**Grid Data Access: Proxy Caches and User Views**

Cristian Traian Cirstea

September 2011

Grid Data Access: Proxy Caches and User Views

Eindhoven University of Technology

Stan Ackermans Institute / Software Technology

|  |  |  |
| --- | --- | --- |
| **Partners** | nikhef_logo.gif |  |
| Nikhef | Eindhoven University of Technology |
|  | | |
| **Steering Group** | J. Templon  M. Petkovic | |
| J.J. Keijser | |
| O. Koeroo | |
| D. Groep | |
|  | | |
| **Date** | September 2011 | |

|  |  |
| --- | --- |
| Contact  Address | Eindhoven University of Technology  Department of Mathematics and Computer Science  HG 6.57, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands  +31402474334 |
|  | |
| Published by | Eindhoven University of Technology  Stan Ackermans Institute |
|  | |
| Printed by | Eindhoven University of Technology  *UniversiteitsDrukkerij* |
|  | |
| ISBN | 978-90-444-1067-9 |
| Abstract | A new grid data access model has been developed at Nikhef, based on a caching mechanism and WebDAV access. This grid proxy cache can be added independently "in front of" existing Logical File Catalogue (LFC) and Storage Resource Manager (SRM) implementations, without having to make any modifications to the LFC or SRM servers. It provides a caching mechanism to speed up access to frequently requested files, as well as a transparent platform-independent access protocol. A secure WebDAV interface is offered, so that no grid-specific client software is needed to access the grid proxy cache. The WebDAV interface is accessible using username and passwords from Windows XP/Vista/7 and Mac OS X clients, and using grid certificates or proxies from Linux worker nodes. The grid proxy cache also includes load balancing and redundancy mechanisms. |
| Keywords | WebDAV, Grid, SRM, Storage Element, WLCG, EGI, Cache, Proxy Cache, Proxy Certificate, GSI, Distributed System, Content Delivery Network, CDN, Data Management, Data Access, Authentication, Authorization |
|  | |
| Preferred  reference | Cristian Traian Cirstea, Grid Data Access: Proxy Caches and User Views. Eindhoven University of Technology, SAI Technical Report, September, 2011.  A catalogue record is available from the Eindhoven University of Technology Library (ISBN: 978-90-444-1067-9) |

|  |  |
| --- | --- |
| Partnership | This project was supported by Eindhoven University of Technology and Nikhef. |
|  | |
| Disclaimer  Endorsement | Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology or Nikhef. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology or Nikhef, and shall not be used for advertising or product endorsement purposes. |
|  | |
| Disclaimer  Liability | While every effort will be made to ensure that the information contained within this report is accurate and up to date, makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information. |
|  | |
| Trademarks | Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners. |
|  | |
| Copyright | Copyright © 2011. Eindhoven University of Technology. All rights reserved. |
|  | No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the Eindhoven University of Technology and Nikhef. |

Foreword

This report concerns a project in Grid Computing.  The name "Grid Computing" has been applied to many different types of computing infrastructures; the Grid in which this project is embedded spans most of the continents, distributed across more than 400 sites and containing tens of petabytes of data.  This data can be relatively easily stored, but access to it is rather cumbersome, and this is an impediment to using the grid by most users.

Last year a major grid data-management workshop was held in Amsterdam, where Content-Delivery Networks were identified as a possible solution to our data access problems.  There were many unknowns, for example do CDNs work for our (grid) data access patterns, and how would industry-standard CDN platforms work with the authentication and authorization tools used in the Grid world?

Cristian Cirstea accepted the challenge to explore these unknowns and provide answers.  He did an outstanding job in discovering the true requirements for such a system -- not so easy to extract from our users -- constructed a prototype, tested it at scale, and answered many of the questions we had.  Most importantly he showed clearly what the potential benefit of such a system is, and he showed exemplary restraint in constructing the system, something I wish more "grid programmers" could do.  Where existing tools fit the bill, he used them, and concentrated his valuable time on solving the \*unsolved\* problems rather than re-solving the solved ones.  Not just Nikhef, but our entire grid community, will benefit from his work.

Dr. J. Templon

September 12, 2011

Preface

This report describes the work carried out during the final project for the Stan Ackermans Institute “Software Technology” program at the department of Mathematics and Computer Science, Eindhoven University of Technology in collaboration with Nikhef[[1]](#footnote-1).

The project consists in the design and development of a solution that provides to the clients an easy and efficient way to access data stored by the grid. The provided solution allows inexperienced users to access their data without the need of any special tools.

The report is addressed to a technical audience that has general knowledge about software design and Grid computing. Readers that are interested in the main goals of the project should read the Executive Summary of the document. Readers that are interested in the design and implementation as well as the initial ideas should reefer to Chapters 3-6. For results and conclusions, readers must address the Chapters 7 and 9.

T.C. Cirstea

October 1, 2011

Acknowledgements

I would like to thank all the people who supported, encouraged and helped me through the last nine months. I could not have come to the results described in this report without their help.

First of all I would like to thank my company supervisors, Jeff Templon, Jan Just Keijser, Oscar Koeroo and David Groep for providing me with this opportunity and for guiding me during the project. I enjoyed our meetings and collaboration that have always been in an encouraging atmosphere, and the tremendous number of emails that we have exchanged during the project. I especially appreciated the trust and freedom they gave me to manage and steer my project. I would also like to thank Ronald Starink for his support, useful advices, and patience showed during the validation phase of the project.

I would like to thank Milan Petkovic, my university supervisor, for his structural comments during the Project Steering Group Meetings and reviews of this report.

I want to thank my fellow OOTI colleagues for the great moments that we have shared in the past two years. Special thanks go out to Ad Aerts, Maggy de Wert, and all our trainers for their continuous support and wise lessons throughout the whole OOTI period.

Last but certainly not least, I would like to express my gratitude to my family and friends for their interest, and support. Thank you.

T.C. Cirstea

October 1, 2011

Executive Summary

This report presents the results of “Grid Data Access: Proxy Caches and User Views”, carried out at National Institute for Subatomic Physics (Nikhef).

The institute focuses on research in the area of (astro) particle physics. Moreover, it provides data storage and processing facilities as part of Worldwide LHC[[2]](#footnote-2) Computing Grid. Scientists use these facilities to analyze and process data that is stored on that grid.

Accessing data that resides on the grid can be achieved in multiple ways. Most of the times it can be done by making use of various tools that that are unique to grid computing and hence unfamiliar to inexperienced users. The utilization of these tools proves to be difficult and in some cases rather inefficient. In order to ease the access to data, several groups of users were surveyed to determine a method that would be considered easy to use and in the same time to add value to their work.

The project described in this report provides a secure WebDAV[[3]](#footnote-3) interface that is accessible from any operating system by making use of username and passwords, and using grid certificates or proxies from Linux worker nodes. To reduce the resource consumption, the project implements a transparent distributed cache system that stands between the users and the storage elements, thus whenever a file is accessed more than once, it will be delivered directly from the cache.

The results show that by providing the WebDAV interface to the user, no grid-specific tools need to be installed upfront, except a Firefox plugin that is used to delegate the credentials. Moreover, the worker nodes can connect and access data via the WebDAV interface without major changes in their configuration. The execution time of the jobs can be highly improved by using the current project due to its distributed cache mechanism.

It is recommended to make use of the WebDAV interface anytime when the same data is accessed more than once. Any site that wants to provide the interface should deploy the current project locally.

Table of Contents

1 Introduction 1

1.1 Context 1

1.2 Purpose and Scope 1

1.3 Stakeholder Analysis 1

1.4 Deliverables 2

1.5 Overview of this report 2

2 Domain Analysis 5

2.1 Introduction 5

2.2 The gLite middleware 6

2.3 Public Key Infrastructure (PKI) 9

2.4 Problem Analysis 12

3 Feasibility Study 15

3.1 Introduction 15

3.2 Development environment 15

3.3 Interaction with the Grid 15

3.4 Communication interface for the user 17

3.5 Authentication methods 23

3.6 Caching methods 27

3.7 Results & Conclusions 37

3.8 Implementation issues 38

4 System Requirements 41

4.1 Introduction 41

4.2 Functional Requirements 41

4.3 Nonfunctional requirements 43

5 System Architecture 45

5.1 Introduction 45

5.3 System description 46

5.4 System overview 48

5.5 Data flow and scenarios 48

5.6 Decisions 51

6 System Design 53

6.1 Introduction 53

6.2 Logical View 53

6.3 Process View 61

6.4 Deployment View 61

7 Validation & Verification 63

7.1 Validation 63

7.2 Verification 73

8 Results 75

8.1 Deployment and test setup 75

8.2 Experimental results 77

9 Conclusions 95

9.1 Conclusions 95

9.2 Recommendation 97

9.3 Future work 97

10 Project Management 99

10.1 Process 99

10.2 Breakdown structure 99

10.3 Milestone Trend Analysis 100

10.4 Risk management 102

10.5 Project retrospective 102

Glossary 105

Bibliography 107

Appendix A*: Job example* 109

Appendix B*: Usage of htproxyput and wdfs-redirect* 110

Appendix C*: PROPFIND response (for Windows 7)* 111

Appendix D*: Windows XP & 7 WebDAV Client required changes* 111

Appendix E*: mod\_authn\_myproxy required changes and compilation* 112

Appendix F*: List of problems identified during the project* 112

Appendix G*: wget and curl – GSI proxy certificates* 114

Appendix H*: Development environment* 114

Appendix I*: Apache Benchmark (ab) example of usage* 114

Appendix J*: Tools and frameworks used for implementation* 114

About the Author 116

List of Figures

Figure 1 – Stakeholders of the current project overview 2

Figure 2 – Schematic diagram of the WLCG Grid 5

Figure 3 – Simplified gLite Architecture (example of a site) 6

Figure 4 – Computing Element (CE) overview 7

Figure 5 – Storage Element (SE) overview 7

Figure 6 – Workload Management System (WMS) overview 8

Figure 7 – Example of interdependency between files and replicas 8

Figure 8 – Overview of the Public Key Infrastructure and example of authentication to 3rd parties by making use of X.509 certificates 9

Figure 9 – How the public and private keys work to securely exchange a message 10

Figure 10 – The X.509 [GSI] proxy certificate creation process (*Delegation*) 11

Figure 11 – Example of how currently a file can be copied from the grid to a specific computer by making use of the UI interface 12

Figure 12 – Overview of the Approach 1 (FUSE + standard WebDAV) 20

Figure 13 – PKI authentication with Apache2 and GridSite sequence diagram 24

Figure 14 – Overview of the username and password authentication method provided by MyProxy Server and the Apache2 module of MyProxy 25

Figure 15 – Overview of the process that adds VOMS attributes to a proxy certificate 26

Figure 16 – Integration between a web server and a reverse-proxy-cache 28

Figure 17 – Overview of the WebDAV prototype that publishes the LFC and SE files 28

Figure 18 – Reverse proxy cache between the Client and the WebDAV Server architecture overview 28

Figure 19 – Authorization mechanism provided by reverse-proxy-caches 29

Figure 20 – Problem description with the reverse-proxy-cache and loss of the user's identity on the WebDAV Server 30

Figure 21 – Caching files when the WebDAV server provides the reverse-proxy-cache functionality as well (red lines show where the file is duplicated) 30

Figure 22 – Reverse proxy cache between WebDAV Server and Storage Element architecture overview 32

Figure 23 – Sequence diagram of actions that are performed when the reverse proxy cache is between WebDAV server and Storage Element 32

Figure 24 – Possible usage of Squid and Varnish between the WebDAV server and Storage Element 33

Figure 25 – Architecture of the prototype that uses reverse-proxy-caches and REDIRECT capability 34

Figure 26 – Generic sequence diagram of the Approach 3 34

Figure 27 – Approach 4: caching without reverse proxy caches 36

Figure 28 – Generic sequence diagram of Approach 4 36

Figure 29 – Overview of the Functional Requirement 4 42

Figure 30 – Architecturally Design Layers Overview 45

Figure 31 – General components interaction overview 46

Figure 32 – Architectural overview of the System Under Development (SUD) 48

Figure 33 – Interaction between the User, Delegation Service and other components of the SUD (such as Cache Node, or WebDAV Server) 48

Figure 34 – Interaction between user, MyProxy Server and other components of the SUD (such as Cache Node, or WebDAV Server) 49

Figure 35 – Interaction between a WebDAV client (supports REDIRECT) and SUD 49

Figure 36 – Example of interaction between a web browser and the SUD 50

Figure 37 – Interaction between client and Cache Nodes: the file is cached but not on the Cache Node where the request was sent initially; the client is redirected to the Cache Node that holds the file 50

Figure 38 – Interaction between client and Cache Nodes: the file is not cached by any of the Cache Nodes; it is copied from the Storage Element 51

Figure 39 – Architecturally Significant Design Packages 53

Figure 40 – Overview of the classes and interfaces that compose the *core* package 54

Figure 41 – Description of the main interfaces provided by the *core* package 54

Figure 42 – Important classes provided by the *core* package 55

Figure 43 – Overview of the classes that compose the *wrappers* package 55

Figure 44 – Overview of the classes that compose the *specific* package 56

Figure 45 – Overview of the classes that compose the *fcgi* package 57

Figure 46 – Overview of the classes that compose the *cdn* package 57

Figure 47 – *WebDAVEngine* detailed class diagram 58

Figure 48 – Operations that take place when a *FrontEndFCGIApplication* is started 59

Figure 49 – Overview of the operations that take place when WebDAVEngine receives a request 59

Figure 50 – Interaction between the modules that compose the WebDAV Server 60

Figure 51 – Interaction between the modules that compose the Cache Node 60

Figure 52 – Division into multiple processes of the WebDAV Server 61

Figure 53 – Division into multiple processes of the Cache Node 61

Figure 54 – Deployment diagram of the SUD 62

Figure 55 – The case when a file can be accessed without permission 66

Figure 56 – Test and deployment environments overview 76

Figure 57 – Mounted WebDAV Repository on a Mac OS X system 78

Figure 58 – Mounted WebDAV Repository explored with the Terminal 78

Figure 59 – Browsing the WebDAV Repository with Firefox 79

Figure 60 – Downloading a file from SUD, by using *curl* 80

Figure 61 – Average execution time of jobs that need the same file and use lcg-cp 81

Figure 62 – Comparison between average duration of jobs, those require the same file, and use the lcg-cp and the SUD that has 1 Cache Node (in total 24GB RAM) and the file is NOT cached 82

Figure 63 – Comparison between average duration of jobs, those require the same file, and use the lcg-cp and the SUD that has 1 Cache Node (in total 24GB RAM) and the file is cached 82

Figure 64 – Comparison between average duration of jobs, those require the same file, and use the lcg-cp and the SUD that has 2 Cache Nodes (in total 48GB RAM) 83

Figure 65 – Job average duration improvement when SUD has 2 Cache Nodes 84

Figure 66 – SUD efficiency when the number of Cache Nodes increases from 1 to 2 84

Figure 67 – Comparison between average duration of jobs, those require the same file, and use the lcg-cp and the SUD that has 3 Cache Nodes (in total 72GB RAM) 85

Figure 68 – Job average duration improvement when SUD has 3 Cache Nodes 85

Figure 69 – SUD efficiency when the number of Cache Nodes increases from 1 to 3 86

Figure 70 – SUD efficiency when the number of Cache Nodes increases from 2 to 3 86

Figure 71 – Average execution time of jobs that need different files and use lcg-cp 87

Figure 72 – Comparison between average duration of jobs, that require different files, and use the lcg-cp and the SUD that has 1 Cache Nodes (in total 24 GB RAM) 88

Figure 73 – Comparison between average duration of jobs, that require different files, and use the lcg-cp and the SUD that has 2 Cache Nodes (in total 48GB RAM) 89

Figure 74 – Job average duration improvement when SUD has 2 Cache Nodes 89

Figure 75 – Comparison between average duration of jobs, that require different files, and use the lcg-cp and the SUD that has 3 Cache Nodes (in total 72GB RAM) 90

Figure 76 – Job average duration improvement when SUD has 3 Cache Nodes 90

Figure 77 – Comparison between average duration of jobs, that require different files, and use the lcg-cp and the SUD with 1 Cache Node that has only 12 GB of RAM 91

Figure 78 – Comparison between the average duration of a job that uses the lcg-cp and the SUD with 1 and 2 Cache Nodes when the data transfer is encrypted (HTTPS) 91

Figure 79 – Comparison between average duration of jobs that require the same 100MB file and use the lcg-cp and the SUD with 1 Cache Node 92

Figure 80 – Comparison between average duration of jobs that require different 100MB files and use the lcg-cp and the SUD with 1 Cache Node 92

Figure 81 – Planned timeline of the project 100

Figure 82 – Milestone Trend Analysis chart 101

Figure 83 – Real timeline of the project 101

List of Tables

Table 1 – List of deliverables 2

Table 2 – Important methods provided by LFC Python API 16

Table 3 – Important methods provided by GFAL and LCG\_UTIL Python APIs 16

Table 4 – WebDAV specific methods added on top of HTTP [12] 17

Table 5 – WebDAV support provided by the Apache2, NginX and Lighttpd 18

Table 6 – Overview of a virtual file system implemented with FUSE 19

Table 7 – Logical description of FUSE application that publishes the files from LFC 19

Table 8 – Approach 1: How multiple users are managed by the FUSE application 20

Table 9 – Mapping between WebDAV methods and required functionality 21

Table 10 – Overview of the two approaches in order to provide WebDAV support 22

Table 11 – Overview of the PKI authentication solution (Apache2 + GridSite) 24

Table 12 – Logical steps to provide username and password authentication 25

Table 13 – Overview of the two required authentication methods 27

Table 14 – Overview of the problems encountered while using reverse-proxy-caches 31

Table 15 – Overview of the reverse-proxy-caches and the problems encountered 31

Table 16 – Overview of problems, disadvantages and advantages encountered with Approach 2 33

Table 17 – Overview of problems and advantages encountered with Approach 3 35

Table 18 – Overview of problems and advantages encountered with Approach 4 36

Table 19 – Overview of the caching approaches 37

Table 20 – Overview of the possible solutions for each of the requirements 37

Table 21 – Main problems identified while using grid frameworks 38

Table 22 – Main problems identified while integrating standard and grid technologies 39

Table 23 – Main problems identified while using standard technologies 39

Table 24 – Description of the main components of the SUD 47

Table 25 – List of tests to validate the WebDAV implementation 63

Table 26 – List of tests to validate the authentication methods 64

Table 27 – List of tests to authenticate the user 64

Table 28 – List of tests to check user permission to access a file 65

Table 29 – List of tests to validate the authentication methods 67

Table 30 – List of tests to validate the WebDAV implementation 67

Table 31 – List of tests to validate the HTTP access to data 67

Table 32 – List of tests to validate the curl and wget 67

Table 33 – List of request types that a user can make 68

Table 34 – List of tests performed to measure the overhead introduced by the SUD 68

Table 35 – Parameters changed during the performance tests 70

Table 36 – List of tests performed to calculate the Performance and the Scalability of the SUD (✔ - Test is performed; ✖ - Test is NOT performed) 70

Table 37 – Description of the job that uses *lcg-cp* 71

Table 38 – Description of the job that uses *globus-url-copy* 71

Table 39 – Description of the job that uses the SUD 71

Table 40 – List of tests to check the robustness of the SUD when unexpected scenarios happen; or when unexpected errors occur 72

Table 41 – Configuration of the machines where the SUD is deployed 75

Table 42 – Configuration of the Worker Nodes 75

Table 43 – Configuration of the Disk Servers that compose the Storage Element 76

Table 44 – Authentication methods supported by the WebDAV Clients (✔ - authentication method is accepted; ✖ - authentication method is NOT accepted) 77

Table 45 – Operations supported by the WebDAV Clients (✔ - operation is supported; ✖ - operation is NOT accepted) 77

Table 46 – Compatibility matrix between web browsers and SUD 79

Table 47 – Compatibility matrix between curl and wget, and SUD 79

Table 48 – Time required by the WebDAV server to resolve requests when the memory cache is disabled 80

Table 49 – Time required by the WebDAV server to resolve requests when the memory cache is enabled 80

Table 50 – High level requirements achievement 96

Table 51 – List of all iterations, deadlines and deliverables 99

Table 52 – Most important risks identified during the project 102

Table 53 – Simple example of job that uses the SUD 109

Table 54 – Example of HTTP-response generated by the SUD when the server receives a PROPFIND request 111

Table 55 – Problems identified while using GridSite tool 112

Table 56 – Problems identified while using MyProxy 112

Table 57 – Problems identified while using the gLite framework 113

Table 58 – Problems identified while using WebDAV 113

Table 59 – Problems identified while integrating the standard and grid technologies 113

Table 60 – Problems identified while integrating the SUD with the Worker Node 113

Table 61 – List of main tools and frameworks used to implement the project 114

# Introduction

*This chapter presents the context and the goal of the current project, the analysis of the most important stakeholders, as well as it summarizes the list of deliverables. The chapter concludes with an overview of the structure of this report.*

## Context

The National Institute for Subatomic Physics (Nikhef) is an institute that carries out research in the area of (astro) particle physics. Scientists and engineers work together on research regarding the smallest building blocks of matterand the forces that act between them [10]. Moreover, the institute provides data storage and computing facilities as part of the Worldwide LHC Computing Grid, which is mostly used by the CERN[[4]](#footnote-4) experiment.

The main customer of Nikhef is, as expected, the High Energy Physics Group, which uses most of the hardware and software resources provided. Because the Dutch government partially funds Nikhef in order to provide a national grid infrastructure, the other customers become very important. The problems that they encounter while using the grid need to be identified and solutions need to be implemented.

The main focus of this research is the users and the way they are interacting with the grid. They have different backgrounds, different views of how the grid (should) works and they are using it according to what they know that works. The same happens when it comes to accessing the data within the grid. There are many possible ways that can be followed in order to access a file. The users will choose from all these possibilities the one that they are used to or the one that fits their needs. Most of the times whatever their choice is, the data access turns out to be difficult to achieve.

This report contains the results and conclusion of the feasibility study, the main problems identified when integrating standard technologies with grid-technologies, and the design and implementation of a project that provides a standard interface for the users to access data. The solution has been deployed on a real grid environment during the tests.

## Purpose and Scope

This technical report presents the project “Grid Data Access: Proxy Caches and User Views,” performed at Nikhef in Amsterdam. The main focus of the report is on the results of the problem analysis and the design process, from both technical and managerial point of view.

The main goal of this project is to offer a user-friendly and efficient alternative way to access data stored on the grid. To attract other users, the interfaces (protocols) that allow interaction with the grid should be as well defined as possible and standard technologies/protocols should be used.

## Stakeholder Analysis

Nikhef is the main stakeholder of the project. The Physics Data Processing Group (PDP) of Nikhef carries out the Grid computing Research, Development and Deployment activities, and its primary focus is to provide the software and hardware infrastructure for High-Energy Physics (HEP) research, and to design security-related Grid software and policy infrastructures. This project is conducted by the PDP group.

The project aims to make easier for the users to access data stored by the Grid. The users needs and perception about a user-friendlier Grid are surveyed in order to define the proper requirements. Thus their participation and input in making decisions and conceiving this project is high, and this makes them one of the main stakeholders.

Along with the user-friendliness, the current project attempts to optimize and reduce the consumption of resources within the Grid (e.g. the network bandwidth), thus the Grid’s technical board becomes an important stakeholder as well.

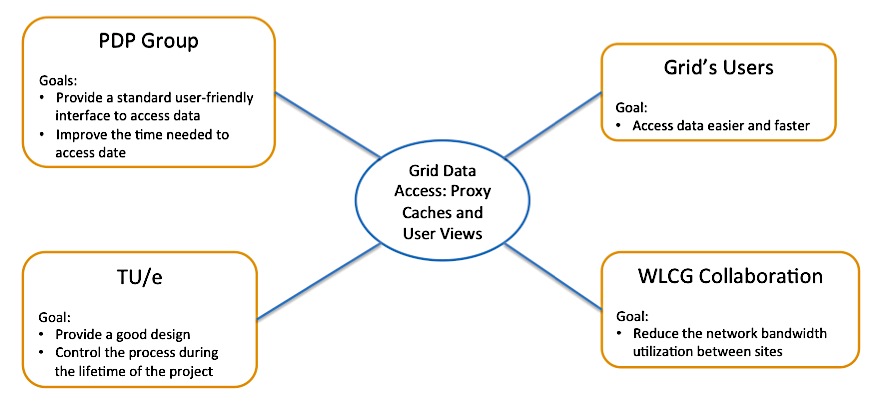


Figure – Stakeholders of the current project overview

Other relevant stakeholders include the software engineers and software designers that need an extensible and understandable design in order to implement new features. The deployment manager is also a relevant stakeholder due to the fact that he/she needs to deploy the current project and integrate it with the grid’s components, thus the project should be well documented.

