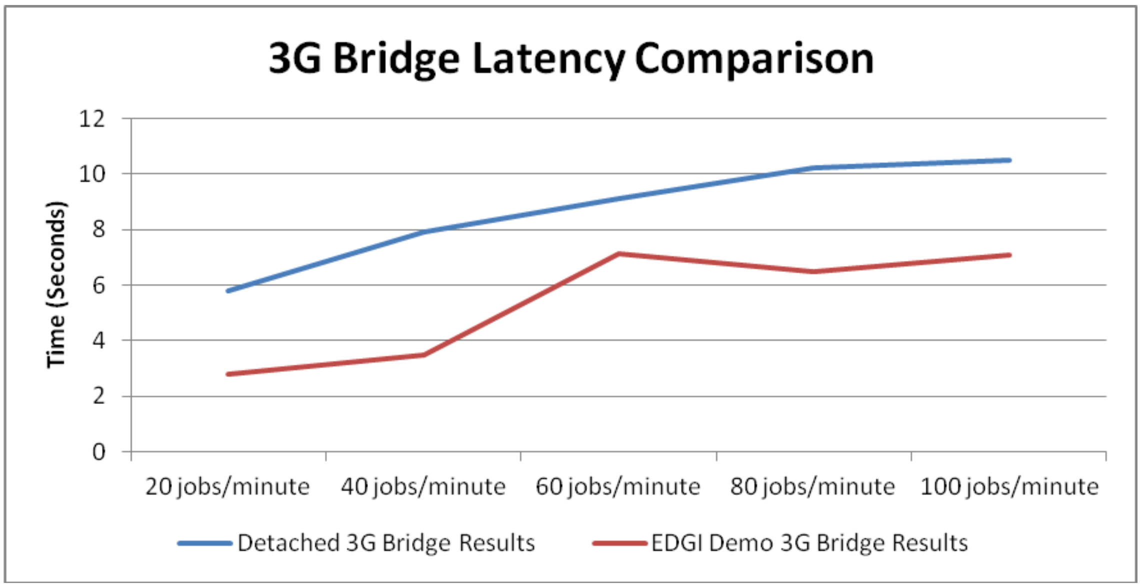


Figure 37: 3G Bridge Latency Comparison

We were also unable to detect any unexpected and drastic 3G Bridge latency growth for these ranges of job submission rate.

After performing and analysing latency, we compare the 3G Bridge throughput calculation against the submitted job rate. The results are shown in Figure 38. Once again we were unable to detect any anomaly in 3G Bridge performance. 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 3G Bridge for the currently existing infrastructures.