## Deliverables

A set of deliverables of this project is defined according to the Nikhef and TU/e needs, and it is presented in the table below.

Table – List of deliverables

|  |  |
| --- | --- |
| *Deliverable* | *Description* |
| Prototype | A software project that publishes the data stored by the Grid through a standard protocol (WebDAV) |
| Technical report (this document) | Describes the design and implementation of the software, the feasibility study that includes all the problems encountered while combining standard and nonstandard technologies, as well as the results and conclusions of the current project. |
| Supporting documents | Project management, analysis document, architecture document and a user guide. |

## Overview of this report

Chapter 2 (*Domain Analysis*) provides basic information on the terminologies that will be used in the following chapters. In the first part, the Grid and the main modules that compose it are briefly presented. The problem analysis is presented in the last part of the chapter and it emphasizes the motivation for carrying out this project from user perspective as well as from the perspective of the resource provider.

Chapter 3 (*Feasibility Study*) provides a comprehensive analysis of the possible ways to implement the required functionality for the current project by implementing small prototypes. Firstly, the research focuses on interaction between the System Under Development (SUD) and the grid, and it continues with communication protocols, authentication methods and caching systems. At the end of each of the main sub-chapters conclusions and an overview of the research is presented.

Chapter 4 (*Requirement specification***)** presents the main functionality that needs to be implemented by the current project. It starts with a short introduction about the process that preceded the gathering of the requirements, and continues with presenting all the functional and non-functional requirements along with the motivation and the rationale for each of them.

Chapter 5 (*System architecture*)provides a comprehensive architectural overview of the current project. It presents a number of different architectural aspects of the system, defines the main components and the interaction between them by making use of several scenarios. It is intended to capture and convey the significant architectural decisions, which have been made on the system.

Chapter 6 (*System design*)provides a comprehensive overview of the design of the current project. In order to depict the software as accurately as possible, the structure of this chapter is based on the “4+1” model view of architecture. The design decisions that are made and their rationale are also presented within the next sections.

Chapter 7 (*Validation & Verification*) presents the methods used for validating the developed solution, as well as the description of the acceptance tests. The verification of the current project is addressed in the second part of this chapter and it describes the techniques used to confirm the correctness of the implementation. The experimental results from running the acceptance tests are discussed in the next chapters, in order to provide a good overview of the results.

Chapter 8 (*Results*) presents, in the first part, the deployment and test setup that was used in order to evaluate the acceptance tests defined in the previous chapter. In the second part, the experimental results of the test designed in order to interrogate the solution with respect the stakeholders’ needs are discussed, and short conclusions are formulated after each of the tests.

Chapter 9 (*Conclusions*) discusses the extent to which the goals of the project have been met. It presents in the first part the main conclusions according with the results from the previous chapter. In the second part, a set of future recommendations to improve the robustness and the performances of the current project are presented.

Chapter 10 (*Project Management*)introduces various issues relevant to project management. The process that was used to manage the project is described in the first part. Other related subjects like Breakdown structure, Milestone Trend Analysis, and Risk management are also presented in this section. A short retrospective of the project encloses the chapter.

# 

# Domain Analysis

*The chapter provides basic information on the terminologies that will be used in the following chapters. In the first part, the Grid and the main modules that compose it are briefly presented. The problem analysis is presented in the last part of the chapter and it emphasizes the motivation for carrying out this project from user perspective as well as from the perspective of the resource provider.*

## Introduction

A Grid can be defined in multiple ways, but in essence it is a collection of hardware and software that are integrated together in order to provide a solution for a particular problem. Based on the problem that is solved, the Grids are divided in two main categories:

* *Computational Grids* provide secure access to large pools of shared processing power (CPUs) suitable for high-throughput and computation-intensive applications.
* *Data Grids* provide an infrastructure to support data storage, data discovery, data handling, data publication, and data manipulation of large volumes of data actually stored in various heterogeneous databases and file systems.

The Grid can be seen as a distributed system, which is composed of a high number of relatively low-cost computers that combined together provide high performance. The parts that compose the distributed system can be located at one site/locations (local grid infrastructure), or they can be distributed across multiple sites/location in different countries or even continents (global grid infrastructure).

To integrate all the components of the grid, so called Grid Middlewares are used. They consist of multiple applications, frameworks and protocols that are installed on several computers of the grid, sometimes up to many thousands. The Grid Middlewares will ensure the good collaboration between sites/components, and should provide a uniform and transparent access of the user to the computing and storage facilities of the grid.

The Worldwide LHC Computing Grid (WLCG), European Grid Initiative (EGI) and Enabling Grids for E-sciencE (EGEE) are both Computational and Data Grids and they make use of the gLite middleware (which is presented in the next section). As an example, the WLCG organization is presented below:

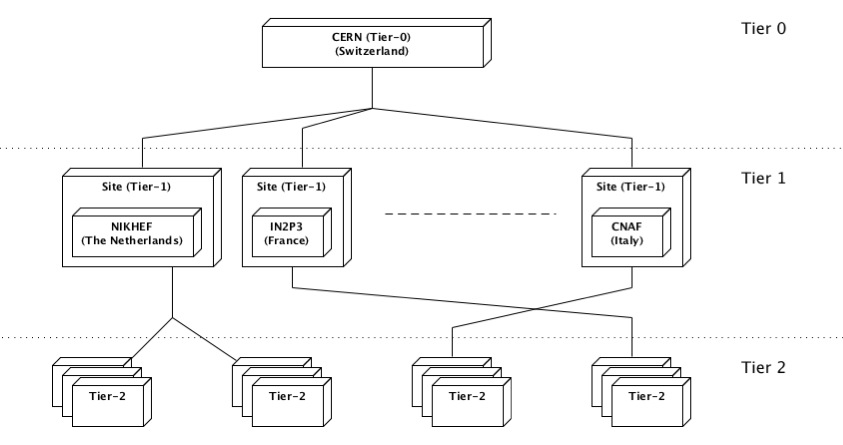


Figure – Schematic diagram of the WLCG Grid

The resources that compose the WLCG are distributed across different institutes in Europe. The grid has a hierarchical architecture, where the institutes that are part of the grid are grouped, by the amount of resources that they provide, in “Tiers”.

CERN is the only Tier-0 institute. It provides one of the largest computational and data storage facilities within the entire grid. Being a Tier-0 institute, it is the place where the data is produced, and from where it is distributed to the other Tiers that compose the grid.

The Tier-1 organizations (so called sites) are (national-wide) located in different countries within Europe. They provide computational and data storage facilities for the WLCG grid.

The Universities and/or Institutes compose the Tier-2 and usually they analyze the data produced by CERN and stored by the Tier-1 organizations.

The current project aims to provide additional services to all Tiers (0, 1 and 2).

## The gLite middleware

The gLite is a Service Oriented Grid lightweight middleware that provides several services. The most important ones are: jobs management and execution; data management and security; events logging; and information services that monitor the availability of resources [9]. A grid that uses the gLite middleware consists of several sites providing computing and storage resources that are interconnected by common information system and shared services [9]. An overview of the main components interesting for the current project is presented in the following figure and a brief description is given below.

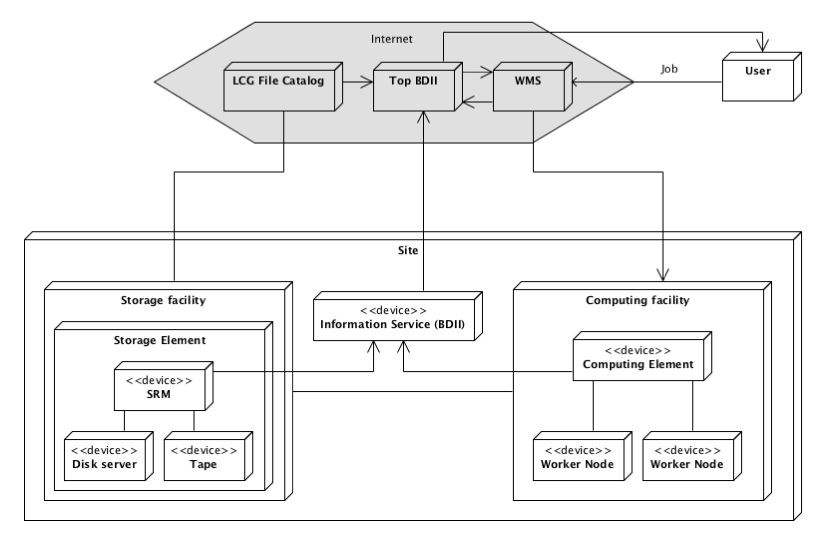


Figure – Simplified gLite Architecture (example of a site)

**Job** represents the primary unit that describes what the Grid needs to do (to calculate, to analyze). The Job Description Language (JDL) needs to be used to create a job that the grid can understand. Within a JDL the input files processed are specified, as well as the tool that is used for processing. Another important parameter that needs to be specified in the JDL is the location of the output generated (in order to be retrieved later on by the user).

**Worker Node (WN)** as its name suggests, is part of the computational facility of the Grid, and is used to execute the jobs that are sent by the user. The WN runs the job and performs all the processing required to resolve the task.

###### Computing Element (CE) is the gateway to the computing resources (Worker Nodes) localized at a site (i.e. a cluster, a computing farm) where the jobs are executed. In addition, the Computing Element includes a generic interface to the cluster and a batch system (Local Resource Management System (LRMS)) where jobs are sent by the user or by the WMS.

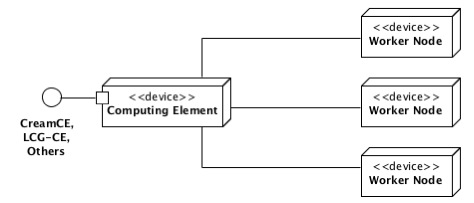


Figure – Computing Element (CE) overview

The jobs received by the CE are kept in different queues until the WNs become available to process them.

**Storage Element (SE)** is the component of the Grid that stores data. It consists in a collection of disk servers, large disk arrays or tape-based Mass Storage Systems (MSS). In order to save and retrieve data, the Storage Element provides different data access protocols and interfaces: GSIFTP (a GSI-secure FTP) is the protocol for whole-file transfers; RFIO and GSIDCAP are local and remote file access protocols. In order to provide a transparent and uniform access to data, the Storage Element introduces a new interface called SRM (Storage Resource Manager).

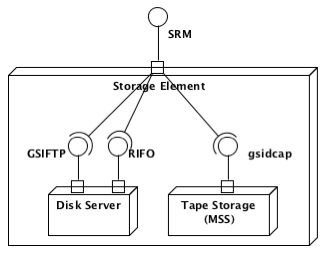


Figure – Storage Element (SE) overview

As described in the figure above, the SRM interface allows the user to access any data stored by that particular Storage Element regardless of the real location of the file on the disk servers or tape, or the protocol with which it can be accessed.

**Workload Management System (WMS)** is the central component of the Grid where all the jobs are sent. Basically, the purpose of the WMS is to accept user jobs and to assign them to the most appropriate Computing Element. In other words, once the WMS receives a job, it checks the availability of the input files at all the sites, and once a site that contains the majority of the files in the local Storage Elements is found, the job is sent to the Computing Element of that site. Also, in the case of multiple matching sites, it chooses the “best” one according to the user-specifications. Moreover, the WMS, records jobs statuses and stores their output.

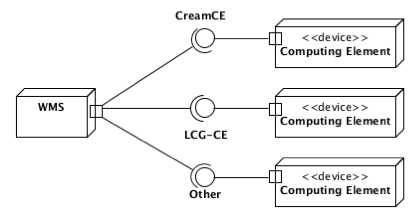


Figure – Workload Management System (WMS) overview

For the Storage Elements, the primary unit of data management is the *file*. The files usually have *replicas,* which are physical copies that are distributed at different sites, on different Storage Elements. Because all replicas need to be consistent, files should be immutable: they should be created once and never modified. They can be deleted and a modified version of them can be created again.

**Logical File Catalog** is a component responsible for keeping track of the location of the replicas on the Grid. It is in fact a database that maps the human-readable file names (LFN – Logical File Name) with their replica locations (SURL – Storage URL). The LFC provides transparent access to the actual files for users, so the user does not need to know where the file replicas are physically located.

As explained previously, all the files have a number of replicas (physical copies) on different SEs. Thus, for a given LFN a number of replicas are available. There is also possible for a file replica to belong to multiple LFNs. Figure 7 gives an overview about the interdependency between the LFNs and SURLs.

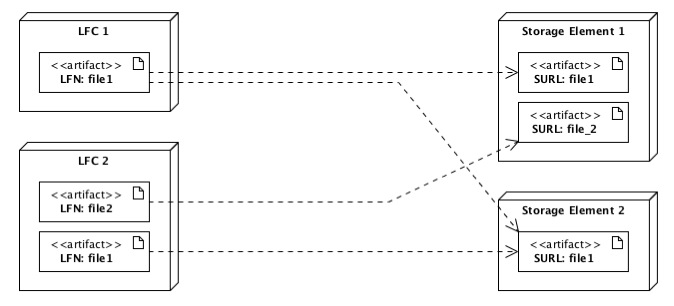


Figure – Example of interdependency between files and replicas

User Interface (UI) represents the user’s gate to interact with the Grid. Actually, it consists of a set of applications and libraries that are installed on a Linux machine. Usually the UI is provided separately on special machines where users have a personal account and the X.509 personal certificate is installed. The UI provides a set of new methods to access data, to send jobs or to interrogate the Information Service. Anyway, when it comes to data, the actual methods try to imitate as much as possible the Unix POSIX methods.

For example:

* **lcg-cp** is the equivalent of **cp** command, and it copies files from a Storage Element to the local machine, or even to another Storage Element
* **lfc-ls** is the equivalent of **ls** command, and it lists all the files and directories contained by a given LFC directory
* **lfc-del** is the equivalent of **rm** command, and it removes a file from the Storage Element and the LFC.

###### Users’ organization (Security)

The user community is grouped into Virtual Organizations (VOs), which usually overlap with the projects that make use of the Grid. A user must join an existing VO in order to be authenticated and authorized to use grid resources. The user authenticates to the Grid by making use of X.509 certificates and the Secure Sockets Layer (SSL) communication protocol, with extensions for single sign-on and delegation. The user can also make use of Grid Security Infrastructure (GSI) certificates, which are an extension of the X.509 certificates. A Certification Authority (CA) trusted by the Grid’s infrastructure running the middleware must issue the digital X.509 certificate.

## Public Key Infrastructure (PKI)

“The Public Key Infrastructure (PKI) consists of a set of hardware, software, people, policies, and procedures needed to create, manage, distribute, use and store digital certificates” [7]. The PKI enables users to securely transfer the data over the unsecure public network such as the Internet by making use of a public and a private cryptographic key pair (the digital certificate) that is obtained from a trusted authority (Certification Authority).

The Certification Authority is part of the PKI infrastructure and its main role is to sign and publish the public key (digital certificate) of a given user. The digital certificate represents the confirmation of the CA that the public key contained in the certificate belongs to a certain person or organization. Before signing the public key of the user, the CA asks for several juridical documents that confirm the real identity of the registrant. Once the CA has the guarantee that the entity that requested the certificate signature is exactly the entity that pretends to be, the digital certificate is issued and made publicly available. After the digital certificate is signed, the owner of the certificate can use it to identify himself/herself to 3rd party services on the Internet.

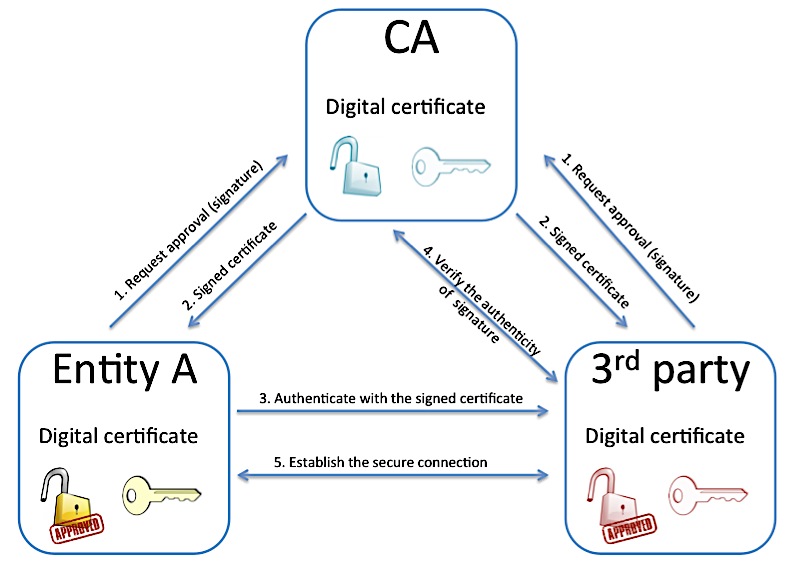


Figure – Overview of the Public Key Infrastructure and example of authentication to 3rd parties by making use of X.509 certificates

As presented in the figure above, the CA is the main component that ensures trust between entities and 3rd parties. The CA must maintain a high level of security for its private key that is used to sign all other digital certificates. For this reason, currently, there are only few Certification Authorities that are highly trusted by everyone. To increase the level of security, usually the digital certificates of the CAs are signed by the other CAs.

### Public and private key concept in PKI

The public and private keys for a user are created simultaneously using the same cryptographic algorithm by a certificate authority (CA) or by the user himself/herself. In the former case, the private key is given by the CA only to the requestor user and is never shared with anyone or sent across the Internet. In the latter case, when the user creates the public and private key, he/she sends a signing request to the CA. The signing request consists in the public key of the user (with minor additions). Once the CA receives it, the private key of the CA signs the public key of the user, and the result is sent back to the requestor in order to use it as public key. The public key is made publicly available in a directory that all parties can access.

The algorithms used to create the public and the private keys are called asymmetric cryptographic algorithms. The keys that are generated have the property that once a message is encrypted with the public key, it can be decrypted only with the pair private key, and vice versa.

Applying this concept on the Internet, the public keys are available to everyone. If, for example, A wants to send a private message to B, then A identifies the public key of B (from some sort of database, or directly from B). Then A encrypts the message by making use of the public key of B, and sends the message to B. Now the message can be decrypted only with the private key of B, which is the only entity that has it.

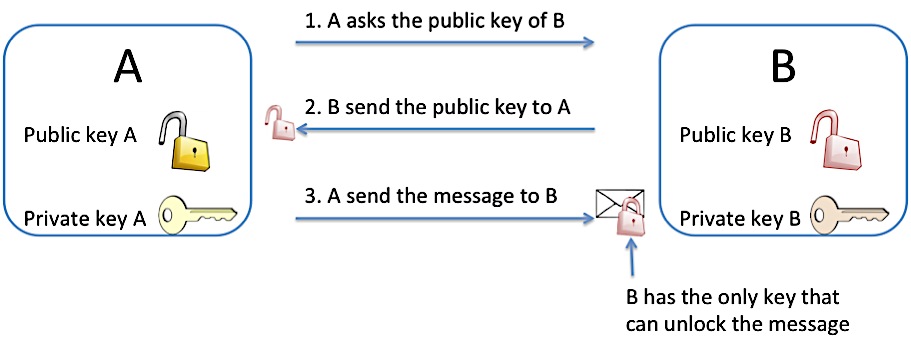


Figure – How the public and private keys work to securely exchange a message

### X.509 Certificates

The X.509 certificate is a widely used format for defining digital certificates. It mainly contains the identity of the owner, the identity of the issuer (the CA), the validity period, the public key (and the algorithm used to create it) and the signature received from the Certification Authority. It can have, depending on the version, different extensions and attributes.

### X.509 [GSI] Proxy Certificates

The X.509 Proxy Certificates are similar to the regular X.509 certificates. Actually, a proxy certificate consists of a new certificate (with a new public key in it) and a new private key. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The major differences between X.509 Proxy Certificates and X.509 Certificates are:

* The X.509 Proxy Certificates are valid a short period of time (few hours).
* The X.509 Proxy Certificates are signed by the X.509 Certificates (and not by the CA that issued the X.509 Certificate).

The X.509 Proxy Certificates must be seen as extensions of the real X.509 Certificates that carry (part) of the privileges that the issuer X.509 Certificates have. They are commonly used in security systems when one entity wishes to grant to another entity some set of its privileges.

The *X.509 GSI Proxy Certificates* are very similar to the X.509 Proxy Certificates, but unfortunately they do not follow entirely the standard specifications. Thus, only special applications and tools can use them. The GSI Proxy Certificates are considered legacy, but currently the Grid community extensively uses them.

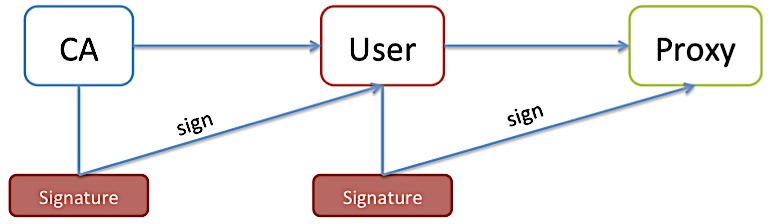


Figure – The X.509 [GSI] proxy certificate creation process (*Delegation*)

The *Delegation* is the process that creates the X.509 Proxy Certificates out of a X.509 Certificate. As name suggests, the proxy certificates are used to authorize a 3rd party to connect to various other systems on behalf of the real user (which have the original X.509 Certificate). In this way, the real user does not copy his/her digital certificate (that contains the private key) to other systems. If someone intercepts the proxy certificate, the level of damages is limited due to the short validity period when it can be used.

### Mutual Authentication

Mutual authentication is the process that establishes a trust relation between two parties. It consists of a set of operations, which ensure that the two parties are who they claim to be. Before mutual authentication can occur, the parties involved must first trust the CAs that signed the certificates of each other.

“To mutually authenticate, the first person (A) establishes a connection to the second person (B). To start the authentication process, A gives B his certificate. That certificate tells B who A is claiming to be (the identity), what A's public key is, and what CA is being used to certify the certificate. B will first make sure that the certificate is valid by checking the CA's digital signature to make sure that the CA actually signed the certificate and that the certificate has not been tampered with. (This is where B must trust the CA that signed A's certificate.)

Once B has checked out A's certificate, B must make sure that A really is the person identified in the certificate. B generates a random message and sends it to A, asking A to encrypt it. A encrypts the message using his private key, and sends it back to B. B decrypts the message using A's public key. If this results in the original random message, then B knows that A is who he says he is.

Now that B trusts A's identity, the same operation must happen in reverse. B sends A her certificate, A validates the certificate and sends a challenge message to be encrypted. B encrypts the message and sends it back to A, and A decrypts it and compares it with the original. If it matches, then A knows that B is who she says she is.

At this point, A and B have established a connection to each other and are certain that they know each others' identities.” [8]

### MyProxy Server

MyProxy is open source software server that manages X.509 proxy certificate. It combines an online credential repository and certificate authority to allow users to securely obtain credentials (proxy certificates) when and where needed [14].

It is used as follows: the user that has a X.509 certificate, delegates a proxy certificate to MyProxy server for usually 7 days, and the user defines a username and a password to access the proxy certificate. Now, within these 7 days, anytime the user needs a short proxy certificate (12 hours) to connect to the grid components, he/she only requests the certificate from the MyProxy Server using the username and the password. In this way, the user does not need to have the original X.509 certificate on the machine where the proxy certificate is required.

## Problem Analysis

The main goal of this project is to offer a user-friendly and efficient alternative way to access data stored on the grid. During the domain analysis, most of the components of the grid that are utilized in order to access data are analyzed, and some ideas about what the real problem is are identified. According to the survey that was carried out at the beginning of the project (in order to find out the difficulties that the users encounter while accessing data), the motivation of this project becomes very clear. It is explained in the next sections and it is described from user’s and resource provider’s perspectives.

### Motivation: user point of view

Currently, when the user needs to read a file that is stored by the grid, he/she utilizes some grid-specific commands and tools. As described at the beginning of this chapter, the User Interface (UI) is used to interact with the Grid. The user first needs to have the UI installed on the personal computer, or he/she needs to have access to utilize an already existing one, usually provided by the different sites.

To have the UI installed on the personal machine, the user needs to use a specific type of Linux and to install the right tools and frameworks, a process that is not always easy. To connect remotely to an already existing UI, he/she needs first to have an account and the user needs to have network access to that machine. The user also needs to have the personal X.509 certificate available there.

However, even if all the required steps presented before are met, the interaction with the grid’s storage elements is rather difficult. Some specific commands (that need to be known upfront) have to be used.

Firstly, the user must make sure that he/she has a valid proxy certificate on the UI machine, because the access to the Storage Elements is by default denied, unless the user presents a valid proxy certificate. Usually, to complete this step, the user needs to have the personal X.509 certificate available on the UI machine and he/she has to use some grid-specific commands to create the proxy certificate (such as *voms-proxy-init* or *grid-proxy-init*). An alternative is to use the *myproxy-logon* command to receive a proxy certificate from a MyProxy server.

Secondly, the user needs to execute the grid-specific commands (in the Terminal/Console) in order to copy the file to the UI machine. Once copied to the local machine the file can be read. Anyway, if the file needs to be analyzed with some advanced tools (such as Matlab), it needs to be copied again to a machine that has those tools.

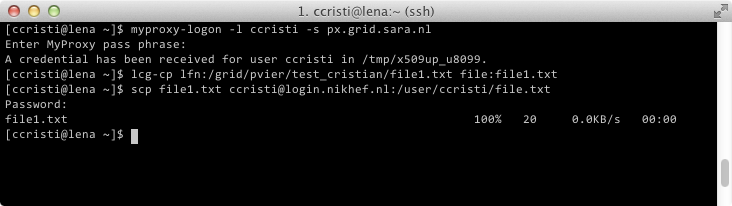


Figure – Example of how currently a file can be copied from the grid to a specific computer by making use of the UI interface

The example above shows all the commands required to copy a file stored by the grid, once the user logged-in the User Interface (UI).

* The user demands a proxy certificate from a MyProxy Server (px.grid.sara.nl) by using “myproxy-logon” command.
* The user copies the file from the grid to the UI machine, by using the “lcg-cp” command.

As a remark, the user needs to know exactly the path of the file from the Grid (“/grid/pvier/test\_cristian/file1.txt”). The UI does not provide the auto-complete functionality. If the path of the file is complex then it becomes very difficult for the user to specify the file that he/she needs.

* In the last step, the user copies the file from the UI machine to a machine where the data can be processed/analyzed. The file can be copied in various ways, by using different protocols that the UI provides.

Possible problems can occur while using the User Interface:

* The user does not have access to the UI machine, due to the network restrictions.
* The user cannot copy the file directly from the UI to its personal machine, again due to the network restrictions. Thus he/she needs probably to copy the file to other several computers until getting direct access to the file.
* The user might not have enough space available on the UI machine when a large amount of data is copied.

As presented, this is a difficult process that requires a lot of time spent only to make the file available where needed. The current project aims to simplify the entire process for data access, and it tries to make the files available whenever and wherever they are needed, without making use of any grid-specific commands, and preferably in a Graphic User Interface (GUI).

Another motivation for this project from a user perspective is that by providing a new interface to access data, the worker nodes can take benefits. More standard application can be deployed on them, and the palette of tools that users can utilize while executing jobs increases.

### Motivation: resource provider point of view

The project is also interesting from a resource provider’s perspective. Currently, the Worker Nodes are the most important data “consumers” from the grid. The grid is a distributed system across different sites (countries). Usually, the jobs are sent to the WMS in order to be executed. The WMS analyses the files that need to be processed, and decides to send the jobs to the sites where most of the files are stored locally. However, not all the files required by a job are always located at the site where the job is executed. Or, even if the files may all be at the site, it might happen that a disk server is down for maintenance, and then some files might be inaccessible. For example, a common case is when jobs that have a large number of input files (>100) at least one of the files is not accessible. Thus, the missing (or inaccessible) files need to be copied from the other sites.

If several jobs are demanding the same set of data several times, then the data is copied several times between sites. This is considered a slow and inefficient process, and it would make sense that once a file is copied from an external site, it should be stored locally (cached) for a relatively short period of time, in case other jobs need it.

From resource provider point of view, copying data between different sites uses the limited network bandwidth available. By copying the same data several times, the network utilization would be inefficiently used (waste). Caching the data at the site level, the network bandwidth would be used less, thus the efficiency of the already existing resources would increase.

Other than optimizing the resource consumption, the overall performances of the system can be improved even if the files are located at the same site where the jobs are executed. According to some measurements, in order to copy a file from a Storage Element, the user/WN needs to wait in average around **7 seconds** before the transfer of data starts. This is the overhead introduced by the SRM interface in order to locate, to check permission and to establish the connection between the client and the machine that stores the file. The time introduced by the overhead, is comparable with the time required to transfer a 1GB file (around 10 seconds). Basically, the overhead almost doubles the time required to copy the file, resulting in around 17 seconds.

Some other measurements are carried out to determine a rough estimate of the number of files of a Storage Element that are used more that once during a day. The results showed that around 30% of the files that are used within one day are read more than once. This means that for each re-read (can be tens of times) of the 30% of the files, 7 seconds are lost due to the overhead. This leads to a high waste of precious resources and time.

By developing the current project, a more efficient mechanism to identify a file and to check the permissions can be implemented in order to eliminate or to reduce as much as possible the overhead. Doing this would improve the overall performance of the system.

# Feasibility Study

*The**chapter provides a comprehensive analysis of the possible ways to implement the required functionality for the current project by implementing small prototypes. Firstly, the research focuses on interaction between the System Under Development (SUD) and the grid, and it continues with communication protocols, authentication methods and caching systems. At the end of each of the main sub-chapters conclusions and overview of the research is presented.*

## *Introduction*

At the beginning of the project the requirements regarding how user interacts with the system were rather vague. One thing was clear though: the current project must interact with the LCG File Catalog and the Storage Element components of the grid. For this reason, the feasibility study starts with deep research on how to use the gLite[[5]](#footnote-5) middleware framework provided by the grid. The research focused on studying documentation published on the Internet about gLite, and on implementing small prototypes to prove that the level of knowledge is sufficient for the real implementation of the current project. In the second part of the feasibility study, the emphasis is on the communication protocol between the user and the SUD, as well as on technologies to use in order to provide required authentication methods.

Before the start of the feasibility study, the development environment is defined, and it is presented in the next section.

## Development environment

According to the requirements of the project, the SUD must operate on CentOS 5[[6]](#footnote-6) (operating system) machine. For this reason, the chosen development environment is identical with the required one. More than that, the Unix repositories with standard and gLite libraries that can be used in the further developments are identified. The full development environment (including the Unix repositories) description is presented in the *Appendix H*.

## Interaction with the Grid

As presented in the *Domain Analysis* chapter, gLite middleware facilitates the interaction with various components of the grid. It defines different interfaces for the main components, in several programming languages, which are generally used to develop new features. According to the requirements, the SUD must publish the files defined by the LFC and SE, so it interacts with the two components, and the next sections present the findings of the feasibility study.

Note: *In order to interact with any of the servers (LFC or SE), the user must authenticate by using a X.509 (proxy) certificate. For the first two prototypes, the proxy certificate is created manually and specified as input parameter.*

### Interaction with the LFC

The LFC (LCG File Catalog) is the main File Catalog that keeps the mapping between a user-friendly file name (LFN) and the real name of the file as stored by the Storage Element (SURL).

In accordance with the documentation found on the Internet [4] about the LFC API interface defined by the gLite middleware, a prototype written in Python is developed and the required operations (i.e., browsing directories, getting the permissions and information about a file) are tested.

Firstly, the required library is identified on the gLite repository (*lfc-python*) and installed to the development machine. Secondly, the methods that need to be used in order to achieve the required functionality are identified, and presented below.

Table – Important methods provided by LFC Python API

|  |  |  |
| --- | --- | --- |
| *Package* | *Method* | *Description* |
| lfc | lfc\_access | Checks whether the user is allowed to read/write a file according to the LFC |
| lfc\_stat | Returns detailed information about a file defined by LFC |
| lfc\_opendir  lfc\_readdirx  lfc\_closedir | Returns the list of files contained by a directory defined by the LFC |

The implementation of the prototype is clear and concise, and it proves that these commands are sufficient to explore the LFC file catalog as required by the SUD.

### Interaction with the Storage Element (via SRM)

The Storage Element is the main component of the grid that stores the content of the files defined by the LFC (as well as other files not registered on the LFC). The SE provides the SRM interface, which is used to access the data stored (by providing the SURL). However, according to the requirements, the current project must publish the files stored by the SE and must make the data available for the user. In other words, the SUD interacts with the SE to access (copy) data, and to browse files and directories.

Again, the libraries and frameworks that can be used to achieve the goal are identified (*GFAL-client-py* and *lcg\_util-py*). In accordance with their documentation [4, 5] the methods that provide the required functionality are analyzed (and presented below). These methods will be used in the prototype.

Table – Important methods provided by GFAL and LCG\_UTIL Python APIs

|  |  |  |
| --- | --- | --- |
| *Library* | *Method* | *Description* |
| gfal | gfal\_access | Checks whether the user is allowed to read/write a file according with the SE |
| gfal\_stat | Returns detailed information about a file defined by SE |
| gfal\_ls | Returns the list of files contained by a directory defined by the SE (SRM interface) |
| lcg\_util | lcg\_cp | Copies a file defined as SURL to the local computer. |

The prototype proves to work in principle, but sometimes it generates “*Segmentation fault[[7]](#footnote-7)*” error. As the prototype is written in Python, this error cannot be generated by the prototype itself. The *gfal* and *lcg\_util* libraries that are written in C must generate the error. An extensive research is done to identify the reason if possible, or at least the conditions under which the prototype throws the error. The outcome of the research does not reveal the reason of the error, but only the scenario when it occurs. It seems that when both of the libraries are used in the same Unix-process there is a high chance for this error to appear. When the libraries are used in different processes, the error never occurs. Due to the fact that there are not many alternatives to implement the prototype (by using other libraries), this behavior is taken into account when designing and implementing the SUD.

### Conclusions

To conclude, the two prototypes that are implemented prove that it is possible to interact with the LFC and SE easily, by simply using the API provided by the gLite middleware. The APIs are available in Python, Perl and Java, so from implementation point of view there do not seem to be any issues. The only problem found is the *Segmentation fault* but anyway, a design of the SUD ensuring that the two libraries are used in different processes can mitigate this issue.

An additional test is carried out in order to profile the performances of the two prototypes, and in fact to measure the response time of LFC and SE. Test results indicate that 100 to 200ms are needed to get an answer for a request from the LFC. The time required to obtain a response from the SE is higher (around 500ms). However, the tests are performed against SE and LFC servers that are located at Nikhef. Most probably, the duration of the request might be (much) higher than what is measured.

Again, the measurements are important, and the design of the current project can take it into account by making use of certain memory-cache tools.

## Communication interface for the user

According to the vague initial requirements (mount the Grid’s data to the local computer; make use of standard technologies), to the survey results [3] and to the input from the main stakeholder (PDP Group), the WebDAV is proposed as the communication protocol between user and SUD. The further research focuses on possible ways to integrate the WebDAV protocol with the Grid.

### WebDAV protocol

The Web-based Distributed Authoring and Versioning (WebDAV) is an extension of the HTTP protocol and it is specified by RFC4918[[8]](#footnote-8).  The intention of the WebDAV protocol is to ensure that the World Wide Web is a readable and writable medium [12]. Therefore, it provides the ability to remotely create, change and move documents to a server (usually a web server). This allows documents to be updated on a website, files to be saved via the web, and re-opened.

Preliminary research on the Internet about the WebDAV protocol revealed that it is a widely used communication protocol. Moreover, most modern operating systems support WebDAV and they allow users to mount the WebDAV repository to the local computer. In other words the user sees the files on a WebDAV server as if they were locally on the computer. The findings cover the initial requirements perfectly.

As specified previously, the WebDAV is an extension of the HTTP protocol. Actually, it adds several methods and headers to the already existing HTTP protocols.

Table – WebDAV specific methods added on top of HTTP [12]

|  |  |
| --- | --- |
| *Method* | *Description* |
| PROPFIND | Used to retrieve properties, stored as XML, from a resource. The method is also overloaded to allow one to retrieve the directory hierarchy of a remote system (WebDAV repository). |
| PROPPATCH | Used to change and delete properties on a resource (file or directory) in a single atomic act. |
| MKCOL | Used to create collections (directories). |
| COPY | Used to copy a resource from one URL to another. |
| MOVE | Used to move a resource from one URL to another. |
| LOCK | Used to put a lock on a resource. |
| UNLOCK | Used to remove a lock from a resource. |
| OPTIONS | Used to communicate to the client which of the previous methods is provided. |

However, the intention of the current project, in accordance with the initial requirements, is to provide read-only access to the data stored by the grid. This means that out of the entire set of new methods introduced by the WebDAV, only two would be used (*PROPFIND* and *OPTIONS*).

An extensive research (including several prototypes) is conducted in order to check whether it is possible (feasible) to integrate the WebDAV protocol with the Grid technologies, and the outcome is presented in the following sections.

### Using WebDAV

WebDAV protocol being an extension of HTTP, it is usually deployed on web servers. To support the WebDAV protocol, the servers usually provide some sort of additional modules that resolve the protocol specific methods.

A set of web servers are identified on the Internet and investigated with emphasis on their WebDAV support. Due to the limited amount of time available for the feasibility study, only three most popular open-source web servers are analyzed: Apache2, NginX and Lighttpd.

Table – WebDAV support provided by the Apache2, NginX and Lighttpd

|  |  |  |
| --- | --- | --- |
| *Web server* | *WebDAV module* | *Description* |
| Apache2 | mod\_dav mod\_davfs | Provides class 1 and class 2 WebDAV, thus all the additional methods described by the protocol specifications are fully supported. |
| NginX | HttpDavModule | Provides an incomplete implementation of the WebDAV protocol. Only PUT, DELETE, MKCOL, COPY and MOVE methods are implemented. PROPFIND method, which allows browsing through files and directories, is not implemented at all. |
| Lighttpd | mod\_webdav | Provides a minimalistic implementation of the original protocol. The available methods include PROPFIND, OPTIONS, MKCOL, DELETE and PUT. |

According to the research and to several web forums, the Apache2 has the most complete and stable support for the WebDAV protocol. However, due to the fact that the current project aims to provide only read-only access to the data, it does not make use of all the functionality provided by the WebDAV protocol. This means that at the current stage, the other two web servers (NginX and Lighttpd) are also interesting for the project.

Further research tries to figure out how the Grid technologies can be integrated with the WebDAV support provided by the various web servers, and in particular how the files defined by the LFC and SE can be published with the WebDAV protocol.

The experiments and the findings on the Internet show that all the WebDAV modules provided by the different web servers usually publish files and directories that are stored on the local machine where the server is deployed, or at least that are accessible through POSIX interface. In other words, files or directories that are published by the web server must be (virtually) available in the file-system.

This is a problem, due to the fact that the files and directories published by the LFC or SE are not real (or virtual) files located on the server’s file system. That means these cannot be integrated directly with the WebDAV protocol. To solve this issue there are two alternatives: (1) manage to make the files defined by LFC and SE as virtual files stored by the web server’s file system; or (2) implement a new module for the web server that provides WebDAV protocol (according to the original specification) but instead of publishing files form the local file system, it publishes directly the files from LFC and SE.

The first approach could be implemented by making use of FUSE[[9]](#footnote-9). A prototype is implemented to check whether it works and it is a feasible solution and it is presented in section 3.4.3. The second approach verifies the feasibility to develop from scratch a module that implements the WebDAV protocol (part of it – to provide read-only access). The prototype is presented in section 3.4.4

### Approach 1: HTTP Server + WebDAV + FUSE mount

This approach tries to virtually mount to the web server’s file system the files defined by the LFC and SE. According to the research, the FUSE is the right technology to achieve the goal of the current approach.

FUSE provides a simple API that can be extended and it allows developers to build a virtual file system along with the file system provided by the operating system. A small example is presented below, for a better understanding of how FUSE works.

Table – Overview of a virtual file system implemented with FUSE

|  |  |  |
| --- | --- | --- |
| *Files and directories hierarchy* | *Location* | *Description* |
| /home | Disk | The files and directories physically exist on the local disk of the test machine. |
| /home/user1/ |
| /home/user1/folder1 |
| /home/user1/file1 |
| **/home/fuse\_folder** | FUSE | The file and the directory are virtually introduced in the file system of the operating system. They do not physically exist on the machine. |
| **/home/fuse\_folder/fuse\_file** |

As presented in the example, the */home/fuse\_folder* and */home/fuse\_folder/fuse\_file* are introduced in the file system of the machine even if physically they do not exist. Based on this, by using FUSE it is possible to introduce into the file system the files and directories that are defined by the LFC and stored by the SE. This would make possible to publish these files with the WebDAV support provided by various web servers (because the LFC and SE files are now (virtually) introduced to the server’s file system).

A prototype is implemented to check whether the findings of the research are correct and sufficient to achieve the goal. Firstly, a FUSE application that publishes the files defined by LFC is developed. The prototypes written previously (interaction with the Grid) are rewritten in C++ and afterwards are integrated with the FUSE API.

Table – Logical description of FUSE application that publishes the files from LFC

|  |  |  |
| --- | --- | --- |
| *FUSE method* | *gLite method* | *Description* |
| access | lfc\_access | Checks user permission to access a file. |
| stat | lfc\_stat | Returns detailed information about a file. |
| opendir | lfc\_opendir | Lists all the resources contained by a directory. |
| readdir | lfc\_readdirx |
| closedir | lfc\_closedir |
| read | lcg\_cp | Copies the file from the SE locally and reads it. |

The amount of code written is rather small and the result is in accordance with the needs for this prototype. In conclusion, it is possible to build a FUSE application that mounts to the local system the LFC folder.

According to the main requirement, the SUD must allow the user to mount the LFC to a computer. Actually this prototype fulfills entirely the requirement, and the usage of WebDAV protocol becomes questionable. However, in order to use the FUSE prototype, several libraries and frameworks (here included Grid frameworks) must be previously installed. This contradicts the requirement about the Usability of the project. Anyway, this is an option that might be useful in the future.

The main goal of the current prototype is to publish the LFC files through WebDAV protocol. Once the LFC files are introduced (mounted) to the development environment, the three web servers previously investigated are tested to check if the files are correctly published. The web servers are installed on the development machine, and configured according to their specification to publish a directory from that machine (the FUSE directory). All the web servers prove to correctly publish the LFC files as a WebDAV repository. The newly built repository is mounted to a test machine and everything proves to work according to the needs for the project.

As described in section 3.3 , in order to interact with the Grid a X.509 proxy certificate must be used for authentication. Again, for this prototype, the proxy certificate is created manually and specified as input parameter for the FUSE application.

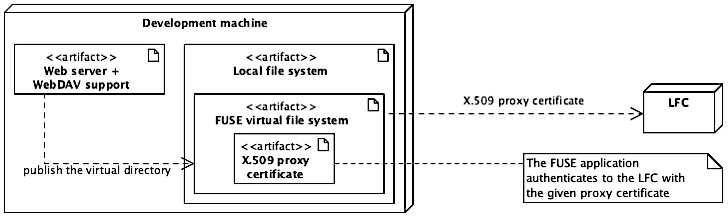


Figure – Overview of the Approach 1 (FUSE + standard WebDAV)

At this moment the prototype publishes the LFC files by using only one set of credentials (the manually created proxy certificate) to authenticate to the Grid. It is easily possible to extend it to publish the files defined by the SE. The goal of the current prototype is achieved. However, further research is carried out to find a way to manage multiple users (proxy certificates) in the same time.

Because the WebDAV server and the FUSE application are different processes, once the WebDAV server authenticates a user, it must dynamically provide to the FUSE application the right proxy certificate for that user. The FUSE application must manage the proxy certificates that are received from server, and it must publish to a user only the files and directories available for that particular user. This is a major issue that proves to be difficult to solve because when the WebDAV server queries the virtual file system for directories and files, it does not specify the user on whose behalf it performs the requests.

Anyway, a work around for this problem is implemented as follows: any user that uses the WebDAV repository is associated with a separate directory in the virtual file system. The FUSE application then, based on the directory that is queried, decides which proxy certificate to use when interacting with the Grid. The example below shows the solution found.

Table – Approach 1: How multiple users are managed by the FUSE application

|  |  |
| --- | --- |
| *Virtual directory (FUSE)* | *Description* |
| /FUSE\_ROOT/user1/folder1/… | When *user1* authenticates to the WebDAV repository, it will browse only his/her virtual directory. The FUSE will use his/her proxy certificate to interact with the LFC. |
| /FUSE\_ROOT/user1/folder2/… |
| /FUSE\_ROOT/user2/folder1/… | The same happens when *user2* authenticates. |
| /FUSE\_ROOT/user2/folder3/… |

To conclude, this solution seems to have the appropriate behavior, and the current approach (FUSE + standard WebDAV implementation) proves to be feasible.

### Approach 2: HTTP Server + WebDAV custom implementation

The second approach checks whether it is possible to implement the WebDAV protocol from scratch in order to publish the LFC and SE files.

An extensive search on the Internet is conducted to find already existing open-source WebDAV implementations in order to adapt them to publish the required files. The research shows that there are multiple implementations that aim to follow the WebDAV protocol specification (most of them partially). They seem to be very complex, hard to understand, thus difficult to modify and adapt to the needs. However, several ideas that can help to implement the protocol from scratch were extracted.

At the first view, according to the WebDAV specification, it seems to be complicated to fully implement the WebDAV protocol, and not feasible for the current project due to the limited (short) amount of time. But, at a closer look, as mentioned in the requirements, the WebDAV repository must be read-only. For building a read-only repository, obviously it is enough to implement only a small part. In fact, only two methods out of seven must be implemented and these are PROPFIND and OPTIONS. A read-only repository must provide: user authentication, directories and files browsing, and obviously files reading. These operations can be mapped with the WebDAV protocol as follows:

Table – Mapping between WebDAV methods and required functionality

|  |  |  |
| --- | --- | --- |
| *Operation* | *WebDAV method* | *Description* |
| Authentication | - | Is done by the web server when the HTTP session is initialized |
| Browse directories | PROPFIND | Returns the files defined on LFC or SE |
| Read file | GET (HTTP) | Returns the content of the file |
|  | OPTIONS | Returns the methods supported by the repository (PROPFIND and GET) |

Based on the open source WebDAV implementations that are analyzed, and based on the research about how to implement additional methods for the HTTP protocol, the current prototype implements the desired functionality as a Common Gateway Interface (CGI)[[10]](#footnote-10) application. It is implemented in Python, and deployed on all the three web servers that were investigated previously.

The CGI application is called anytime the web server receives a request from the user. Once the application receives the request, it identifies the method that is required by the client. If it is PROPFIND, then it interrogates the LFC and returns in the WebDAV specific-PROPFIND-response-format the content of the required directory. If the requested method is GET, then the CGI application downloads the file from the SE and makes it available for the user.

Again, at this moment, the prototype uses a proxy certificate that is manually specified in the CGI application. The extension of the prototype for management of multiple users seems to be much easier than in the previous approach. Once a user requires something from the WebDAV repository, the web server authenticates the user and sends his/her identity to the CGI application. The CGI application, based on the received identity of the user, can decide what proxy certificate to use to interact with the Grid.

This approach looks very promising because it offers full control of what is published by the WebDAV repository for every user, and it is also easy to implement.

However, the problem with this approach is that the WebDAV needs to be implemented accordingly with the protocol specifications, which sometimes may be a delicate task. Moreover, the compatibility with various WebDAV clients must be checked manually, because it is a custom implementation written from scratch.