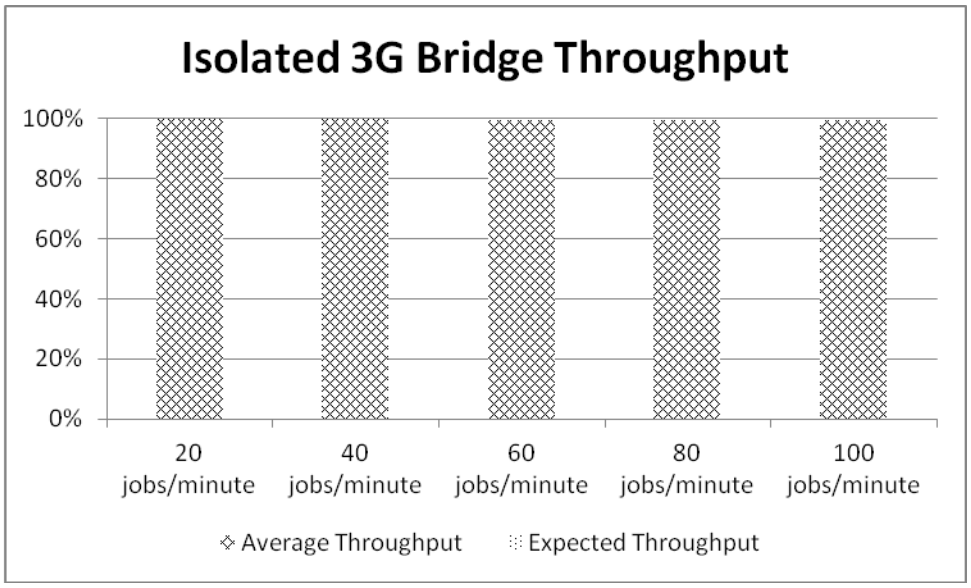


Figure 38: Isolated 3G Bridge Throughput

5. Conclusion

In this document, we evaluated the operation of the EDGI infrastructure, namely the EDGI Demo, deployed at SZTAKI, Budapest, Hungary. This effort consisted of evaluating the 3G Bridge and different kinds of Computing Elements. It is important to note that the FCTUC team has not taken part on the development of any benchmarked EDGI component, and it was mostly limited to the reading of the technical documentation, some installation and usage manuals, although the team occasionally received some help from developers, to overcome occasional technical problems. The testing strategy, test results, and test configurations were entirely decided and implemented by the FCTUC team.

The main conclusion we can take from the work done for this document is that the overall infrastructure can deal with up to 1 job/second without demonstrating any particular problems. After that point, scalability problems emerge on the CREAM Computing Element. Our tests indicated that the CREAM CE job finalization process was not efficient enough to handle higher workloads. Throughput and latency tests confirmed the unavailability of this process to keep up with the rest of the architecture. Evaluation of the source code of CREAM Computing Element suggested that the UpdateManager component is 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.

Unlike CREAM CE, UNICORE does not seem to have any job finalization problem, which was proved by the latency metric benchmarking. Although we could not execute tests with high job submission loads, due to the identified throughput loss problem, the time of job finalization remained quite stable in the performed measurements. We could also conclude that there are several problems in the UNICORE Client-Server communication, making this part the major bottleneck of the system, leading to considerable throughput problems. Deep analysis of the different phases of job flow on this Computing Element revealed its problems on effectiveness of request processing by UNICORE server and client CPU resource consumption while dealing with considerably high load.

We could also make another conclusion after performing the ARC CE benchmarking. The capacity of processing job submission of the CREAM Computing Element was higher than that of ARC. But within the currently explored ARC limits, ARC performs a much better job completion process, introducing lower latencies in the overall infrastructure. Unfortunately, once again, we could not execute all the tests we planned due to some ARC stability issues under high load and we could not evaluate the behaviour of this Computing Element when it is overloaded.

Unlike 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 behaviour in the 3G Bridge measurements, because this component responds linearly to increments in the submission of jobs. In general we concluded that performance and quality of service provided by the 3G Bridge were highly satisfactory both in isolated and EDGI Demo measurements.

We could also make another conclusion after performing ARC CE benchmarking. The capacity of processing job submission of CREAM Computing Elements was much higher than ARC CE performance even with the bottlenecks identified. To make more comparisons we would need ARC CE data with higher job submission rates, putting two Computing Elements in the same workload level. Since it was impossible to put both Computing Elements at the same workload we can conclude that the architecture of CREAM CE is more efficient and better integrated with EDGI bridging technology providing better reliable and quality service.

Analysing performed measurements and obtained results we can conclude that define in the beginning of internship goals were successfully achieved.