The robustness of the CGI application is very high, in the sense that if something happens (crashes from various reasons), the server starts it automatically and reprocesses the request received from the user.

From performance perspective, the prototype proves to be rather slow because of the usage of CGI, which for each request instantiates a new process. The solution for this problem is to use FCGI (Fast CGI), which is expected to be faster. Transforming the CGI application into FCGI seems to be easy and straightforward, thus the prototype is modified to use the new technology. The result shows a major improvement of the performances.

### Conclusions

During the feasibility study regarding WebDAV implementation, two approaches are investigated:

* Usage of a standard WebDAV implementation provided by various web servers, in combination with a FUSE application that connects to the grid.
* Custom implementation of WebDAV protocol as a [F] CGI application that connects to the grid components.

The results prove that it is possible to publish the files defined by the LFC and SE by making use of WebDAV protocol with both approaches. The following table summarizes the findings on the two approaches investigated.

Table – Overview of the two approaches in order to provide WebDAV support

|  |  |  |
| --- | --- | --- |
| *Aspect* | *Approach 1* | *Approach 2* |
| Publish files defined by LFC and SE with WebDAV | Yes | Yes |
| WebDAV clients compatibility with approach | High | Medium[[11]](#footnote-11) |
| Effort to implement | Medium | Low |
| Possibility to customize the WebDAV repository[[12]](#footnote-12) | Low | High |
| Effort to manage different users | Medium | Low |
| Robustness | Medium[[13]](#footnote-13) | High[[14]](#footnote-14) |

To conclude, it is feasible to provide WebDAV access to grid’s data and to integrate standard technologies (HTTP, WebDAV) with grid’s non-standard technologies.

Due to the time limitation, no other possible communication protocols that can be used to access data were investigated. The WebDAV is considered good enough to continue the research about the other issues that need to be solved in order to provide all the required functionality.

## Authentication methods

Currently, not everyone is allowed to access any data stored on the Grid. Although the current project aims to provide an easier way to access the data, this should not be achieved at the expense of a reduction in data security.

The feasibility study tries to discover the right standard technologies that can be used to achieve the goal of the project from authentication perspective, and the findings are presented in the following sections.

### PKI authentication

As explained in the previous sections, in order to communicate with the Grid, the user needs to authenticate himself/herself with the X.509 (proxy) security certificate. According to the requirements, the SUD must allow the user to authenticate by making use of his/her X.509 certificate. Moreover, the user can use X.509 proxy certificate (which is standard) but also a GSI proxy certificate. The latter is a modified X.509 proxy certificate, created in the early stage of the Grid, but that is still used nowadays.

The feasibility study about PKI authentication is a continuation of the research on WebDAV. In fact, it is an attempt to add the PKI authentication on top of previous implemented prototypes for WebDAV. Because, the WebDAV implementation is deployed on a Web server, the research focuses on integration of PKI authentication with Web servers.

Through the Transport Layer Security (TLS) mechanism as described in [13] an authenticated and encrypted connection is established between both parties based on their key pairs and their X.509 certificates. The process establishes *mutual authentication*, and it means that once the connection is established, the server knows exactly with whom it is communicating.

Firstly, the emphasis is placed on finding ways to set up the (mutual) authentication on the Web servers that are investigated, and testing whether all the certificate types (X.509 ((GSI) proxy)) are accepted for authentication. The results revealed that all of the investigated servers accept for authentication the X.509 certificate and X.509 proxy certificate (both standard technologies). None of them can authenticate a user that presents a GSI proxy certificates. This is a major issue, due to the fact that this format of certificates is widely used.

To solve the problem there are two alternatives: (1) to modify the web servers (or their modules that provide the HTTPS) in order to accept the GSI certificate type, or (2) to implement some additional module (when possible) that handles separately the GSI certificates. The first option is not taken into consideration, because once the source-code of a widely used tool (Web server, module) is modified, its maintenance becomes very difficult. The second option, as explained earlier, is to implement an additional module for the web server that would manage the GSI certificates.

Fortunately, after some research on the Internet it turned out that such a module already exists for Apache2, and it is called *mod\_gridsite[[15]](#footnote-15)* (part of GridSite[[16]](#footnote-16) project).

Based on this, the research focuses on Apache2 and it tries to integrate the *mod\_gridsite* module into the web server. Once the module is integrated, the Web server is able to identify the user that authenticates with the GSI proxy certificate. Now, the goal is achieved: the server can identify a user that presents for authentication a X.509 certificate, X.509 proxy certificate and X.509 GSI proxy certificate.

Secondly, once the web server knows the identity of the user, in order to interact with the grid on behalf of the identified user, it must have a proxy certificate of that user. It means that the server must, somehow, receive from the user a proxy certificate that can be employed further to interact with the Grid. This process is called delegation, and again, it can be implemented by making use of two other components offered by GridSite (*gridsite-delegation.cgi[[17]](#footnote-17)* on the server side, and *htproxyput[[18]](#footnote-18)* on the client side). The gridsite-delegation.cgi application, once deployed on a web server, publishes a web-service that allows *htproxyput* client to connect and to perform the delegation. In order to work, it needs the *mod\_gridsite* module, thus it works only on Apache2 web server.

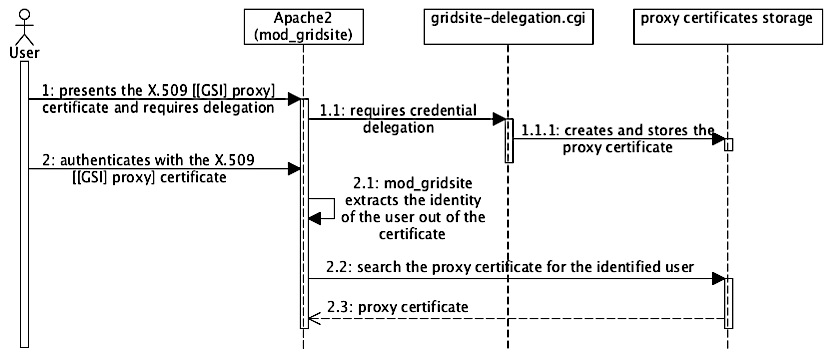


Figure – PKI authentication with Apache2 and GridSite sequence diagram

So far, the feasibility study shows that Apache2 can support the PKI authentication type with minimal effort, whereas the other web servers (NginX and Lighttpd) require some development in order to accept the GSI proxy certificates. The credential delegation mechanism can easily be implemented with the GridSite suite in combination with the Apache2 web server.

Table – Overview of the PKI authentication solution (Apache2 + GridSite)

|  |  |
| --- | --- |
| *Aspect* | *PKI authentication (Apache2, GridSite)* |
| Possible to implement PKI authentication? | **Yes** |
| * Effort required to implement | **Easy** |
| Possible to implement delegation? | **Yes** |
| * Effort to implement | **Easy** |
| * Delegation can be done from any OS? | **No**, *htproxyput* works only on Linux |
| VOMS attributes support | **Yes** (the newly created proxy certificate contains the VOMS attributes of the delegator. |

In conclusion, the required PKI authentication can fully and easily be implemented if **Apache2** is used as web server in combination with **GridSite**. The only problem would be that the delegation client runs only on Linux.

### Username and password authentication

As explained, to communicate with the Grid, the user needs to authenticate himself/herself with the X.509 (proxy) security certificate. Because the current project tries to ease the access to data from user perspective, an alternative authentication method (username and password) must be provided.

The new method can be easily implemented on any of the investigated Web servers as basic or digest authentication. Nevertheless, even if the user authenticates with the username and the password, the SUD still needs to interact with the Grid, thus it needs to use a proxy certificate. The user must somehow delegate or upload a valid proxy certificate to the SUD that maps it to the username needed for login. The entire process to implement the new authentication method is summarized in the following table.

Table – Logical steps to provide username and password authentication

|  |  |
| --- | --- |
| *Step 1* | *The user defines a username and a password pair on SUD* |
| Solution | A web interface can be provided to allow users to define a username and a password |
| *Step 2* | *Delegate user X.509 certificate to the SUD* |
| Solution | The GridSite framework, as discovered in the previous section, can easily be used to create proxy certificates on SUD |
| *Step 3* | *Map the username with the created proxy certificate* |
| Solution | An additional module for the Web servers (when possible) or a [F] CGI application can be implemented to map the username with the right proxy certificate |

Anyway, the findings of the research showed that there is already implemented an Apache2 module that provides basic authentication (username and password) and based on given credentials at login it requests a proxy certificate from a MyProxy Server. The module is called *mod\_authn\_myproxy[[19]](#footnote-19)*. In other words, all the steps described before, are fully implemented by the MyProxy suite as follows: the user defines a username and password pair when uploading a proxy certificate on the MyProxy server (the mapping between username and proxy certificate is implicit); the Apache2 module provides username and password authentication, checks the credentials against the MyProxy server, and creates a proxy certificate on the SUD.

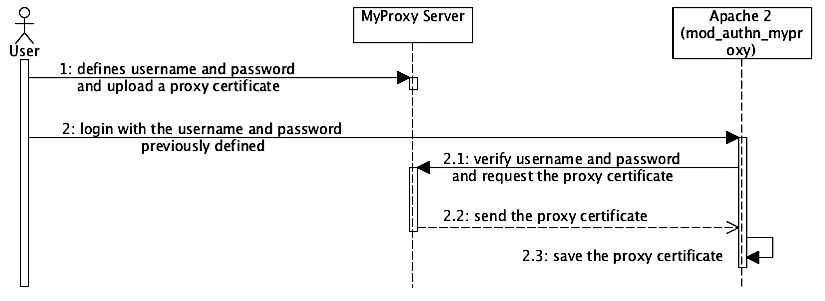


Figure – Overview of the username and password authentication method provided by MyProxy Server and the Apache2 module of MyProxy

As a remark, the *mod\_authn\_myproxy* is not provided as a compiled, ready-to-use Apache 2 module. The corresponding source code is available on the Internet, and it needs to be compiled manually, process that is not straightforward and several libraries need to be installed to the development system. The full description of the script that compiles the module is presented in the *Appendix E*.

Because the project aims to be user friendly, all the steps described above, that need to be performed by the user, have to be easy to realize. In order to define the username and the password and to upload the credentials (the proxy certificate) to MyProxy servers, the user has several alternatives out of which only two are interesting: the command line tool provided by MyProxy suite, and *Grid Proxy Manager[[20]](#footnote-20)* plugin for Firefox.

The command line tool is available only for Linux and a set of libraries needs to be installed upfront. The user needs to have the personal X.509 certificate on the machine where the command is executed. The tool allows the user to choose the username and the password, the MyProxy server hostname, and the Virtual Organization attributes that need to be added to the proxy certificate. An example of usage is presented below.

|  |
| --- |
| myproxy-init –l username –s myproxy\_hostname –voms VO |

The *Grid Proxy Manager* plugin works on any operating system supported by Firefox. The X.509 certificate of the user needs to be imported to the web browser before utilizing the plugin. The user can choose the username and the password and the MyProxy Server where the credentials need to be uploaded. However, there is no possibility to specify the Virtual Organization (VO) attributes that need to be added to the delegated proxy certificate. This is a major issue, and without a solution to the problem, this plugin would be useless for the user.

Because the Firefox plugin is publically available and it is found on the Firefox plugins repository, to add the missing functionality to it, and provide to the users a modified version of the plugin is not feasible.

During the feasibility study, a compromise solution that does not require a significant amount of time to be implemented is identified. As described in Figure 14 the Firefox plugin uploads a X.509 proxy certificate to the MyProxy server. The uploaded proxy certificate does not contain any VOMS attributes. When a user does login to the SUD, the username and password are sent to the *mod\_authn\_myproxy* module, which connects the MyProxy Server and retrieves the proxy certificate. The proxy certificate received is saved into a file by the Apache2 module (see Figure 14). Anyway, the solution consist in some minor changes into the *mod\_authn\_myproxy* module in the sense that once the proxy certificate is received, the module adds automatically the VOMS attributes to the proxy certificate and afterwards saves it into a file. The VOMS attributes that the user wants to add to the proxy certificate can be specified within the username. As an example, if the user id defined by the user is called “joe”, then in order to specify the VO, the user id as login can be “joe#<voname>” (e.g. “joe#atlas”). The mod\_authn\_myproxy module would need somehow to extract the “#voname” (e.g. “#atlas”) out of the received user id as login, and add the VOMS attributes to the proxy certificate.

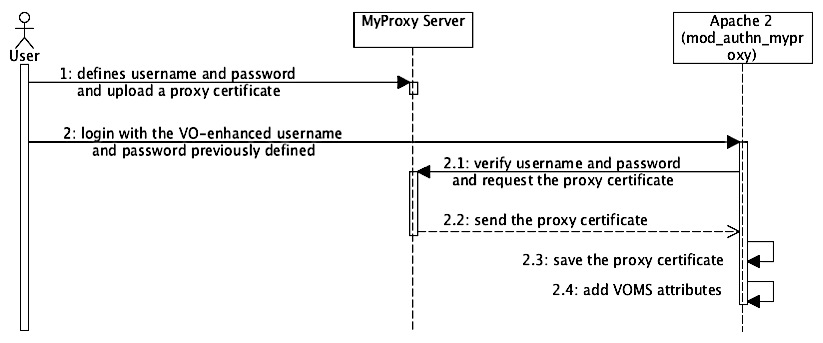


Figure – Overview of the process that adds VOMS attributes to a proxy certificate

The solution is feasible due to the fact that none of the already compiled libraries and tools is changed. The changes would be made on a module, which is provided as source code, and compiled first. The adaptations are minor and all the required functionality can be easily implemented.

To conclude, the prototype implements partially the previously presented findings, but this is proven to be sufficient to check that everything works as expected. More details about the changes required by the *mod\_authn\_myproxy* module can be found in the *Appendix E*.

### Intermediary conclusions

The feasibility study regarding the user authentication shows that it is possible to provide both authentication methods required within the current project by making use of existing tools and frameworks, without needing extensive development. Because the providers currently maintain most of the needed frameworks and tools and the changes that need to be done are rather minor, the solution aims to be robust and easy to implement and maintain.

Table – Overview of the two required authentication methods

|  |  |  |
| --- | --- | --- |
| *Aspect* | *PKI* | *Username and password* |
| Possible to implement? | **Yes** | **Yes** |
| * Effort required to implement | **Easy** with Apache2  **Medium** with the other web servers | **Easy** with Apache2  **Medium** with the other web servers |
| Possible to delegate credentials? | **Yes** | **Yes** |
| * From any operating system (usability) | **No**, *htproxyput* works only on Linux | **Yes**, by making use of *Grid Proxy Manager* Firefox plugin |
| * Effort to implement | **Easy**, all the required tools already exist if Apache2 and GridSite are used.  **Hard**, in the other cases. | **Medium**, some minor changes needs to be done on *mod\_authn\_myproxy* Apache2 module  **Hard** for the other servers |
| VOMS attributes support | **Yes**, manually by the user when creates the proxy certificate | **Yes**, automatically by the *mod\_authn\_myproxy* module at login. |

As the table above shows, all the required functionality is easy to be implemented if Apache2 is chosen as the Web server. For other Web servers (NginX and Lighttpd) it is also possible, but the implementation requires much more time.

*Note: The NginX web server is modified (in the last part of the project) in order to accept the GSI proxy certificates. The changes are minor, and rather easy to do. However, the entire philosophy of this project is to not modify any standard tools or frameworks available on the Linux repositories. Thus, by using a modified NginX web server, the technologies used would not be standard anymore.*

## Caching methods

Another functional requirement of the current project is that when making use of the SUD to access a file, it needs to be stored for a period of time (cached). If the file would be accessed again, in a short period of time, it must be served to the client from the cache.

Based on this requirement, and on the fact that the communication protocol between SUD and the client is WebDAV protocol (which is extension of HTTP), the research focuses on standard methods to cache the data served by a web server.

According to the findings on the Internet, there are a large number of tools that can be used to cache the data (usually called *reverse proxy caches[[21]](#footnote-21)*). The most popular are: NginX and Apache (they are web servers and reverse proxy caches in the same time), Squid and Varnish (they are only reverse proxy caches).

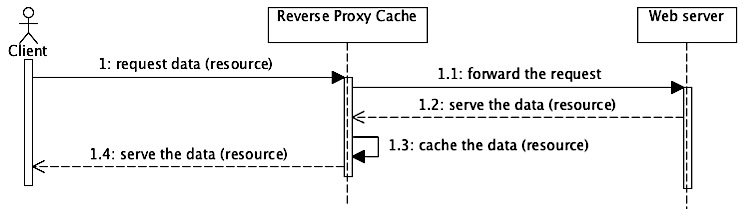


Figure – Integration between a web server and a reverse-proxy-cache

The figure above shows the main idea that stands at the base of a reverse proxy cache. However, if the reverse proxy cache would be web server as well (such as NginX and Apache2) then the two lifelines (Reverse Proxy Cache and Web server) would be combined in one.

If the WebDAV prototype described in the previous sections is reanalyzed there are mainly three actors: the client, the WebDAV Server (which is a web server) and the Storage Element.

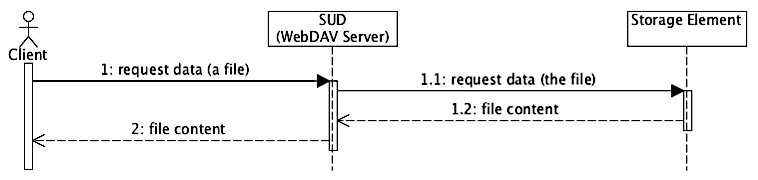


Figure – Overview of the WebDAV prototype that publishes the LFC and SE files

According to the definition of a reverse proxy cache, it is located between the client and the server, thus in this project a reverse proxy cache can stand between the client and the WebDAV server (*Approach 1*), or between the WebDAV and the Storage Element (*Approach 2*). Other approaches that use reverse proxy caches are also investigated during the feasibility study. All the approaches are presented in the following sections and their feasibility is discussed.

### Approach 1: Reverse proxy cache is between the Client and the WebDAV Server

If the reverse proxy cache is introduced within the WebDAV prototype, then the system looks similar to the model presented below.



Figure – Reverse proxy cache between the Client and the WebDAV Server architecture overview

According to the findings from the previous chapter and the initial prototype, the WebDAV server decides whether a user is allowed to access a file or not. Once the reverse proxy cache is introduced into the system, it holds a copy of the data (cache). It must then somehow decide whether it serves the file (from cache) to user that makes the request, or not. The reverse proxy cache must provide an authorization interface that can be implemented and in which the decision can be made.

After extensive search on the Internet about the investigated reverse-proxy-caches (NginX, Apache2, Squid and Varnish), it turned out that they all provide authorization interfaces as follows: NginX and Apache2 call a custom CGI application, and Squid and Varnish call an executable script. The CGI application and the executable script can be implemented from scratch with the desired authorization checks, and based on the returned value, the reverse proxy cache serves the requested file or not. The main idea is presented in the figure below.

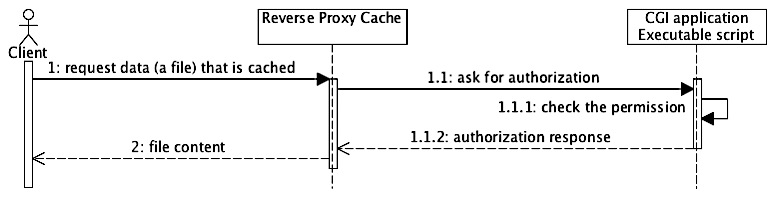


Figure – Authorization mechanism provided by reverse-proxy-caches

So far, if the reverse proxy cache is introduced into the system, it caches the data and it can provide the required security level.

As Figure 18 shows, is analyzed, the Client connects to the reverse-proxy-cache by using HTTPS (it is required in order to provide/use the PKI authentication). It means that between the Client and the reverse-proxy-cache, a session with mutual authentication needs to be established. This generates the first major issue, which is similar to the issue discovered in previous sections regarding the web servers’ support for GSI proxy certificates, but now for reverse-proxy-caches:

|  |  |
| --- | --- |
| **Problem 1 – GSI proxy certificate is not supported by any reverse-proxy-cache** | |
| Description | The Apache2, Squid, Varnish and NginX do not support by default the GSI proxy certificates (similar to the problem described in 3.5.1 - PKI authentication). |
| Possible solution | In order to make them accept the GSI proxy certificate some sort of module needs to be implemented and integrated with them. For Apache2 there is already implemented the GridSite module. For the others, with quite considerable effort this problem can be solved. |

However, the previous problem (1) is omitted and the GSI authentication is not considered a problem anymore (it can be solved with Apache2 + GridSite). If the client request a file from the reverse proxy cache, but the file is not cached, the reverse-proxy-cache forwards the request to the WebDAV server. Here, the second major problem occurs:

|  |  |
| --- | --- |
| **Problem 2 – The identity of the user is unknown to the WebDAV Server** | |
| Description | Firstly, the mutual authentication (SSL session) must be established between the client and the reverse-proxy-cache (PKI authentication). If the file is not cached, the reverse proxy cache needs to forward the request to the WebDAV server. When the request is forwarded, a new mutual authentication (SSL session) is required, now between the reverse proxy cache and the WebDAV server. In this case, the reverse proxy cache initiates the last session by making use of its own X.509 certificate (host certificate), and establishes the mutual authentication with the WebDAV server by using its own identity. The WebDAV server does not know anymore which user initiated the request. It only sees that the reverse proxy cache initiates the request (with no other details of the real user). |
| Possible solution | In order to make it work, a delegation mechanism must be somehow implemented by the reverse-proxy-caches. In the case of Squid and Varnish this is almost impossible to achieve due to the fact that this would change their entire SSL architecture[[22]](#footnote-22). With Apache2 and NginX it would probably work, but with a rather big effort. |

In other words, in accordance with *Problem 2*, the WebDAV server will never know which user initiated the request, thus it cannot identify the right proxy certificate for that particular user. This major issue cannot be (easily/at all) solved, and it proves the impossibility of using of reverse proxy caches between the client and the WebDAV server (as different machines).

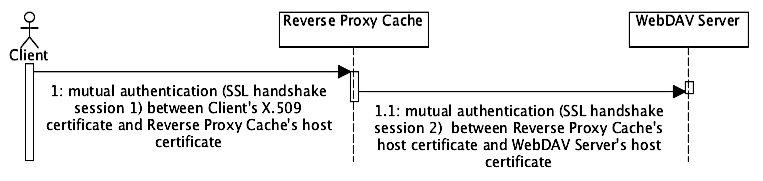


Figure – Problem description with the reverse-proxy-cache and loss of the user's identity on the WebDAV Server

Anyway, *Problem 2* can be avoided if the web server used to WebDAV implementation implements the reverse-proxy-cache functionality. The reason is that the session created between the client and the reverse-proxy-cache, can be reused internally by the web server between the reverse-proxy-cache and the WebDAV implementations (they run on the same physical machine). According to this, Squid and Varnish are eliminated from the list (they are not web servers – thus they cannot host the WebDAV implementation), and the only remaining candidates are Apache2 and NginX (due to their double functionality).

If the GSI proxy certificate authentication problem (1) is still omitted, and the last problem (2) is avoided, the current approach is feasible.

The process of caching data with web servers that provide the reverse-proxy-cache functionality is presented in the figure below. To be more clear, the Reverse Proxy Cache and the SUD (WebDAV Server) are presented as different entities, but in reality the same tool provides both of them.

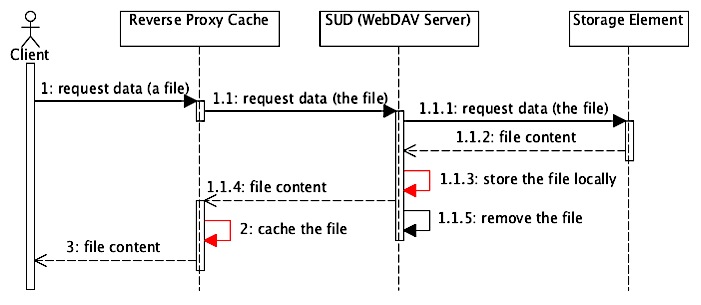


Figure – Caching files when the WebDAV server provides the reverse-proxy-cache functionality as well (red lines show where the file is duplicated)

Because the current project needs to serve files to the clients (users), it must interact with the Storage Elements, download the files, cache the files and send them to the requestor (the client). As presented in the figure above, firstly the WebDAV server needs to copy the file from the Storage Element to the local disk. Secondly, the file is read from the disk and passed through the reverse proxy cache to the client. When the file is passed through the reverse proxy cache, the reverse proxy cache saves it again, usually on the same disk where the file was downloaded initially (because the reverse proxy cache is the same machine as the WebDAV Server). After the file is served to the client (and cached) the WebDAV Server removes the initial copy.

However, this means that in order to cache a file, for a relative short period of time the file is saved twice on the same machine. This is not a major problem initially, when there are not many concurrent requests. The problem occurs when there are many concurrent requests.

|  |  |
| --- | --- |
| **Problem 3 – The cached file takes twice more space than its size** | |
| Description | When there are many simultaneous requests, the WebDAV Server (which is a reverse-proxy-cache in the same time) might run out of free space quite easily, and it will not use efficiently the available space for cache. If files are saved on the same machine twice, the performances of the system decrease, due to the high usage of the hard disks. |
| Solution | A normal and elegant solution would be to implement a HTTPS interface for the Storage Element (along with the SRM). Anyway, this solution is not feasible to be implemented during the project. Another solution would be to implement some sort of application (probably FUSE application) that can send data to the web server while it is downloaded from the Storage Element. |

Anyway, the following table summarizes the problems discovered so far.

Table – Overview of the problems encountered while using reverse-proxy-caches

|  |  |
| --- | --- |
| *#* | *Problem* |
| *Problem 1* | Reverse proxy caches do not support GSI proxy certificates by default (need changes in order to work). |
| *Problem 2* | When using reverse proxy caches as different machines, the identity of the user is lost when the request arrives on the WebDAV server. |
| *Problem 3* | When using reverse proxy caches, the files that are cached utilize twice more space than their original size (for a relative short period of time). |
| Another problem that is derived from the previous one is that the hard disk is two times more utilized (it writes two times the file), which could decrease the performances of the current project. |

The following table reviews all the reverse-proxy-caches, and the problems that are encountered if each of them is utilized (*Yes* – if used then the problem occurs; *No* – if used the problem does not occur).

Table – Overview of the reverse-proxy-caches and the problems encountered

|  |  |  |  |
| --- | --- | --- | --- |
| *Reverse proxy cache* | *Problem 1* | *Problem 2* | *Problem 3* |
| NginX | Yes (anyway, it might work if its source code is modified) | No | Yes |
| Apache2 | No (*GridSite* module solves it) | No | Yes |
| Squid | Yes | Yes | Yes |
| Varnish | Yes | Yes | Yes |

To conclude, **it is not feasible** to integrate the reverse-proxy-cache in between the client and the WebDAV server (as a different machine) mainly due to the *Problem 2*. The current approach **is feasible** only if the same web server that hosts the WebDAV implementation provides the reverse-proxy-cache functionality. Anyway, the space needed to cache a file is twice as big as the original size of the file (*Problem 3*). The easiest way to realize this approach is to use the Apache2 and the GridSite module that together solve *Problem 1* and *Problem 2*.

If Apache2 is used to host the WebDAV implementation and the reverse-proxy-cache functionality then the system scales horizontally. It is possible to multiply the number of WebDAV servers (which are also reverse proxy caches) and to increase the performances, without major issues.

### Approach 2: Reverse proxy cache is between the WebDAV Server and the Storage Element

With the new approach, the overall architecture of the prototype is presented below.



Figure – Reverse proxy cache between WebDAV Server and Storage Element architecture overview

In this case, because the client connects directly to the WebDAV server, the last will always know the identity of the user, aspect that is mandatory for the security of data.

As presented in Section 3.5, the Apache2 web server and GridSite module can be easily integrated in order to provide the required authentication methods, thus the approach uses the Apache2 web server to host the WebDAV implementation.

The reverse-proxy-cache now stands between the WebDAV Server and the Storage Element. This means that when the client requests a file, the WebDAV Server check whether the user is allowed to access the file, and afterwards it sends the request to the reverse-proxy-cache for the file. Two scenarios are possible:

* If the file is cached, then it will be served (the WebDAV Server will act as a proxy and will forward the file to the client).
* If the file is not cached by the reverse-proxy-cache, somehow the reverse-proxy-cache needs to download the file from the Storage Element. To interact with the grid it needs to have a proxy certificate, thus somehow the WebDAV Server (which has the proxy certificates of the users) needs to send a copy (or to delegate a new one) to the reverse-proxy-cache. The diagram below shows the main sequence of actions that would take place.

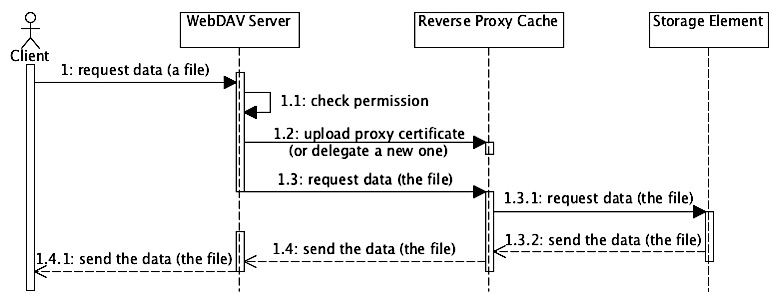


Figure – Sequence diagram of actions that are performed when the reverse proxy cache is between WebDAV server and Storage Element

An advantage (1) of this approach is that none of the investigated reverse-proxy-caches (NginX, Apache2, Squid and Varnish) would have any problems to work. The reason is that the client does not connect directly to the reverse-proxy-cache (Squid, Varnish, NginX), which does not accept the GSI proxy certificates. Instead the user connects to the WebDAV Server (Apache2 + GridSite = GSI proxy support) and the last connects to the reverse-proxy-cache by making use of its host certificate, which is a standard X.509 certificate (and is accepted by all of the reverse-proxy-caches).

*Remark*: *As Squid and Varnish are designed, they cache the data published over the HTTP or HTTPS protocols. In the current approach, since the SE does not provide any of these protocols, Squid or Varnish cannot be used directly. A web server needs to be placed between them and the SE in order to act as an adaptor between grid specific protocols (such as GSIFTP or RFIO) and HTTP (S) protocol.*

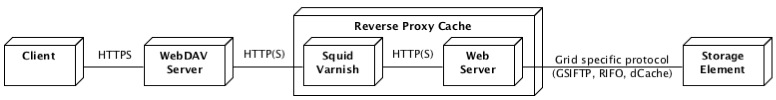


Figure – Possible usage of Squid and Varnish between the WebDAV server and Storage Element

From scalability perspective, this approach is horizontally scalable, but if the number of reverse-proxy-caches is increased the number of WebDAV Servers needs to be increased as well, due to the fact that the lasts proxy all the traffic received from the reverse-proxy-caches. If the number of WebDAV Servers would be lower, then their network connectivity would become the bottleneck of the entire system.

Table – Overview of problems, disadvantages and advantages encountered with Approach 2

|  |  |
| --- | --- |
| *#* | *Problem* |
| *Problem 1* | When using reverse proxy caches, the files that are cached utilize twice more space than their original size (for a relative short period of time). |
| Another problem that is derived from the previous one is that the hard disk is two times more utilized (it writes two times the file), which could decrease the performances of the current project. |
| *#* | *Disadvantage* |
| *Disadvantage 1* | The approach is scalable, but it is quite expensive, due to the fact that the number of reverse proxy caches must be equal with the number of WebDAV servers in order to provide the best performances. |
| *Disadvantage 2* | The solution needs to have a web server along with any of the reverse-proxy-caches, thus the design tends to be complicated. |
| *#* | *Added value* |
| *Advantage 1* | Any of the reverse proxy caches (NginX, Apache2, Squid and Varnish) can be used to implement caching, because in this approach they do not need to support GSI proxy certificates. |

To conclude, because none of the problems encountered are major, this approach is feasible to be implemented, but it is not efficiently scalable as explained previously (Disadvantage 1).

### Approach 3: Reverse proxy cache is between the Client and the Storage Element

The current approach is a combination of the previous ones, with the aim of improving scalability.

The WebDAV protocol is an extension of the HTTP. As explained in the first sections of this chapter, when a file is required the client sends to the WebDAV server (web server) a request for the GET method. The HTTP protocol specifies that based on the availability of the resources, the web server can respond to the client that the resource is located to another machine. This process is called REDIRECT, and since it is being supported by the HTTP protocol, it should be supported by the WebDAV protocol as well.

In the current approach, the WebDAV server handles all of the WebDAV specific methods (OPTIONS and PROPFIND). When GET method is requested from the WebDAV server, it redirects the client to another machine (reverse proxy cache) that handles the request for data. In this way, the main duty of the WebDAV Server is to resolve the browsing files and directories requests, and to redirect the requests for data to other machines. These machines (the reverse-proxy-caches) transfer the data directly to the client. This improves the scalability of the system due to the fact that now, the WebDAV server does not proxy any data, and the entire load is distributed across multiple reverse proxy caches of the system. If the number of reverse-proxy-caches increases then the number of WebDAV servers might not, and the efficiency would be still high.

Based on the REDIRECT capability, a new architecture is built and it is presented in the following diagram.

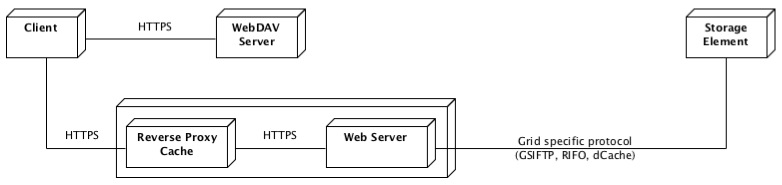


Figure – Architecture of the prototype that uses reverse-proxy-caches and REDIRECT capability

The WebDAV Server deals only with user authentication and protocol-specific requests such as directory browsing, file permissions and details about files and directories. The Reverse Proxy Cache component handles all the requests for the data by copying the required files from the Storage Element, caching locally and serving them to the client.

The current approach is analyzed in order to determine what are the technical issues that might occur. As in the first approach, the client connects directly to the reverse-proxy-cache. The problem (1) again, is that the client can connect by making use of GSI proxy certificates, which are not supported by any of the reverse-proxy-caches. Another problem (2) of this approach (which is found in all the other approaches as well) is that to cache a file, the space needed is twice the size of the file.

Anyway, according to the investigations from the *Approach 1*, the only possible easy-to-implement solution is if the Apache2 is used as reverse-proxy-cache together with the GridSite module. Therefore the *Problem 1* is considered solved.

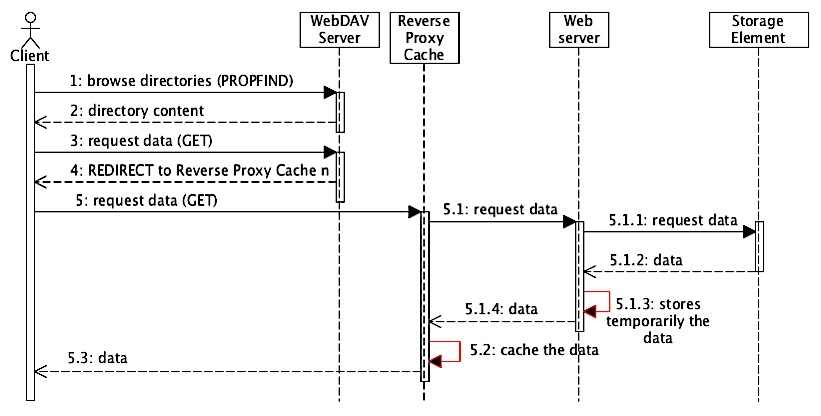


Figure – Generic sequence diagram of the Approach 3

As presented in the diagram above, now the WebDAV functionality is completely decoupled from the caching functionality, which is an important added value from the design perspective of the current project. Basically, all the reverse-proxy-caches receive only HTTP requests to serve and cache data, thus they create a real web Content Delivery Network (CDN[[23]](#footnote-23)). A disadvantage of this approach, as well as the previous one, is that along with the reverse-proxy-cache, a web server needs to be added in order to copy the data from the Storage Element and pass it through the reverse-proxy-cache. This makes the design and the implementation more complicated and sometimes less reliable (due to the multiple components that interact).

Table – Overview of problems and advantages encountered with Approach 3

|  |  |
| --- | --- |
| *#* | *Problem* |
| *Problem 1* | Reverse-proxy-caches do not support GSI proxy certificates by default (adaptations are needed in order to work). |
| *Problem 2* | When using reverse proxy caches, the files that are cached utilize twice more space than their original size (for a relative short period of time). |
| Another problem that is derived from the previous one is that the hard disk is two times more utilized (it writes two times the file), which could decrease the performances of the current project. |
| *Problem 3* | The solution needs to have a web server along with any of the reverse-proxy-caches, thus the design tends to be complicated. |
| *#* | *Added value* |
| *Advantage 1* | It is a scalable solution. To increase the performances of the system, only the number of reverse proxy caches needs to be increased (and not the number of WebDAV servers). |
| *Advantage 2* | The WebDAV implementation and the caching are decoupled, thus new protocols can be easily integrated into the system. |

To conclude, this seems a promising approach that makes the system easily scalable and in the same time does not require as many resources as in *Approach 2* (the number of WebDAV servers does not need to increase while increasing the number of reverse proxy caches). Another advantage of this approach would be that the WebDAV protocol is decoupled by the caching mechanism. This increases the extensibility of the current project, by making easily possible the addition of new protocols that can be utilized by the users to access data.

### Approach 4: The cache is managed manually

This approach uses the same motivations and ideas as the previous one (scalability and decoupling), but with the main scope to eliminate the problem identified in all the previous approaches: the space required to cache a file is double compared to the size of the file. As explained in the first approach, the problem occurs because a reverse-proxy-cache, caches only data received from a web server. The web server, in order to serve the file, needs first to copy the file from the Storage Element, and only after, the reverse-proxy-cache caches the file (stores the file in its internal format) and serves it to the client. This means that the file is stored two times: once for the web server, and once in the cache of the reverse proxy.

The current approach eliminates the reverse-proxy-caches engines from the system architecture. Thus Squid and Varnish are completely eliminated, and NginX and Apache2 are investigated only as web servers. The web server now handles the caching by making use of a manually implemented application (such as [F] CGI).

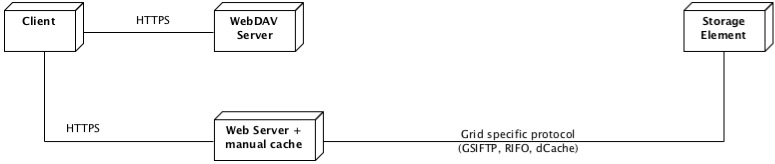


Figure – Approach 4: caching without reverse proxy caches

As presented in the diagram above, a web server that caches manually the files replaces the reverse-proxy-cache box. In other words, the difference between this approach and the previous one, is that once the file is copied from the Storage Element to the web server, it remains there as cached data. A disadvantage in using this approach is that the web server does not provide by default the caching functionality (as it would for any reverse proxy cache) and it needs to be implemented in the current project.

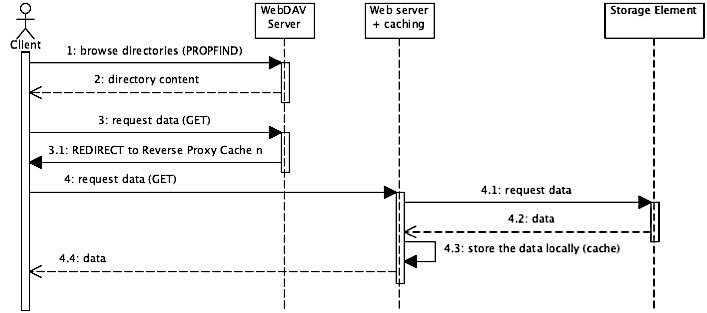


Figure – Generic sequence diagram of Approach 4

The sequence diagram above shows how data is requested, cached and served to the client by making use of the new architecture. As in the previous approach, the file is copied from the Storage Element to the web server. Once copied, unlike in the previous approaches, the web server serves the file to the client directly without being passed to any other parties. This drives to a simplification of the entire process, and it reduces the space needed to cache the file. The table below shows an overview of the problems encountered and the advantages generated by the current approach.

Table – Overview of problems and advantages encountered with Approach 4

|  |  |
| --- | --- |
| *#* | *Problem* |
| *Problem 1* | The caching policy must be implemented manually (the removal of the files once they are not used anymore) |
| *#* | *Added value* |
| *Advantage 1* | It is a scalable solution. To increase the performances of the system, only the number of reverse proxy caches needs to be increased (and not the number of WebDAV servers) |
| *Advantage 2* | The current solution has full control on the files that are cached. New caching policies can be easily implemented. |
| *Advantage 3* | The files that are cached can be grouped in any kind of hierarchy or they can even be compressed. |
| *Advantage 4* | The WebDAV implementation and the caching are decoupled, thus new protocols can be easily integrated into the system. |

To conclude, this approach is feasible and makes the system scalable (as the previous approach), but it does not need more space to cache a file than the size of that file. The only inconvenience is that the caching policy must be implemented manually, but according to some research it is not difficult to realize.

### Intermediary conclusions

It is possible to cache the data accessed by the users by using standard technologies such as web servers and reverse-proxy-servers, thus it is possible to integrate the Grid technologies with standard technologies in order to build a reliable system. However, it is not a straightforward process and some major issues are identified during this feasibility study.

Several approaches that use different reverse-proxy-cache engines (Squid, Varnish, Apache2 and NginX) are investigated and the conclusions are presented in Table 19.

* *Approach 1*: Reverse proxy cache is between the Client and the WebDAV Server
* *Approach 2*: Reverse proxy cache is between the WebDAV Server and the SE
* *Approach 3*: Reverse proxy cache is between the Client and the Storage Element
* *Approach 4*: The cache is managed manually

Table – Overview of the caching approaches

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Aspect* | *Approach 1* | *Approach 2* | | *Approach 3* | *Approach 4* |
| Feasible | Yes | | Yes | Yes | Yes |
| *NginX* as reverse proxy cache | Yes | | Yes | Yes | Yes |
| *Apache2* as reverse proxy cache | Yes | | Yes | Yes | Yes |
| *Squid* as reverse proxy cache | No | | Yes | Yes | Yes |
| *Varnish* as reverse proxy cache | No | | Yes | Yes | Yes |
| Scalability | Medium | | Low | High | High |
| Cache efficiency[[24]](#footnote-24) | No | | No | No | Yes |
| Control of the cached files[[25]](#footnote-25) | Low | | Low | Low | High |
| Implementation effort | Medium | | Medium | Medium | Medium |
| WebDAV is decoupled | No | | No | Yes | Yes |
| Recommended | No | | No | Yes | Yes |

All of the tools investigated (NginX, Apache2, Squid and Varnish) could be used to provide the required functionality. However, using the Apache2 as the web server that hosts the caching functionality is by far the easiest and the most reliable choice.

## Results & Conclusions

According to the findings revealed by the feasibility study, all the initial requirements of the current project could be implemented. Several prototypes are implemented in order to explore the issues that might occur during the development phase of the current project. The main functionality, along with the most feasible solutions, is presented in the following table.

Table – Overview of the possible solutions for each of the requirements

|  |  |  |
| --- | --- | --- |
| *Requirement* | *Solution* | *Effort to implement* |
| Interaction with the GRID | gLite framework: lfc, lcg\_util and gfal libraries | Easy |
| PKI authentication | Fully supported by Apache2 + GridSite | Easy |
| Modify or create additional module for NginX, Squid and Varnish | Hard |
| Username and password auth. | Already implemented for Apache + MyProxy | Easy |
| Manually implemented for NginX, Squid & Varnish | Hard |
| WebDAV implementation | Approach 1: WebDAV implementation provided by the Apache2, NginX and Lighttpd + Fuse | Medium |
| Approach 2: Manual implementation of the WebDAV protocol | Medium |
| Caching | Apache2+GridSite+manual cache implementation | Medium |
| NginX, Squid, Varnish+manual cache implementation | Hard |
| Apache2, NginX, Squid and Varnish as reverse-proxy-caches | Hard |

Due to the time limitation, no other possible approaches are investigated. Some decisions regarding the design and implementation of the current project, based on the feasibility study findings, are made as follows:

|  |
| --- |
| **Design decision:** WebDAV protocol is used between the SUD and the user  Motivation: The WebDAV protocol is supported by default by most of the operating systems. It covers all the functionality required by the current project, as well as it is one of the stakeholder’s preference. |

|  |
| --- |
| **Design decision:** Apache2 is used as web server for WebDAV and Caching  Motivation: The Apache2 web server can easily support all the required authentication methods (including GSI proxy certificates) by making use of some already existing additional modules (GridSite and MyProxy). |

|  |
| --- |
| **Design decision:** WebDAV protocol is implemented manually as a FCGI application  Motivation: The WebDAV protocol is implemented from scratch due to the fact that in this way, the current project would have more control on how the files are published. The custom solution would satisfy directly and more efficiently the needs of the project. |

|  |
| --- |
| **Design decision**: Caching mechanism is implemented manually as FCGI application  Motivation: By implementing manually the caching functionality actually means to copy the files from the Storage Element and store them into certain structure. Even if the effort to implement this solution is higher, the added value is more significant. |

## Implementation issues

During the feasibility study several problems regarding the Grid tools and framework are identified and presented in the following table.

Table – Main problems identified while using grid frameworks

|  |  |
| --- | --- |
| *#* | *Problem & Solution* |
| 1 | **The *htproxyput* does not delegate correctly a Limited GSI proxy certificate. It does not make difference between GSI proxy certificate and Limited GSI proxy certificate, thus it adds all the time to the DN the “CN=proxy”, instead of “CN=limited proxy”.** |
| *Solution* | The *htproxyput* needs to be modified in order to make the difference between the two types of GSI proxy certificates, and add accordingly to the DN the right type of the proxy: “CN=proxy” or “CN=limited proxy” |
| 2 | **None of the web server supports the GSI proxy certificates for mutual authentication.** |
| *Solution* | 1. Modify the web servers in order to support the GSI proxy certificates. 2. Create new modules (when possible) that will validate the GSI proxy certificates.   Apache2 already has such a module: *mod\_gridsite*. |
| 3 | **The *Grid Proxy Manager* plugin does not allow user to specify the VO (VOMS attributes).** |
| *Solution* | The certificate is uploaded to MyProxy Server without VOMS attributes. The SUD needs to add the attributes when the certificate is received from the MyProxy Server.  For Apache2: the *mod\_authn\_myproxy* module needs to be modified. |
| 4 | **The file path of the proxy certificate MUST not contain % character. Otherwise *Segmentation fault* error is generated.** |
| *Solution* | The “%” needs to be replaced with “%%” everywhere it is found in the path.  Reason: The error is generated due to the fact that the path to the proxy certificate file is passed as the format parameter of sprintf function, in the gLite middleware. |
| 5 | **The GFAL and LFC libraries cannot be used together in the same process, because *Segmentation fault* error is generated.** |
| *Solution* | Unknown |
| 6 | **The GFAL and LCG\_UTIL libraries cannot be used together in the same process, because *Segmentation fault* error is generated.** |
| *Solution* | Unknown |

Due to the lack of time, the 5th and the 6th problems identified and presented in the table above are not further investigated.

Table – Main problems identified while integrating standard and grid technologies

|  |  |
| --- | --- |
| *#* | *Problem & Solution* |
| 1 | **None of the standard tools support GSI proxy certificates** |
| *Solution* | The tools need to be modified in order to accept the GSI proxies (the problem usually originates from the *OpenSSL[[26]](#footnote-26)* library). |

Table – Main problems identified while using standard technologies

|  |  |
| --- | --- |
| *#* | *Problem & Solution* |
| 1 | **Some of the WebDAV clients does not support REDIRECT** |
| *Solution* | For Linux, there are several open source WebDAV clients that can be modified.  *Wdfs-redirect* is a modified version of *wdfs* WebDAV Client that allows REDIRECT and GSI proxy certificates as authentication credentials. |
| 2 | **The WebDAV clients do not support partial download of the files. Before reading the file, the WebDAV client copies the entire file to the local computer, and only after this allows reading.** |
| *Solution* | - |

For the full list of problems identified during the entire project, the reader must refer to the *Appendix F*.