6. References

1. P. Domingues, P. Marques, and L. 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.
2. D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In Proceedings ot the 6th International Symposium on Cluster Computing and the Grid. IEEE ComputerSociety Press, Los Alamitos, CA, 73–80, 2006.
3. Gridcafe: the place for everybody to know about grid computing. **http://www.gridcafe.org/**. Visited on July 9, 2012.
4. Egee portal: Enabling grids for e-science. **http://www.eu-egee.org/**. Visited on July 9, 2012.
5. Enabling desktop grids for e-science. **http://edges-grid.eu/**. Visited on July 9, 2012.
6. 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.
7. F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Neri, and O. Lodygensky. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. Future Generation Computer Systems, 21(3):417-437, 2005.
8. T. Kiss, I. Kelley, and P. Kacsuk. Porting computation and data intensive applications to distributed computing infrastructures incorporating desktop grids. In The International Symposium on Grids and Clouds and the Open Grid Forum, PoS(ISGC 2011 & OGF 31) 060, 2011.
9. European desktop grid initiative. **http://edgi-project.eu/**. Visited on July 9, 2012.
10. Advanced resource connector. **http://www.nordugrid.org/arc/**. Visited on July 9, 2012.
11. Uniform interface to computing resources. **http://www.unicore.eu/**. Visited on July 9, 2012.
12. 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 July 9, 2012.
13. European Grid Infrastructure. **http://www.egi.eu/**. Visited on July 9, 2012.
14. LIPCA - Certification Authority. **http://ca.lip.pt/**. Visited on July 9, 2012.
15. M. Cardenas-Montes, A. Emmen, A. C. Marosi, F. Araujo, G. Gombas, 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.
16. C. Aiftimiei, P. Andreetto, S. Bertocco, S. Dalla Fina, A. Dorigo, E. Frizziero, A. Gianelle, M. Marzolla, M. Mazzucato, M. Sgaravatto, S. Traldi, L. Zangrando. Design

and implementation of the gLite CREAM job management service". Future Generation Computer Systems, 26(4): 654-667, 2010.
17. S. Boychenko and F. Araujo. "Benchmarking the EDGI Infrastructure". In INForum 2012, Lisbon, Portugal, September 2012. (IN PROGRESS)
18. P2P data sharing software architecture. **http://www.atticfs.org/**. Visited on July 9, 2012.
19. F. Araujo, D. Santiago, D. Ferreira, J. Farinha, L. M. Silva, P. Domingues, E. Urbah, O. Lodygensky, A. Marosi, G. Gombas, Z. Balaton, Z. Farkas and P. Kacsuk. "Monitoring the EDGeS Project Infrastructure". In 3rd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid 2009), Rome, Italy, May 2009.
20. J. Kovacs, F. Araujo, S. Boychenko, M. Keller, and A. 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
21. N. Ivaki, D. Ferreira, and F. Araujo. Benchmarking the 3g-bridge in the edges infrastructure. In Cracow Grid Workshop (CGW'09), September 2009.

Appendix A

INForum 2012 Paper

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 [17, 7, 13]. Ian Foster provides the following checklist for a grid [17]: 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 currently lives as the European Grid Infrastructure [5]. It gave birth to well-known middleware, such as gLite, currently maintained by the European Middleware Initiative (EMI) [8].

Approximately at the same time as these service grids (SGs) were gaining momentum, desktop grids (DGs) were starting to become popular, by scavenging resources from millions of users. Most of the time, many of the computers that are turned on are simply idle [16]. The abundance of spare resources eventually motivated researchers to take advantage of underutilized computational power. BOINC is a result of this effort [12].

The Enabling Desktop Grids for e-Sciente (EDGeS) [15] 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 [14].

The resulting EDGI infrastructure is a large scale distributed system, involving integration of many different technologies. Given the size and complexity of the overall infrastructure, the EDGI team deemed as necessary to benchmark the performance of its components. This process involved the overall job flow process and the 3G Bridge[15], 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 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 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 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 the bandwidth usage for frequently used job input files. Refer to the figure. After 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.