# System Requirements

*This chapter presents the main functionality that needs to be implemented by the current project. It starts with a short introduction about the process that preceded the gathering of the requirements, and continues with presenting all the functional and non-functional requirements along with the motivation and the rationale for each of them.*

## Introduction

The main goal, as it was defined at the beginning of the project, is to offer an alternative to the current way of accessing grid data in order to make the Grid more user-friendly and easy to use by the regular users. In other words, the current project must increase the usability of the current way to access data.

To determine how user-friendliness and easiness to use must be implemented by the current project, during the first phase of the project, a survey was done across several physicists, and non-physicists users of the Grid. The survey consists in several questions focused on how the users currently use the Grid’s middleware, and how they would see the access to data more user-friendly. For the complete outcome of the survey, the reader can refer to “Survey” document [3].

According to the findings of the survey and with the input from the main stakeholder (PDP group), a set of functional and non-functional requirements is defined, and presented below. For a more complete and formal description of the requirements as they were refined through the process, the reader can refer to “Requirement Specification” document [1]. A set of use-cases is formally defined in the “Use-cases document” [11], and presented in this document as scenarios in different chapters.

## Functional Requirements

**Functional requirement 1:** The SUD must allow user to mount the files and directories defined on LFC and SE

**Rationale**: For most of the experiments the input files are processed/analyzed as remote files (which proved to be inefficient due to the high network overhead) or as *local files*. In the later case, the user first needs to copy manually the file on the Worker Node, or personal computer by making use of some Grid specific tools, and only after doing so he/she can process/analyze the file. This is not a complicated process, but it requires a special setup of the working environment (special type of Linux, gLite framework and so on). More than this, special applications need to be built, or the already existing ones need to be modified, in order to access data directly from the Grid (by using the specialized IO frameworks such as GFAL).

By mounting the Grid’s data to a machine, accessing a file would become a very easy process. It will be possible with the standard tools that the operating system provides (POSIX interface) to copy, open or list directories (cp, cat or ls) directly on the Grid’s files. This will help the user to access data more easily and without needing to install any of the Grid’s frameworks. It will be possible to access data from any operating system. A broader set of application would be able to use the data stored by the Grid, without requiring any sort of adaptations.

**Motivation:** *Usability*

**Functional requirement 2:** The SUD must support PKI and username-password authentication mechanism

**Rationale:** The security of data is a very important issue faced by the Grid community. The current architecture of the Grid uses PKI authentication method, thus to access a file, to browse a directory or to send a job the user must present his/her X.509 personal certificate. Because currently, it is the only authentication mechanism accepted by the Grid’s components and due to the fact that the users already have the X.509 personal certificates, it is desirable for the SUD to allow users to authenticate with these certificates. However, because the SUD aims to improve the usability and the ease-of-use of the Grid, it must provide an alternative authentication mechanism, such as username and password. This is desirable, due to the fact that the users might try to connect to the SUD from different machines (e.g., public computers) where they do not have the certificate.

Providing both authentication methods, the SUD will become easy-to-use from user perspective, and in the same time it will support the PKI authentication method that will be useful when the Worker Node will connect to it.

**Motivation**: *Usability* and *Security*

**Functional requirement 3**: The SUD must preserve the same authorization rules to access files as defined on the SE.

**Rationale:** Once the user is authenticated to the system, he/she can try to access data. In general, the files stored by the Grid have different permission rules (ACL). The ACL defines strictly who is allowed to access, write or execute a file, and who is not. The permissions are defined mainly in two places: in the LFC, which is the catalogue with the filenames, and in Storage Element, which actually stores the physical file. In order to keep data secure, the SUD must ensure that it serves the data only to the allowed users as defined by LFC or/and SE.

However, in reality the permission rules stored by the LFC might be different than the rules stored by the SE. Given that, because the SE stores the physical data, its permission rules will be enforced by the SUD.

**Motivation:** *Security*

**Functional requirement 4**: The SUD must store (cache) the files that are used, for an undetermined time period, as long as space permits

**Rationale:** As the Grid is a distributed system across multiple sites that can be in different countries, the data that is stored by it is distributed as well. In general, each of the Grid sites provides two different types of services: data *storage* and *computing*.

Currently, anytime a file is needed for processing, it is copied from one of the sites that store it to the site that needs it. Obviously, it is often the case that the place where data is processed is different from the place where the data is stored. When this happens, due to the limited network connectivity between the different sites, copying the file is a rather slow process. And, if this process is repeated several times, the cumulated time spent to copy the data becomes very long.

The SUD intends to be deployed close to the computing part of each site, so the network bandwidth between it and the Worker Nodes is high.

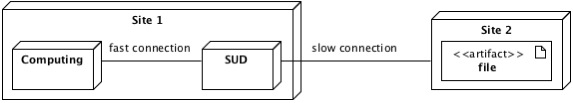


Figure – Overview of the Functional Requirement 4

Because of the previously presented reason, and due to the fact that copying the same file from a different site, several times generates a high network bandwidth usage, it is desirable for the SUD to cache the files that are used. By introducing the caching mechanism, if a file is used by several users, or by the same user several times in a relatively short period of time, then it is desirable that the file will be only once copied from the source (Site 2), and then served several times directly from the cache to the computing elements. This approach will decrease the bandwidth usage between the sites, and (preferably) will speed-up the access to data, due to the fast network connection.

**Motivation:** *Performance*

## Nonfunctional requirements

**Nonfunctional requirement 1:** The SUD must provide a uniform and transparent access to data from user perspective

**Rationale**: While using the SUD, the user must not be aware from which Storage Element the file can be copied, or by making use of what protocol it can be accessed. The user should not be aware whether the file is cached by the SUD or not. The user will only see the files and directories as defined by the LFCs and SEs, and he/she will be able to access any of the files (if permission rules allow this).

**Motivation**: *Usability*

**Nonfunctional requirement 2**: The SUD should use as much as possible standard technologies

**Rationale**: Most of the technologies that will be utilized to implement the SUD must be as standard and widely used as possible. This is needed to ensure a lower maintenance cost. Using standard tools and frameworks that are employed by large community is a guarantee that they are robust and well tested. In addition, usually a good documentation and support is available for these tools. Another reason to use standard technologies is that they evolve in time (e.g., new functionality is added, performance is improved), fact that will usually affect positively the SUD.

**Motivation**: *Maintainability*

**Nonfunctional requirement 3**: The SUD should be scalable and additional resources can be easily added

**Rationale**: When the number of jobs that use the current project increases, only one machine will not be able to handle the entire amount of data that is required. To cope with this problem, the SUD must be scalable in the sense that additional resources can be added to the system in order to increase the performances.

**Motivation**: *Scalability*

**Nonfunctional requirement 4:** The changes required on the already existing Grid components must be minimum

**Rationale**: Because the Grid is a widely distributed system, it becomes very difficult to apply changes on its resources. The usage of SUD should not require any changes on the current production environment of the Grid.

**Motivation**: *Usability*

**Nonfunctional requirement 5**: The number of tools that the user needs to install in order to connect to the SUD should be 0

**Rationale**: Because the SUD tries to be user-friendly, it is not desirable to force the user to install different tools. More than that, if the user is not allowed to install these tools, then he/she will not be able to use the SUD.

**Motivation**: *Usability*

**Nonfunctional requirement 6:** Failure rate of jobs that use the SUD, caused exclusively by failures of the SUD, should be less than 2%

**Rationale:** The jobs can fail for various reasons. For example, during the high loads of the SUD, job failure can occur due to hardware limitations of the machines where the SUD is deployed. However, if the ratio between the WNs and disk servers and the ratio between WNs and SUD resources are equal, then the rate of failure caused exclusively by the SUD must not exceed 2%.

**Motivation**: *Robustness*

**Nonfunctional requirement 7:** The total execution time of a job that uses SUD should be less than the current system

**Rationale**: Because of the large amount of data that is processed by the Grid, it is not desirable for the jobs that use the SUD to spend more time than they currently spend to access data. The SUD must provide a user-friendly access to the data, but in the same time it must be in general faster than the current system.

**Motivation**: *Performance*

# System Architecture

*The chapter provides a comprehensive architectural overview of the current project. It presents a number of different architectural aspects of the system, defines the main components and the interaction between them by making use of several scenarios. It is intended to capture and convey the significant architectural decisions, which have been made on the system.*

## Introduction

According to the requirements defined in the *Chapter 4*, to the survey results [3] and the feasibility study presented in *Chapter 3*, it was decided to use the WebDAV protocol as the main communication protocol between the SUD and the users. The decision that was made has a strong support proved by the prototypes that were implemented during the feasibility analysis, but it was also influenced by the limited amount of time available for the project.

The following sections present a light overview of the SUD, and in particular its division in layers and the main components that are build in order to provide the required functionality.

### *Architecturally Design Layers*

The architecture of the current project has been designed using the layered architectural pattern. A two dimensional layering approach has been used to structure the software firstly by responsibility and secondly for reuse.

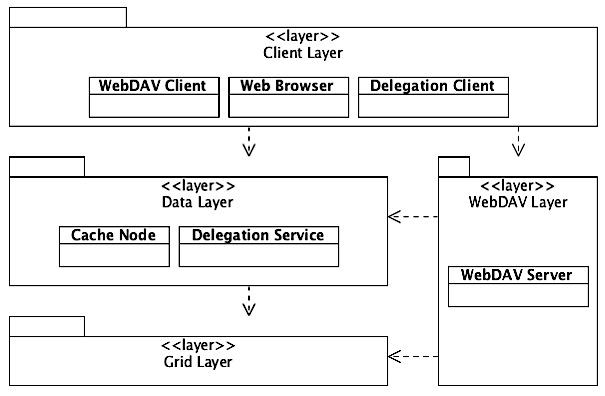


Figure – Architecturally Design Layers Overview

As shown in the figure above, the entire functionality is divided in four main layers. The first dimension (responsibility) consists of Client, Data and Grid layer. All together provide the entire functionality needed to allow users to access data (e.g. throughout HTTPS). The second dimension (reusability) involves the WebDAV layer that uses independently most of the other layers in order to provide additional functionality.

The main idea that stands at the base of the entire architecture is to provide a scalable solution, with well-defined layers that can be easily extended. In order to cope with the risk of choosing the wrong technology, and due to the fact that the SUD aims to be extensible, the WebDAV support is implemented and integrated in the entire system in such a way that it can be easily replaced or modified without impacting the design of the project. By using this approach, any other transfer protocol that would be required to access data will be easy to integrate and implement in the current architecture. In the following sections each of the layers are briefly described, together with any significant components and their key responsibilities.

**Grid Layer** represents all the components of Grid that interact with the SUD in order to provide the desired functionality. According to the requirements and to the research conducted during the feasibility study, the current project must interact with only three main components provided by the current Grid infrastructure. These components are the two file catalogs that must be published by the SUD (LCG File Catalog and Storage Element), and the MyProxy Server that provides support for username and password user authentication (address *Chapter 2* for more details).

**Data Layer** represents the main part of SUD that interacts with the Grid's components for data, manages and caches files. It implements a distributed system, which is the base of the current project. The layer provides the first essential components for implementing an eventual Content Delivery Network[[27]](#footnote-27) (CDN). This layer consists of two components: *Cache Node* and *Delegation Service*.

**WebDAV Layer** represents the adaptor that allows end-users to connect to the *Data* layer by using the WebDAV protocol. It is an addition to the *Data* layer that publishes data over the HTTP (S) protocol. It is composed by a single main component (the WebDAV Server), which handles all the WebDAV-specific requests, and generates the right responses in accordance with the protocol specifications.

**Client Layer** represents all the end-users and tools needed to connect and request data from the SUD. The end-users might be the human users, Worker Nodes or any other third parties that need data from the Grid. The client by default connects to the Cache Node by making use of the HTTPS. Because on top of the HTTPS protocol the SUD provides WebDAV, the user can connect to the SUD also by making use of a WebDAV Client.

## System description

The figure below shows the main connections between different components described in the previous section, as well as the interfaces/protocols that are used.

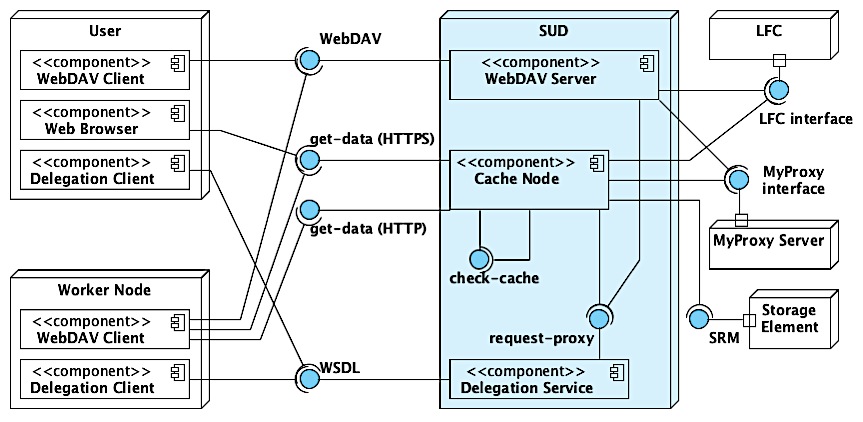


Figure – General components interaction overview

The User-box usually represents any human users that connect to the SUD by making use of a web browser or a WebDAV client. The Worker Node-box represents the Worker Nodes that compose the grid, and that connect to the SUD by using a WebDAV client. The User and Worker Node boxes compose the Client Layer. The SUD-box represents all the modules that compose the SUD and that need to be implemented by the current project. The LFC, Storage Element and MyProxy Server are part of the Grid Layer, they are already implemented, and thus the SUD only needs to use them. All components mapped onto the Figure 31 that belong to the SUD are presented in the next table, and their main responsibilities, together with their corresponding interfaces (provided and required) are presented in Table 24.

Table – Description of the main components of the SUD

|  |  |
| --- | --- |
| *Component* | *Description* |
| **Delegation Service** | Functionality:   * Creates and stores proxy certificates. * Provides proxy certificates to the SUD components.   Provided interfaces:   * WDSL interface: users connect and delegate their credentials. * Request-proxy interface: SUD components connect and retrieve a proxy certificate. |
| **WebDAV Server** | Functionality:   * Provides WebDAV protocol and publishes files defined by LFC and SE. * Authenticates the user. * Checks user’s permissions to browse a directory. * Handles only PROPFIND and OPTIONS requests. * Proxies the GET requests to the Cache Nodes, in case the client is a web browser. * Proxies the GET requests to the Cache Nodes, in case the WebDAV client does not support REDIRECT. * Redirects the user’s GET requests to the Cache Nodes for data, in case the client supports REDIRECT.   Provided interface:   * *WebDAV interface*: users connect by making use of WebDAV protocol.   Required interfaces:   * Request-proxy: retrieves a proxy certificate from the Delegation service. * MyProxy interface: retrieves a proxy certificate form MyProxy Server. * LFC interface: connects to the LFC for files. |
| **Cache Node** | Functionality:   * Downloads and caches the required files from the SE. * Authenticates the user. * Checks the user’s permissions to access the required files. * Handles only GET requests. * Generates the HTML views for users that connect to SUD by making use of web browser. * Communicates with other Cache Node instances in order to check whether the file is already cached. * Flushes the cache when full, according to the cache policy.   Provided interface:   * *Get-data (HTTPS)*: main interface that allows users to connect and require files. The files transfer is encrypted. The interface might redirect the user to the Get-data (HTTP) interface, for unencrypted transfer. * *Get-data (HTTP)*: allows users to transfer the files unencrypted (faster transfer). Through this interface the user cannot require a file. First the user needs to use the Get-data (HTTPS) interface to authenticate. * *Check-cache*: allows other Cache Node instances to connect and to check whether the file is cached locally.   Required interfaces:   * *Request-proxy*: retrieves a proxy certificate from the Delegation service. * *MyProxy interface*: retrieves a proxy certificate form MyProxy Server. * *LFC interface*: connects to the LFC for files. * *SRM*: connects to the Storage Elements to copy data. |

## System overview

The overview of the current project (see Figure 32) shows the main ideas about how the product must look from architectural point of view.

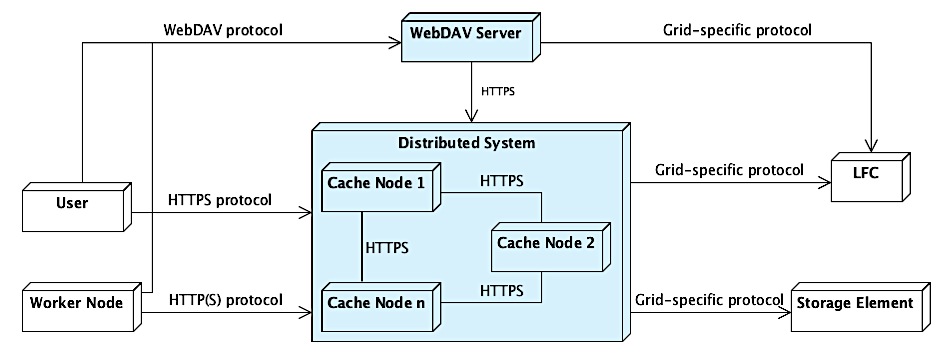


Figure – Architectural overview of the System Under Development (SUD)

As presented previously, the main part of the current project is a Content Delivery Network (CDN) composed by multiple Cache Nodes (Data Layer). The CDN can solve any HTTP (S) requests for data. On top of it, for convenience, the WebDAV protocol is introduced to the system.

## Data flow and scenarios

### Credential delegation

As required, the current project must allow users to authenticate by making use of their X.509 [GSI][proxy] certificate, or a username and password pair. According to the feasibility study this is possible, and the username and password authentication can be achieved by integrating the MyProxy Server into the system. There are two possible scenarios that permit credential delegation for both authentication methods.

1. When the users authenticate to the SUD by using the X.509 certificate, they must first delegate their credentials to the Delegation Service. Any component of the SUD, where the users authenticate, interrogates the Delegation Service for the proxy certificate.

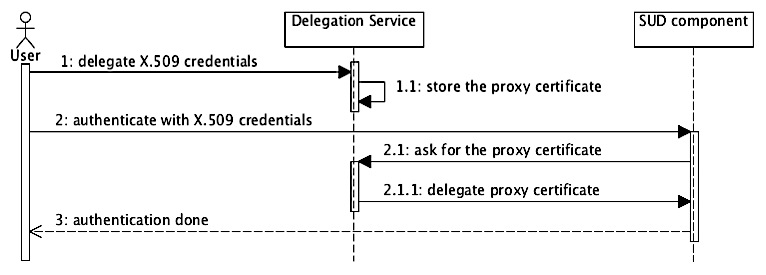


Figure – Interaction between the User, Delegation Service and other components of the SUD (such as Cache Node, or WebDAV Server)

1. When the users use the username and password for authentication, they need first to delegate their credentials to the MyProxy Server, and afterwards to define a username and password pair. The components of the SUD, where the users authenticate, check the username and password against the MyProxy Server. Once they are valid, MyProxy delegates the users credentials to the SUD component.

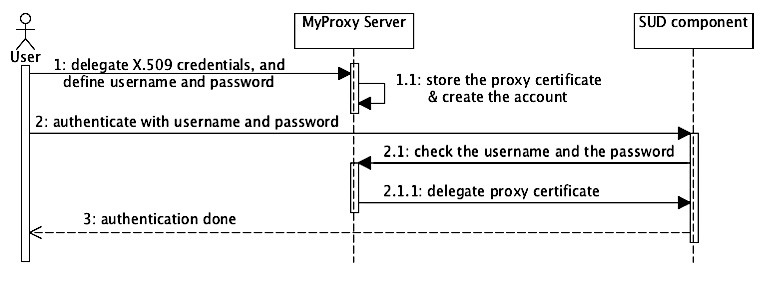


Figure – Interaction between user, MyProxy Server and other components of the SUD (such as Cache Node, or WebDAV Server)

Note: *In the following diagrams of this chapter, the authentication might be or might not be explicitly presented in the diagrams. However, any interaction between a client (e.g. WebDAV Client or Web Browser) and any component of the SUD require authentication, and it is implicit.*

### Interaction between WebDAV Client and SUD

As presented in the *Feasibility Study*, a WebDAV client might (or might not) support REDIRECT functionality. For this reason, the architecture of the current project must account for both types of clients.

1. WebDAV client supports REDIRECT: If the request received by the WebDAV Server is not for directory browsing (PROPFIND), but instead it is for data (GET) then the WebDAV Server sends a message to the client in which it specifies the location of the Cache Nodes from where the data can be retrieved. In this case, the WebDAV Server will not handle the data at all; instead the client copies the data directly from one of the Cache Nodes.
2. WebDAV client does not support REDIRECT: When the request received by the WebDAV Server is for data (GET) then it forwards (proxies) the request to one of the Cache Nodes in order to transfer data. In this case, the WebDAV Server needs to proxy the data from the Cache Nodes to the client.

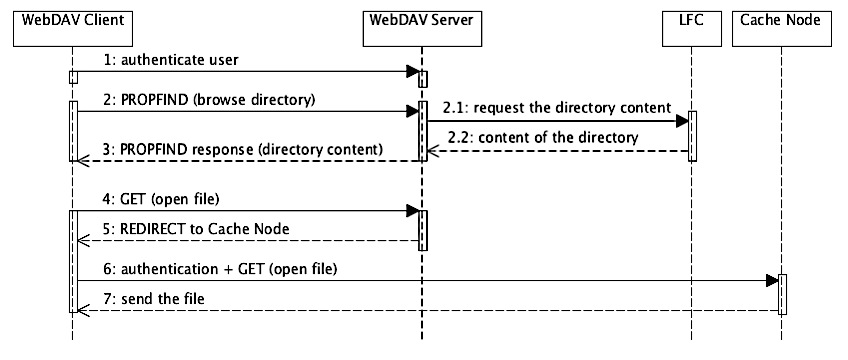


Figure – Interaction between a WebDAV client (supports REDIRECT) and SUD

Figure 35 shows the communication between various components in order to browse a directory and to read a file by using a WebDAV client that supports REDIRECT.

### Interaction between Web Browser and SUD

The users can utilize a web browser in order to connect to the SUD, thus they can connect directly to one of the Cache Nodes (which provide HTTPS interface). However, the users can connect with the web browser directly to the WebDAV Server. In this case, the server needs to proxy the requests to one of the Cache Nodes. The process is described in the figure below.

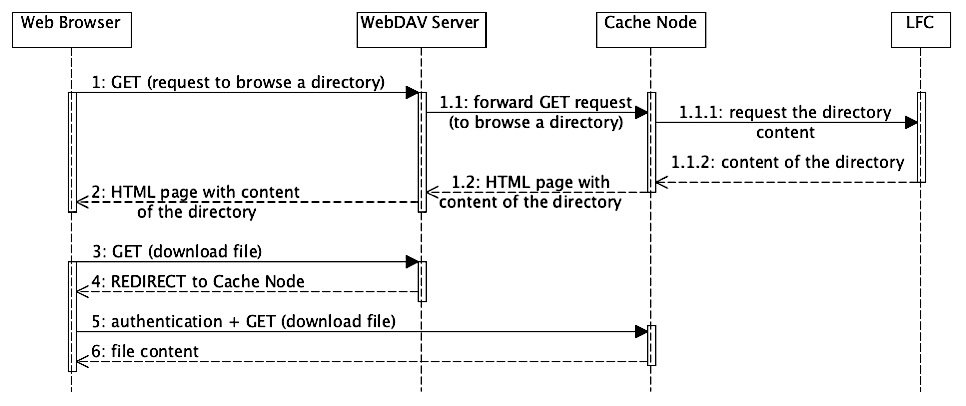


Figure – Example of interaction between a web browser and the SUD

The 5.5.2 and 5.5.3 sections show very clearly the distinction between the WebDAV Server and Cache Node responsibilities:

* WebDAV Server handles only the PROPFIND (and OPTIONS) requests received from a WebDAV client. The GET requests received from a WebDAV client or Web Browser are redirected or forwarded to the Cache Nodes.
* The Cache Node handles only the GET requests for data, and GET requests received form a Web Browser for browsing directories.