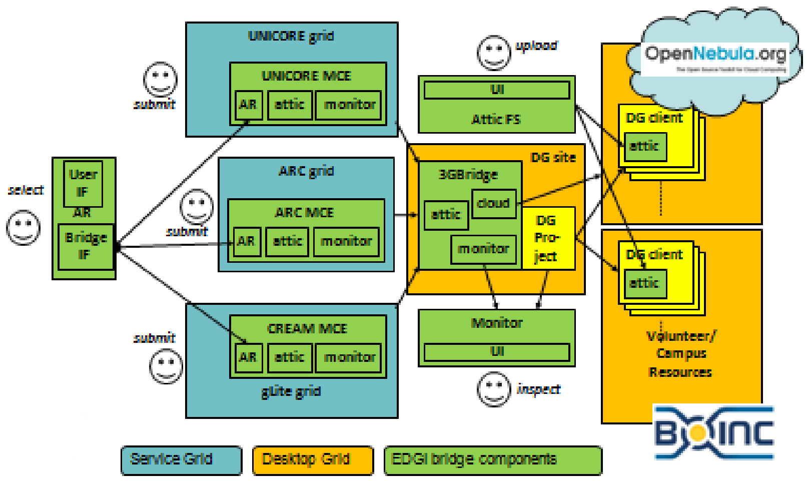


Fig. 1. EDGI Infrastructure.

3 Benchmarking Methodology

Central to the benchmarking process are the metrics we decided to take. We believe that latency and throughput are the most important criteria for grid users. The existing EDGI monitoring system [3], allowed us to collect all necessary data for every particular component involved in the flow of jobs. Moreover, in our experiments, we also tried to overload the system, to find EDGI's bottlenecks. This test serves to understand the impact of heavy job traffic on the whole EDGI infrastructure, as well as the impact on each particular middleware component. To understand if the system is overloaded, we used throughput and latency measurements. Moreover, with these results, we were able to tell which of the EDGI components is limiting the performance.

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 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

There are two applications that we use to collect all necessary data for the infrastructure benchmarking (refer to Figure 2). 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 [19]. The Data Manager Application also includes the Submitter Script data to perform the calculations required for each particular metric.

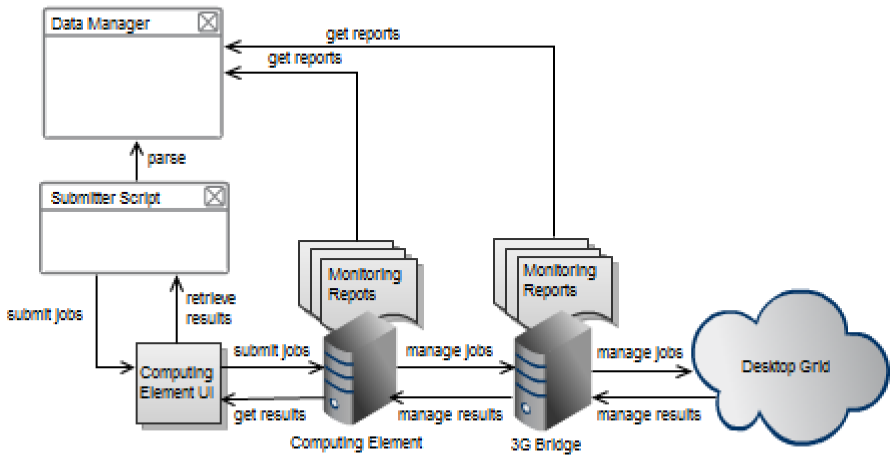


Fig. 2. Developed applications data collection scenario.