In other words, the Cache Nodes handle all the HTTP requests and the WebDAV Server handles only the WebDAV-specific requests.

### Interaction with Cache Nodes

Any HTTP [S] client can connect to the Cache Nodes in order to download data, or to browse the directories as HTML pages. The current project aims to be highly scalable and multiple Cache Node instances can be deployed in the same time. This means that the instances need to communicate with each other in order to check whether a file that is requested is cached to any of them. The process is explained in the next two figures.

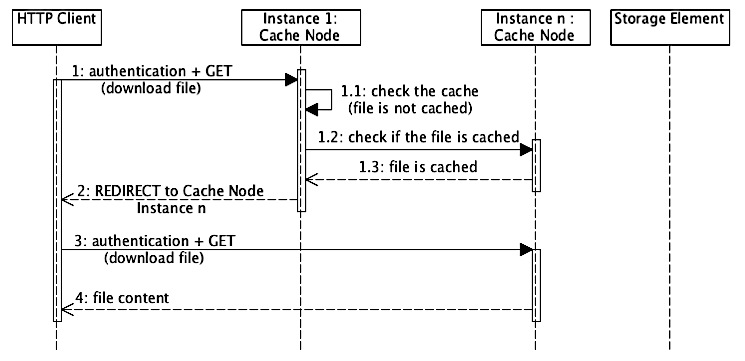


Figure – Interaction between client and Cache Nodes: the file is cached but not on the Cache Node where the request was sent initially; the client is redirected to the Cache Node that holds the file

In Figure 37 the request goes initially to a random instance of the Cache Node. The file is not cached locally and the instance queries the other Cache Nodes for the required file. If the file is already cached by any of the other Cache Node instances, then the client is redirected to that particular Cache Node.

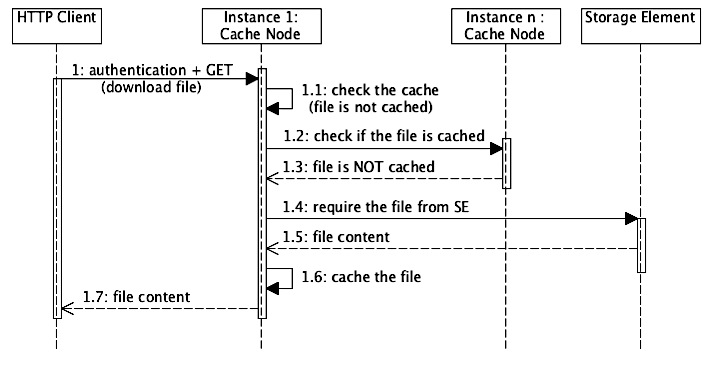


Figure – Interaction between client and Cache Nodes: the file is not cached by any of the Cache Nodes; it is copied from the Storage Element

In the figure above, the request is received by a Cache Node instance. The requested file is not locally cached, thus the Cache Node queries the other instances. If the file is not already cached by none of the instances, then the Cache Node instance that received initially the request copy the file from the Storage Element.

## Decisions

The architecture presented in the current chapter aims to be highly scalable, and easily extensible. Adding more Cache Node instances to the system makes the system scalable. Adding other protocol than WebDAV is possible, due to the fact that the current Web Content Delivery Network (build from several Cache Nodes) does not depend at all on WebDAV Server. Thus other protocol can be implemented and it can easily make use of the data that is provided by the CDN.

However, some decisions can be made in order to continue with the design.

|  |
| --- |
| **Design decision:** The web server that hosts the WebDAV implementation dispatches (or proxies) the multitude of requests for data across the Cache Nodes.   * When the client that initiated the request has the REDIRECT capability, the web server acts as a Dispatched (or Redirector) and it redirects all the requests for data randomly to one of the Cache Nodes. * When the client that initiated the request does not support REDIRECT then the web server acts as a load balancer by proxying the requests randomly to one of the Cache Nodes and handling the entire amount of data that is transferred.   The clients that support REDIRECT are defined manually in the configuration file of the web server. From now on the dispatching and load-balancing module is generically called Load Balancer.  Motivation: The web server (Apache2) already provides a load-balancing mechanism. The most important advantage of it is that it is easy to configure without modifying the SUD. Only an external text file that resides on the computer where the web server is installed defines all the Cache Nodes where the requests are redirected. |

|  |
| --- |
| **Design decision:** The wdfs WebDAV client for Linux must be modified in order to accept all authentication methods and to support REDIRECT.  Motivation: As presented before, the REDIRECT capability of the WebDAV clients stands at the base of the entire design. Thus, because the Worker Nodes are Linux machines and there is no WebDAV client that accepts REDIRECT and GSI proxy certificates for them, it is essential to provide one. |

|  |
| --- |
| **Design decision:** The Cache Node can send data unencrypted to the client if required (by default, the data is encrypted).  Motivation: The overhead introduced while data is encrypted is very high. Because several internal networks, which are considered secure, connect the SUD and the Worker Nodes it would be desirable to transfer the data unencrypted to increase the performances. |

# System Design

*The chapter provides a comprehensive overview of the design of the current project. In order to depict the software as accurately as possible, the structure of this chapter is based on the “4+1” model view of architecture. The design decisions that are made and their rationale are also presented within the next sections.*

## Introduction

The “4+1” view model consists of logical view, process view, implementation view, deployment view and use-case view. The logical view shows how the system is structured and organized. It addresses the functional requirements of the system, provides an abstraction of the design model and defines main design subsystems and classes. The process view captures the concurrency and synchronization aspects of various components of the design. The implementation view describes the static organization of software. It is omitted from this chapter due to the fact that most of the aspects are discussed in the feasibility study and in the logical view. The deployment view describes the distribution of software components across the hardware. The use-case view is left out from this section, because a set of scenarios is defined in the previous chapter.

## Logical View

The logical view of the system is divided into the *static logical view* and the *dynamic logical view.* The static logical model describes the classes and their relationships to each other through class diagrams. The dynamic logical view describes the system dynamic behavior through sequence diagrams.

### Static Logical View

The design of the current project has been divided in several main packages in order to structure the software firstly by responsibility and secondly for reuse. In the following sections each of the high-level packages are briefly described, along with any significant classes or components, their key responsibilities and the name that will be given to the package when it is implemented.

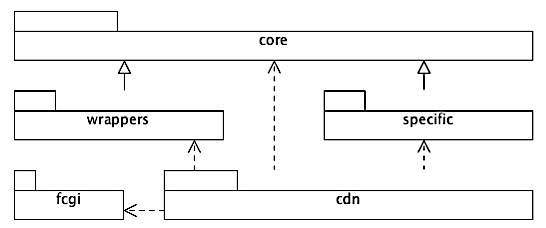


Figure – Architecturally Significant Design Packages

The main packages are summarized below:

* ***core*** package represents the base of the SUD in terms of extensibility and reusability. It consists mainly of a set of interfaces and abstract classes that are implemented in the other packages in order to provide the desired functionality.
* ***wrappers*** package is composed by several classes that implement interfaces from the core package. These classes perform the connection between the SUD and the Grid components.
* ***specific*** package implements different classes from core package in order to provide the required functionality such as user interaction with the SUD or WebDAV protocol.
* ***fcgi*** package represents several classes required to use the FCGI framework.
* ***cdn*** package represents various subsystems of the SUD that compose the main component presented in the System Architecture chapter.

***core* package**

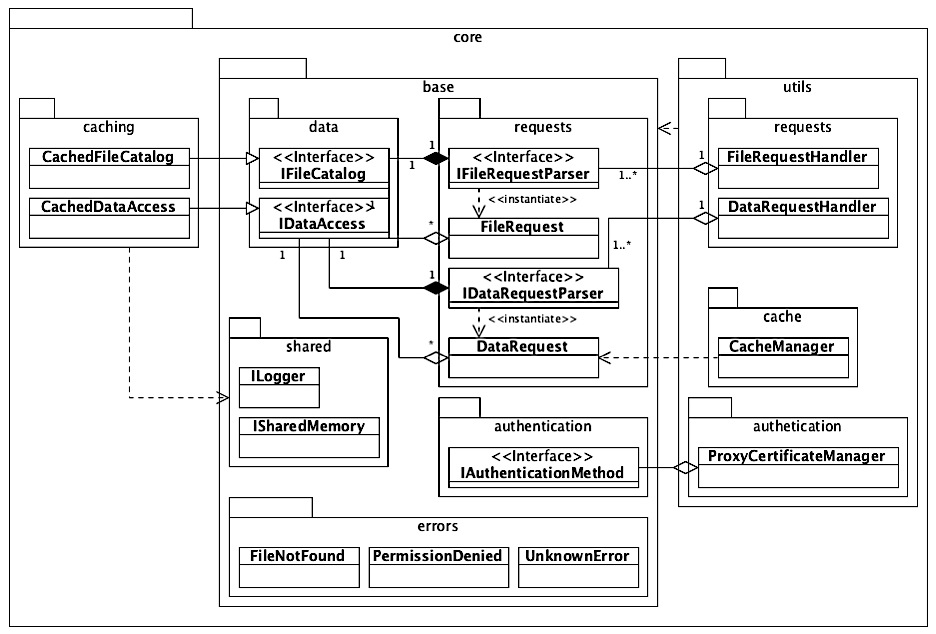


Figure – Overview of the classes and interfaces that compose the *core* package

The most important sub-packages that compose the *core* package are:

* ***core.base*** contains a set of classes and interfaces that are used by most of the SUD components. It consists in several sub-packages such as error types, logging and interaction with file catalogs and data providers. It defines all the extension points that can be exploited to enrich the SUD features.
* ***core.caching*** contains only two classes that implement, for convenience, a short-time cache mechanism for the file catalogs and data providers. They are used by the cdn package.
* ***core.utils*** contains a set of “commodity” classes that manages the *core.base* classes' implementation.
* ***core.shared*** contains two interfaces: *ILogger* and *ISharedMemory*. ILogger needs to be implemented and it receives log messages from most of the components. *ISharedMemory* is used mainly by the classes contained in the *core.caching* package, and provides access to a shared memory (from now on called memory-cache).

The most important interfaces provided by the *core* package are: IFileCatalog, IDataAccess, IAuthenticationMethod, IFileRequestParser and IDataRequestParser.

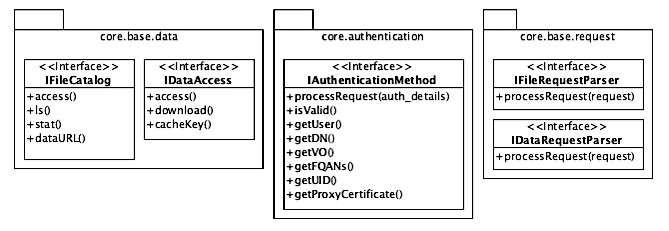


Figure – Description of the main interfaces provided by the *core* package

* *IFileCatalog* is the interface between the SUD and any external file catalog such as LCG File Catalog (LFC) or the Storage Element (SE). Any files and directories published by the SUD are retrieved through this interface. The interface does not handle the data, but only the files definition (metadata).
* *IDataAccess* is the interface between the SUD and any external data repository such as Storage Elements. It copies files from the data repository to the SUD.
* *IAuthenticationMethod* is the interface used by the SUD to authenticate the user and to retrieve a valid proxy certificate.
* *IFileRequestParser* is the interface used by SUD to identify the type of the request received from the user. It might be for example a SURL or a LFN. The interface maps the request to the file catalog for which the request is meant.
* *IDataRequestHandler* is the interface used by the SUD to identify the *IDataAccess* interface through which the file (the content) requested by the user can be accessed.

The most important classes provided by the *core* package are: *FileRequestHandler*, *DataRequestHandler*, *CacheManager* and *ProxyCertificateManager*.

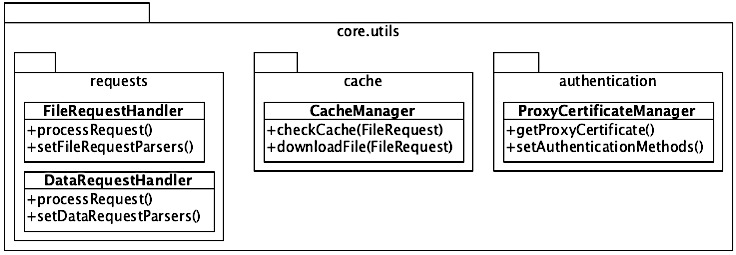


Figure – Important classes provided by the *core* package

* *FileRequestHandler* is a class that manages multiple *IFileRequestParser* instances. The *processRequest* method returns a list of *FileRequest* instances that bind the initial request to the corresponding *IFileCatalog* instance by calling the processRequest method of all the IFileRequestParser instances that are connected to the FileRequestHandler.
* *DataRequestHandler* is a class that manages multiple IDataRequestParser instances. The workflow is similar with *FileRequestHandler.*
* *ProxyCerficficateManager* makes available a proxy certificate for the user that initiated the request, by querying *processRequest* method of any *IAuthenticationMethod* implementations (e.g. GridSiteAuthentication and MyProxyAuthentication).
* *CacheManager* is a very important class of the current design, and it is in charge of organizing data that is cached by the current project. It provides two methods that ignite the process to cache a file and check whether a file is already cached.

***wrappers* package**

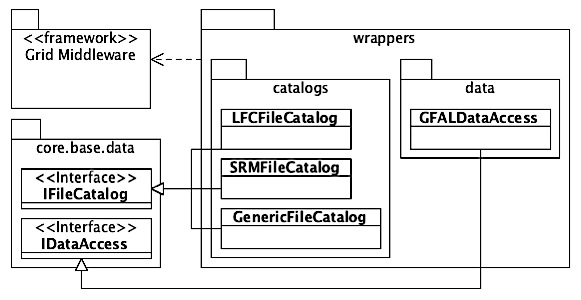


Figure – Overview of the classes that compose the *wrappers* package

The classes defined by this package represent the bindings of the SUD to the Grid’s components.

* *LFCFileCatalog* implements the IFileCatalog interface and connects to the LCG File Catalogs and retrieves information about files and directories.
* *SRMFileCatalog* implements the IFileCatalog interface and connects to the SRM interface of Storage Elements and retrieves information about files and directories.
* *GenericFileCatalog* implements the IFileCatalog interface and is in charge of connecting to the BDII server and retrieving the Storage Elements and the LCG File Catalogs available for the current user based on his/her VO.
* *GFALDataAccess* implements the IDataAccess interface and connects to the Storage Elements to access (retrieve) data, by making use of the GFAL protocol (SRM interface).

***specific* package**

*Specific* package consists in a set of classes and sub-packages that implement several interfaces defined in the *base* package. The main role of these classes is to implement how the SUD is used from different perspectives such as the request format, authentication method, or the response received by the user from the SUD (HTML, WebDAV). The implementation of these classes is specific to the technologies that are used to realize the SUD.

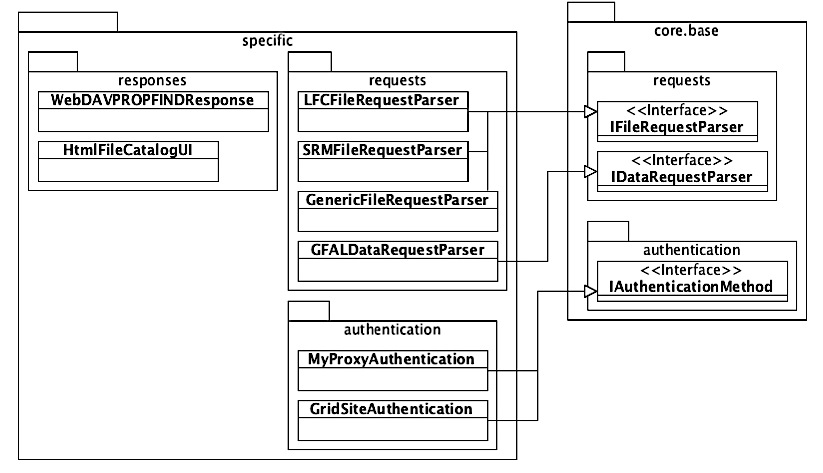


Figure – Overview of the classes that compose the *specific* package

*core.base.authentication.IAuthenticationMethod* implementations:

* *MyProxyAuthentication* analyzes the system environment variables that are updated by the MyProxy framework and identifies the user’s proxy certificate.
* *GridSiteAuthentication* analyzes the system environment variables that are updated by the GridSite framework and makes sure that the user’s proxy certificate is on the local system.

*core.base.request.IFileRequestParser* implementations:

* *LFCFileRequestParser* defines the rules that are used to determine whether a user-request represents a file or directory defined on the LFC.
* *SRMFileRequestParser* defines the rules that are used to determine whether a user-request represents a file or directory defined on the SE.
* *GenericFileRequestParser* defines the rules that are used to determine whether the user wants to see the list of LFCs or SEs that can be browsed. It also defines rules that transform the generic user-request and then it passes the modified request to the others file request parsers (*LFCFileRequestParser* and *SRMFileRequestParser*).

*core.base.request.IDataRequestParser* implementations:

* *GFALDataRequestParser* defines the rules that are used to determine whether the data required by the user can be accessed with the GFAL framework.
* *WebDAVPROPFINDResponse* generates the WebDAV XML response for a list of files defined by the *IFileCatalog* implementations. The output is conforming to the WebDAV protocol specification (RFC 4918).
* *HTMLFileCatalogUI* generates the HTML view of the files and folders defined by the *IFileCatalog* implementations.

***fcgi* package**

Is composed by all the classes and frameworks required to use the FCGI technology.

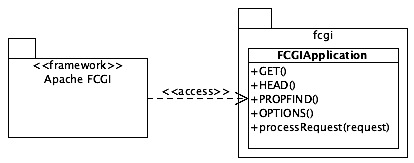


Figure – Overview of the classes that compose the *fcgi* package

*FCGIApplication* is the skeleton of an FCGI application. It is an abstract class where the various HTTP methods that are important must be implemented. Once the request is received, the required method is identified and one of the GET, HEAD, PROPFIND and OPTIONS methods of *FCGIApplication* is called accordingly.

***cdn* package**

The *cdn* package contains the classes that combine all the functionalities offered by the other packages (specific, wrappers and fcgi) in order to produce the desired behavior. It is divided in two sub-packages that separate the WebDAV and caching functionalities and two extra that provide (1) two specializations of the *FCGIApplication* class, and (2) the communication interfaces between the SUD components.

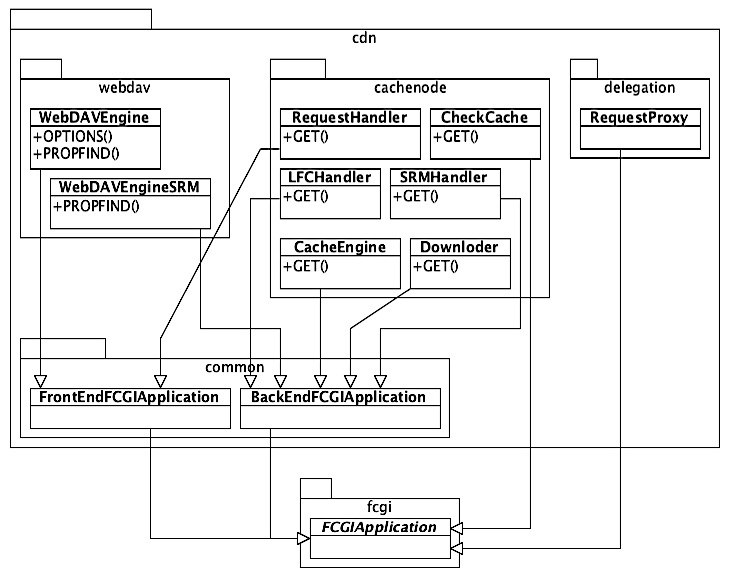


Figure – Overview of the classes that compose the *cdn* package

* *FrontEndFCGIApplication* represents the web applications that are publically available and where the user connects directly. The main functionality provided is the user authentication. It makes sure that the X.509 proxy certificate of the user is on the machine where the web server runs.
* *BackEndFCGIApplication* represents the applications that are not publically available, thus the user cannot access them directly. They are called only by the front-end application after an internal redirect. The identity of the user is received from *FrontEndFCGIApplication* as part of the internal redirect request.

|  |
| --- |
| **Design decision:** The interaction between SUD, and LFC or SE is implemented in different processes due to the *Segmentation Fault* error generated by the usage of the lcg\_util, lfc and gfal frameworks (as described in the *Feasibility Study)*. |

**WebDAV Server implementation**

* *WebDAVEngine* authenticates the user; handles all the file requests for LFC.
* *WebDAVEngineSRM* handles all the requests for files defined on the SE.

|  |
| --- |
| **Design decision:** Serving data to the client, and copying data from the SE to the Cache Node is implemented as two distinct processes. The reason is that both operations are very resource consuming, and usually they require large amounts of time to complete. Moreover, the two operations are very different and they can easily work in parallel (a file can be served to the client in the same time as another file can be copied from the SE to the Cache Node). |

**Cache Node implementation**

* *RequestHandler* authenticates the user; identifies the type of request (LFC or SE).
* *LFCHandler* generates the HTML view if the request points to a directory; retrieves the SURL from the LFN.
* *SRMHandler* generates the HTML view if the request points to a directory.
* *CacheEngine* checks the local cachefor the requested file; queries other Cache Node instances for requested file; and serves the file to the client.
* *Downloader* downloads the file requested by the user from the Storage Element, and serves the file to the client.
* *CheckCache* is an interface that can be called only by other Cache Node instances to check whether a file is cached locally or not.

**Delegation Service implementation**

* *RequestProxy* is an interface called only by the SUD components (WebDAV Server or Cache Nodes). It delegates a proxy certificate (of a user specified within the request) to the component that initiated the request.

The *WebDAVEngine* module is presented in the following figure. For a detailed description of the other modules, the reader must refer to the Software Architecture Document [3].

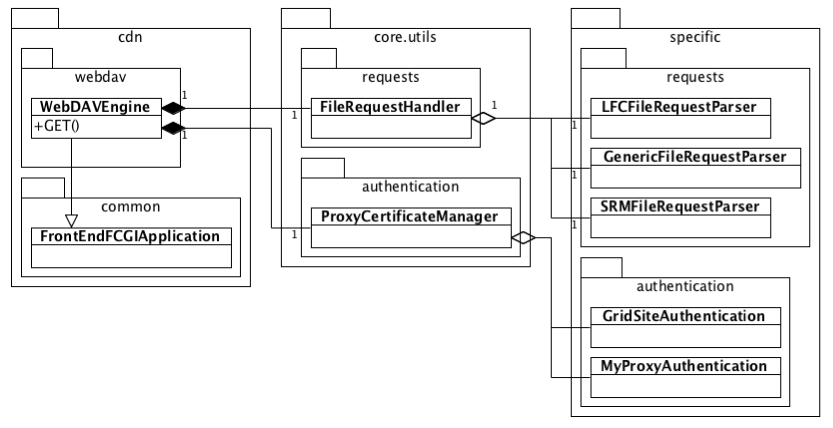


Figure – *WebDAVEngine* detailed class diagram

### Dynamic Logical View

Figure 48 shows the main operations flow of any *FrontEndFCGIApplication* instances. Once the server is started, it initiates all the FCGI applications.

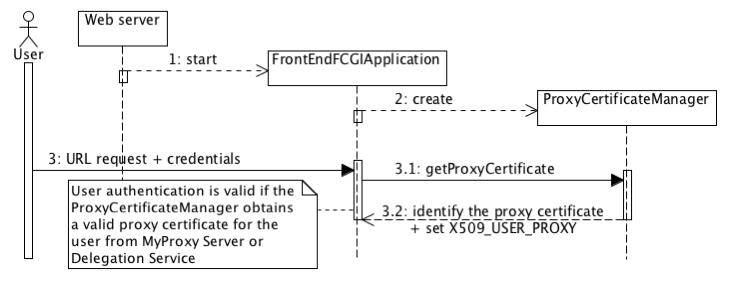


Figure – Operations that take place when a *FrontEndFCGIApplication* is started

When a *FrontEndFCGIApplication* is initiated, the authentication module is initiated automatically. Any request of the user is processed by the application only after a valid authentication.