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

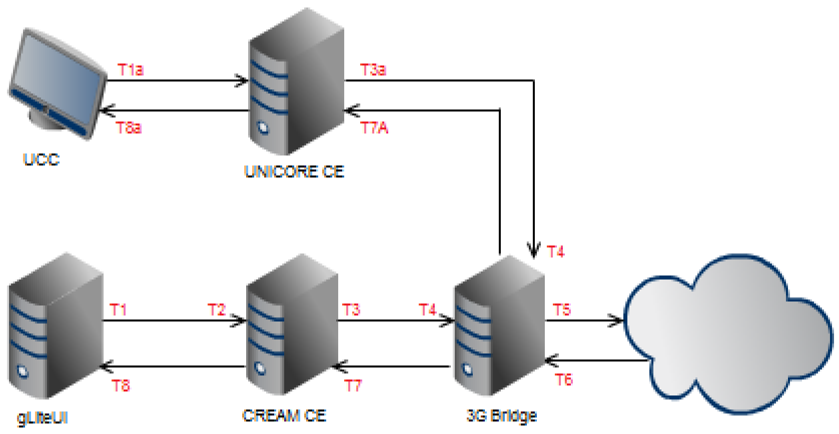


Fig. 3. Timestamps collected for throughput and latency measurment.

Timestamps T1 and T8 are collected on the submitter endpoint of the infrastructure, by the Submitter application. The remaining timestamps are collected by the Data Manager application, recurring to monitoring data generated by the infrastructures 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. The submission rate varied from 20 to 100 jobs per minute, incrementing 20 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:

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

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.

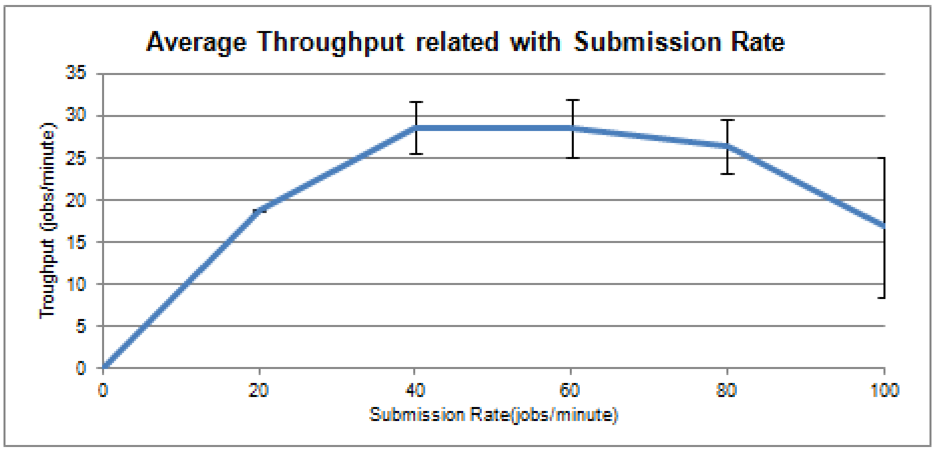


Fig. 4. CREAM average throughput related to submission rate.

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 column show 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).

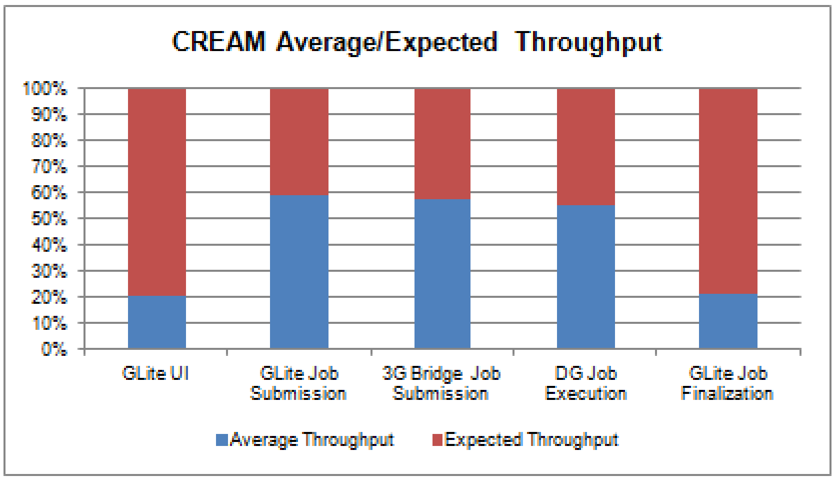


Fig. 5. CREAM Average/Expected throughput for 100 jobs/minute submission rate.

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. Analyzing results of the average throughput of the CREAM CE job submission, we observed a throughput loss whenever we raised the submission rate beyond a certain rate. 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.

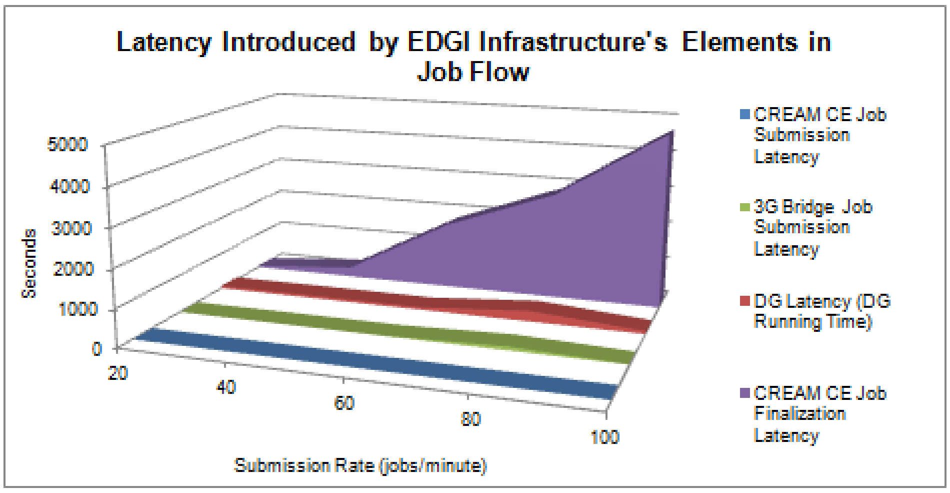


Fig. 6. Latency introduced by EDGI infrastructure components.

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 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

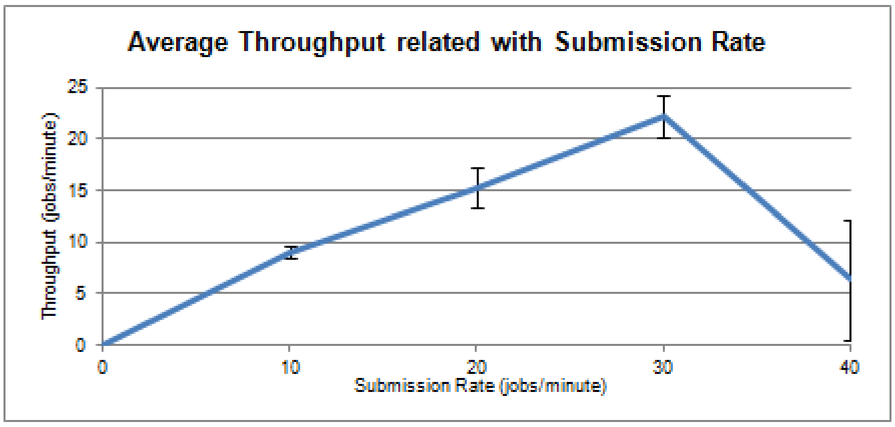


Fig. 7. UNICORE average throughput related to submission rate.

throughput of the UNICORE system. Figure 8 shows the results of our experiment, using blue for the actual throughput and red for the expected (but unreached). The problem resided on the job submission from the client application (UCC) to the UNICORE server. The submission time from UCC to UNICORE server was growing quite fast relatively to 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 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.

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

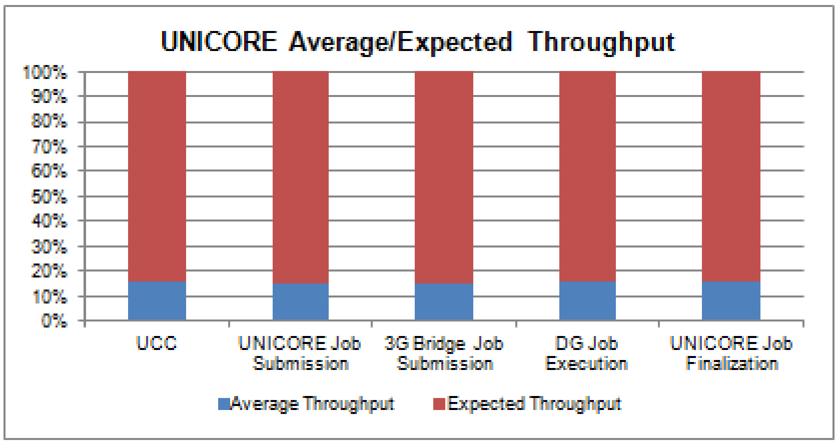


Fig. 8. UNICORE Average/Expected throughput for 40 jobs/minute submission rate.

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.

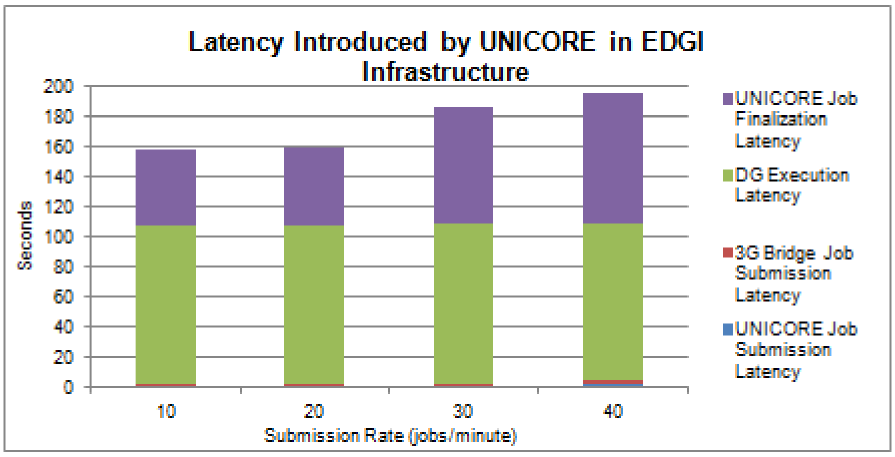


Fig. 9. UNICORE in the EDGI infrastructure.

4.3 3G Bridge Results

Besides the test ran on the EDGI Demo site, we ran benchmarking tests on the 3G Bridge installed on the FCTUC infrastructure. This version of the bridge is newer than the one we tested in [18]. The main goal of the 3G Bridge benchmarking was to check if possible bottlenecks exists 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 is marking 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 infrastructures.

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 that 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 suggested that the UpdateManager component is 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.

As we mentioned before, we experienced some stability problems with ARC that prevented us from repeating the experiments as many times as we needed. However, up to a submission rate of 40 jobs/minute, we could not observe any significant latency or throughput overheads with this technology.

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.

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. 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.
13. 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).
14. 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.
15. Miguel Cardenas-Montes, Ad Emmen, Attila Csaba Marosi, Filipe Araujo, Gabor Gombas, Gabor Terstyanszky, Gilles Fedak, Ian Kelley, Ian Taylor, Oleg Lodygensky, Péter Kacsuk, Robert Lovas, Tamas Kiss, Zoltan Balaton, and Zoltan Farkas. Edges: bridging desktop and service grids. In 2nd Iberian Grid Infrastructure Conference (IBERGRID 2008), Porto, Portugal, May 2008.
16. 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.
17. Ian Foster. What is the Grid? - a three point checklist. GRIDtoday, 1(6), July 2002.
18. Naghmeh Ivaki, Diogo Ferreira, and Filipe Araujo. Benchmarking the 3g-bridge in the edges infrastructure. In Cracow Grid Workshop (CGW'09), September 2009.
19. 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.