Figure 49 presents the dynamic logical view of the *WebDAVEngine* module of the WebDAV Server. The other modules that compose the SUD follow a similar pattern.

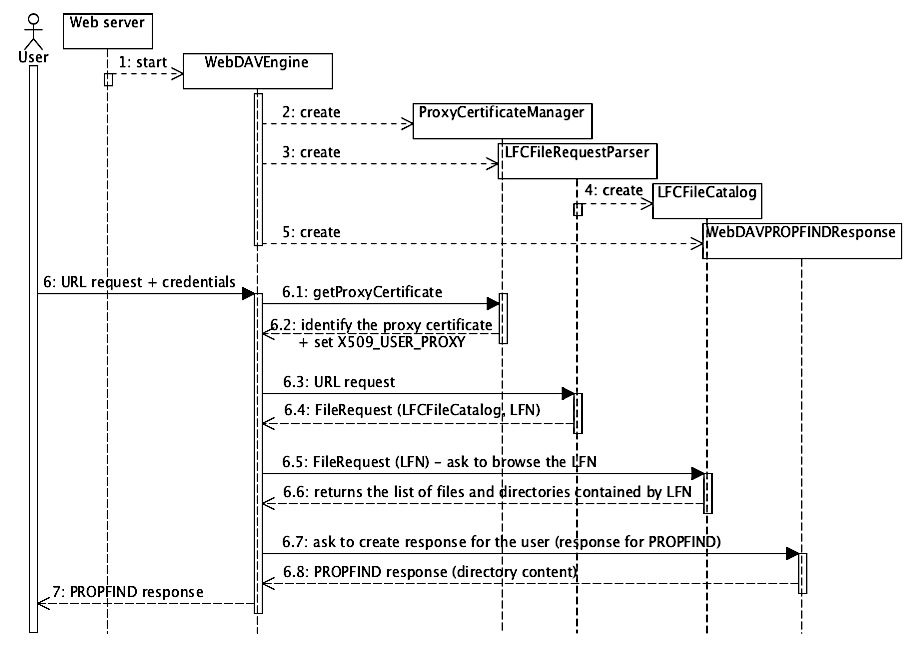


Figure – Overview of the operations that take place when WebDAVEngine receives a request

As presented in the figure above, initially when the web server starts the *WebDAVEngine* application (which extends the *FCGIApplication* class) is initiated. When WebDAVEngine application is started a set of classes defined in the *specific* package are instantiated: the *ProxyCertificateManager* provides the authentication; the *FileRequestParser* verifies whether the request is for LFC files; *LFCFileCatalog* represents the binding between the SUD and LFC servers; and *WebDAVPROPFINDResponse* that generates the response according to the WebDAV protocol specification. After all the classes presented previously are instantiated, the WebDAVEngine is ready to accept requests from the users.

The **WebDAV Server** is composed by two FCGI applications (processes) that are implemented in the *cdn.webdav* package: *WebDAVEngine* and *WebDAVEngineSRM*.

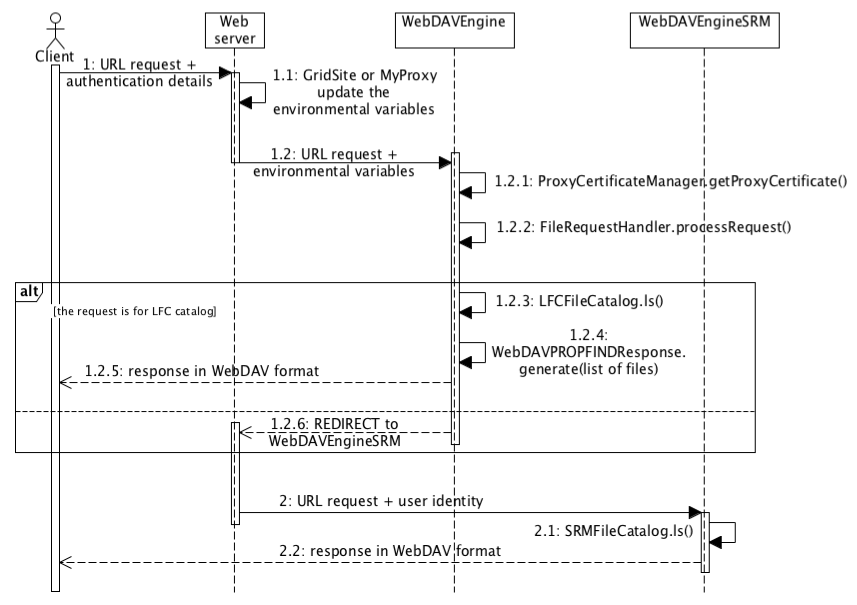


Figure – Interaction between the modules that compose the WebDAV Server

The **Cache Node** is composed by five FCGI applications (processes) that are implemented in the *cdn*.*cachenode*: *RequestHandler*, *LFCHandler*, *SRMHandler*, *CacheEngine*, *Downloader.* The *CheckCache* interface is part of the Cache Node, but it is not shown in the process view diagram.

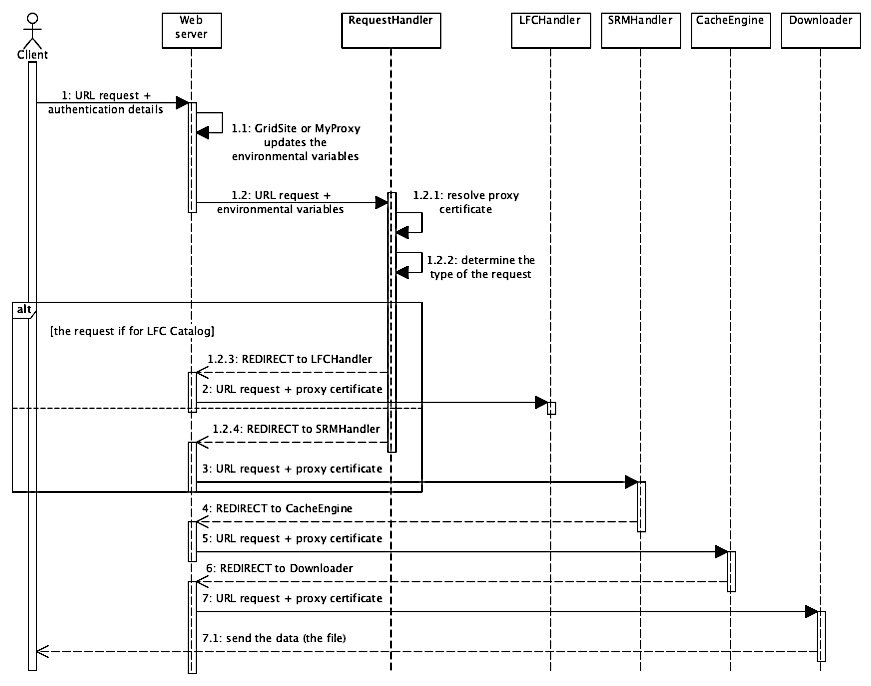


Figure – Interaction between the modules that compose the Cache Node

As presented in the figure above, the RequestHandler is the only application in which the user can make a direct request. For the other modules, only the web server is capable of sending requests.

## Process View

The web servers that host the SUD support parallel processing of the user requests. Because in the current project a user request can take large amount of time till completion, it is recommended to split the entire request in several smaller sub-requests.

In the case of WebDAV Server, even if the WebDAV-specific requests are lightweight, they are split into two processes (WebDAVEngine and WebDAVEngineSRM). This decision was made due to the *Segmentation Fault* error generated by the grid frameworks.

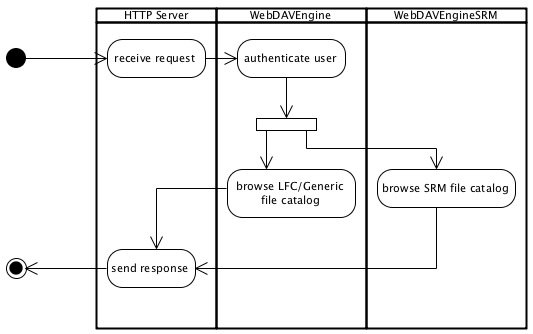


Figure – Division into multiple processes of the WebDAV Server

The Cache Node usually handles heavyweight requests for data. In this case a request basically is divided and distributed across 4 processes (*RequestHandler*, *LFCHandler* or *SRMHandler*, *CacheEngine* and *Downloader*), as presented in the figure below.

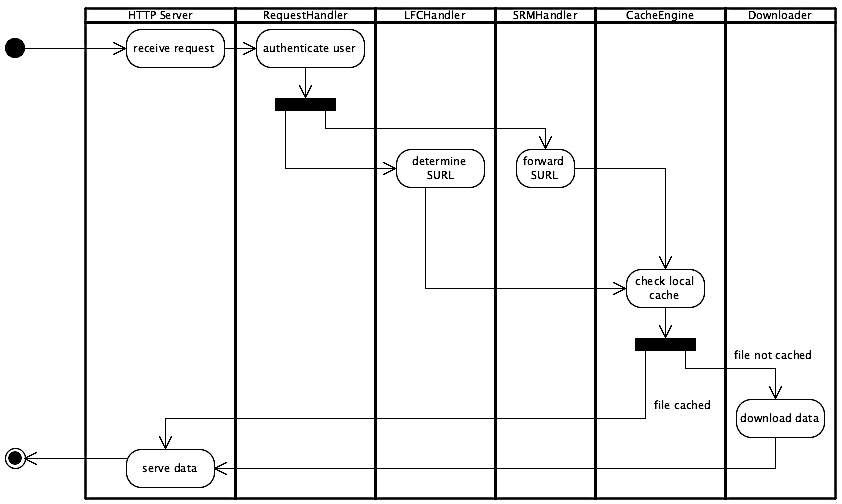


Figure – Division into multiple processes of the Cache Node

The heavier process is the *Downloader*, which copies data from the SE into the local cache of the Cache Node. If there are requests for files that are cached, and the *Downloader* process is busy, they will be resolved immediately by the CacheEngine, because no extra files need to be downloaded, thus the request does not need to wait for other files to be downloaded.

## Deployment View

A deployment diagram (see Figure 54) visualizes the topology of the physical components of a system where the software components are deployed. The current project can run with infinite number of Cache Nodes, but the figure above shows only one.

|  |
| --- |
| **Design decision:** The Delegation service is deployed on the WebDAV machine.  Motivation: Because the users connect to the WebDAV Server address (host), it is desirable that they can delegate their credentials to the same address (host). By deploying the Delegation Service to another machine, the user would need to remember two addresses (the WebDAV and the Delegation Service), which might be sometimes confusing. |

|  |
| --- |
| **Design decision:** The WebDAV server proxies all the requests from web browsers to the Cache Nodes (and not redirect).  Motivation: It is desirable that the users know only the address of the WebDAV Server, and not a particular address of a Cache Node (as they would know if the request would be redirected). |

|  |
| --- |
| **Design decision:** The NginX web server is used to transfer unencrypted data to Worker Nodes, instead of Apache2.  Motivation: According to several measurements carried out during the feasibility study, the NginX web server is more reliable and efficient than the Apache2. Because it is possible to easily integrate NginX within the current design, it is decided to use it. |

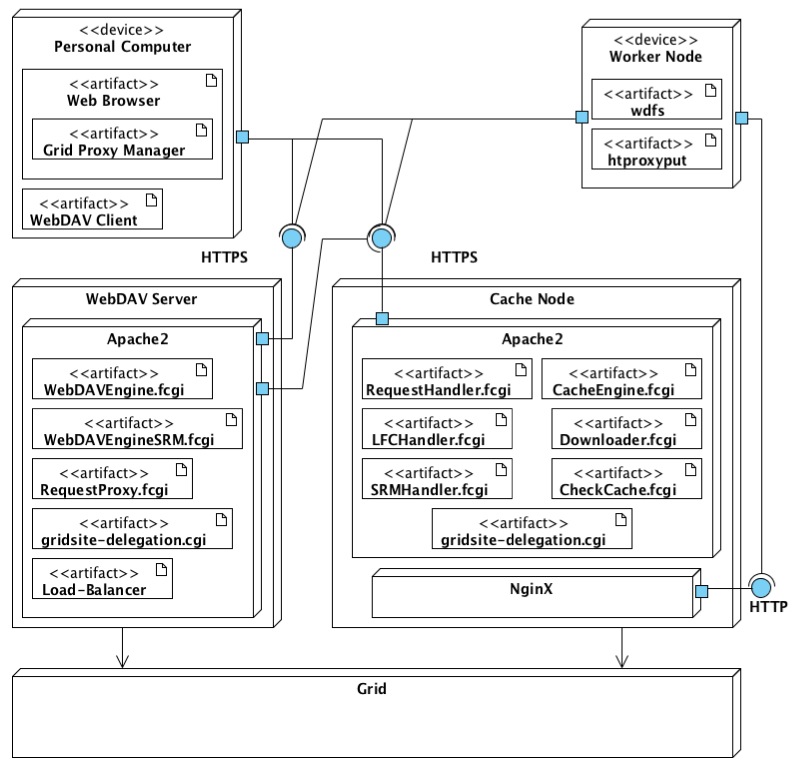


Figure – Deployment diagram of the SUD

The Cache Nodes need to be registered to Load-Balancer that resides on the WebDAV Server. Each of the Cache Node instances need to know the other instances in order to cooperate for cached files.

# Validation & Verification

*The chapter presents the methods used for validating the developed solution, as well as the description of the acceptance tests. The verification of the current project is addressed in the second part of this chapter and it describes the techniques used to confirm the correctness of the implementation. The experimental results from running the acceptance tests are discussed in the next chapters, in order to provide a good overview of the results.*

## Validation

The validation of SUD is divided in two main parts: functional validation, where all the functional requirements are evaluated separately; and non functional validation that consists in several sets of tests and scenarios that are trying to demonstrate the correctness of the system as well as they try to discover the bottleneck and the limitations of the entire architecture of the current project.

### Functional validation

#### A. WebDAV support (FR-1)

The first functional requirement (FR-1) states that it must be possible to mount the files and directories defined by the LFC and SE to a computer. Due to the results of the feasibility study and the *Survey*, the WebDAV protocol was decided to be the link between the user and the SUD. The current project implements the WebDAV protocol as a *server* and the implementation tries to follow the *RFC 4918[[28]](#footnote-28)* protocol specification.

**Evaluation process:**

To validate the quality of the WebDAV protocol implementation by the SUD and whether it follows entirely the protocol specifications, an extensive research was conducted to find the appropriate tool that would be able to validate it. The research on the Internet proved that the validation could not be done with any specialized testing tool (because such a tool does not exist) and in most of the cases the process is done manually and it includes the usage of a well-known WebDAV client.

The manual validation approach is done in order to validate the current project. The *davfs2*[[29]](#footnote-29)and *cadaver[[30]](#footnote-30)* WebDAV clients provided by the Linux operating system prove to be fully compatible with the protocol specification and these seem to be the most used clients to validate custom implementations of WebDAV protocol. A set of tests is defined in order to cover all the functionality a *read-only* WebDAV server must provide, and the tests are presented below.

Table – List of tests to validate the WebDAV implementation

|  |  |
| --- | --- |
| # | *Test description* |
| W-1 | Mount the WebDAV repository to the local computer |
| W-2 | Explore directories from Console (Terminal) |
| W-3 | Explore directories from GUI (File Explorer) |
| W-4 | Open file from Console (Terminal) |
| W-5 | Open file from GUI (File Explorer) |
| W-6 | Search for file |

The *davfs2* is the main client used to verify the WebDAV protocol implementation due to the fact that it allows mounting of the repository to the local file system of the computer and thus all the tests defined previously can be performed. The *cadaver* client is a command-line tool and it does not mount the repository to the local files system, thus only part of the tests (W-2, W-4 and W-6) can be performed, and they are meant for double-checking.

However, in order to ensure the quality of the implementation, and to reduce the time spent for testing, a shell script is developed. This script enables an automatic run of the tests presented above. The script is executed anytime changes are performed on the SUD implementation. Eventual implementation errors are thrown immediately and so, the time spent for debugging is significantly reduced.

**Evaluation results:**

During the development of the current project, all the tests defined previously are passed. This is meant to prove the WebDAV protocol implementation is fully compatible with chosen WebDAV clients, and it is consistent with to the protocol specifications.

Additional testing of the WebDAV protocol implementation is performed during the *Usability* tests that use different clients provided by different operating system, and execute the tests described before.

#### Authentication methods (FR-2)

According with the second functional requirement (FR-2), the current project must provide two authentication methods to the client: PKI and username-password. The PKI authentication involves usage of X.509 certificates, as well as X.509 proxy certificates and X.509 GSI proxy certificates.

**Evaluation process:**

To verify whether the current project is able to correctly authenticate the user, two sets of tests are implemented and each of the required authentication methods is evaluated. Firstly, the tests are focused on positive authentication with valid credentials (certificate or username and password) when the expected behavior is to grant the user to access the SUD.

Table – List of tests to validate the authentication methods

|  |  |  |
| --- | --- | --- |
| # | *Test description* | *Expected behavior* |
| A-1 | Authentication with correct username and password | Permission granted |
| A-2 | Authentication with valid X.509 Certificate |
| A-3 | Authentication with valid X.509 Proxy Certificate |
| A-4 | Authentication with valid GSI Proxy Certificate |

Secondly, the tests are focused on authentication with wrong credentials when the expected behavior is to deny the user from using the SUD.

Table – List of tests to authenticate the user

|  |  |  |
| --- | --- | --- |
| # | *Test description* | *Expected behavior* |
| A-5 | Authentication with wrong username and password | Permission denied |
| A-6 | Authentication with expired X.509 [GSI] [Proxy] Certificate |
| A-7 | Authentication with valid X.509 [GSI] [Proxy] Certificate, but without previous (valid) delegation done |

The tests are performed with the two WebDAV clients (*davfs2* and *cadaver*) presented in the previous chapter when possible, and in one case with *curl* (for X.509 GSI proxy certificate authentication). The tests cover all the possible scenarios that might happen when the user authenticates to the system and they check the SUD behavior.

To ease the process, again the tests are implemented as a shell script, and executed anytime when the implementation of the SUD changed.

**Evaluation results:**

All the tests successfully passed in both cases, proving that SUD supports all the required authentication methods as intended. Additional tests are performed during the *Usability* and *Performance* evaluation of the current project, with different WebDAV clients using various authentication methods.

#### C. Authorization (FR-3)

The data security is one of the most important issues in grid computing and one of the main functional requirements of the project. The validation of the current project from this perspective is a difficult process that requires special preparation of the test environment.

In real life, a file might be (or not) available for a specific user or a group of users. Once the user demands a file or a set of files from the Storage Element (SE), the security layer implemented by the SE checks whether the user is authorized to access that file or set of files. This is a secure process already implemented by the Grid’s components and nothing more can be done here.

However, while using the SUD, two possible scenarios may occur:

1. User requests a file or a group of files, which are not cached by the SUD

The file or the group of files must be transferred from the Storage Element. The SUD does not decide whether the user is allowed to read the file, but it forwards the request of the user to the SE, and the last decides if the user is allowed (or not) to read the file. In this scenario, the SUD is not involved in any security-related decisions, thus no tests are implemented.

1. User requests a file or the group of files, which are cached by the SUD

The SUD has a copy of the data. It must decide whether the user is allowed (or not) to access the cached file or group of files. It must preserve the same permission rules as defined by the Storage Element.

**Evaluation process**:

To validate the current project from security perspective, it is enough to validate whether the SUD has the same behavior as the SE when it must authorize a user to access the file or group of files. In other words, the SUD’s decision to serve a file or not to a user must be identical with the decision that the SE would take.

The tests cover all possible situations that might happen when the user wants to access a file. The test suite checks automatically a set of files on which the user has different types of permissions. It first tries to access the file directly from the Storage Element, and then it tries to access the file from the SUD. The test suite compares the outcome of the Storage Element and of the SUD and signals when there are differences.

Table 28 – List of tests to check user permission to access a file

|  |  |  |  |
| --- | --- | --- | --- |
|  | # | *Test description* | *Expected behavior* |
| File permission | FP-1 | Access file, with valid permission | Permission Granted |
| FP-2 | Access file that user did not have permission in the past but he/she currently has permission |
| FP-3 | Access file without permission | Permission Denied |
| FP-4 | Access file that user had permission in the past but does not have permission anymore |

All the tests previously defined cover most of the possible situations where the SUD could take a wrong decision regarding the data security.

The web servers that are used to implement the SUD, such as Apache2 and NginX, might generate other security issues, not related with the SUD implementation. Those issues do not take part of the SUD security-evaluation.

**Evaluation results:**

From this perspective, the SUD does not substantially affect the security level of data. Users permissions are checked against currently implemented security mechanism of the Grid, so no other special security mechanism is implemented by the SUD.

Most of the tests defined previously are successfully executed, and the behavior of the SUD is as expected, with one exception (FP-4).

The following situations might happen:



Figure – The case when a file can be accessed without permission

First, the user requires a file for which he/she has permission to read. The SUD downloads the file from the Storage Element, and caches the file and the user permission for a short period of time. Then, the System Administrator (or someone else), invalidate the permission to read the file of the user. Immediately after, the user requires the file again and even if he/she does not have permission to read the file anymore, the user will be allowed to access it (according with the SUD’s cache). This drives to an unauthorized access to the data.

However, this situation is not likely to happen, due to the fact that in real life the permissions of files do not change often. Decreasing the period of time for which the SUD caches the users permissions to access data, can easily solve the security issue, but will decrease the overall performances of the system.

### Nonfunctional validation

#### A. Usability (NFR-1, NFR-2, NFR-3)

The usability is the most important quality of the current project and it refers to the interaction between the users and the SUD. The premise that stands at the base of the entire design from usability point of view is the following: “*the users can access data from their personal computer without installing any other third party application*”.

This means that regardless the operating system the user can access the data by using the SUD. He/she can mount the WebDAV repository by making use of a WebDAV Client, or can explore and access data by using a Web Browser without installing any 3rd party applications or frameworks.

**Evaluation process:**

To validate the current project from usability perspective, a set of WebDAV clients, web browsers and command-line tools provided by different platforms are identified, and sets of tests are defined in order to check their compatibility with the SUD. The tests aim to identify what are the supported operations and whether they can successfully connect to the SUD.

The first set of tests is focused on the authentication methods supported by the WebDAV clients.

Table – List of tests to validate the authentication methods

|  |  |  |  |
| --- | --- | --- | --- |
|  | # | *Test description* | *Platform / Tools* |
| WebDAV Client | 1 | Authentication with username and password | Windows XP  Windows 7  Mac OS X  Linux |
| 2 | Authentication with X.509 Certificate |
| 3 | Authentication with X.509 Proxy Certificate |
| 4 | Authentication with GSI Proxy Certificate |

The second set of tests is focused on the WebDAV protocol and the compatibility between the SUD and different WebDAV clients.

The WebDAV protocol has native support in all of the most-used-operating-systems currently on the market (Mac OS X, Windows, Linux). However, its implementation does not fully follow in all the cases the protocol specification. Due to this fact, all the WebDAV clients provided by the chosen platforms (and other freeware third party application) are manually tested. The tests check whether the required operations (by the SUD as read-only repository) have the intended behavior.

Table – List of tests to validate the WebDAV implementation

|  |  |  |  |
| --- | --- | --- | --- |
|  | # | *Test description* | *Platform* |
| WebDAV Client | 1 | Mount the WebDAV repository to the local computer | Windows XP  Windows 7  Mac OS X  Linux |
| 2 | Explore directories from Console (Terminal) |
| 3 | Explore directories from GUI (File Explorer) |
| 4 | Open file from Console (Terminal) |
| 5 | Open file from GUI (File Explorer) |
| 6 | Search for file |

The third set of tests focuses on accessing the data by using the web browsers.

As the WebDAV is an extension of HTTP standard, the WebDAV repository (SUD) is also accessible by using the HTTP protocol, and in particular web browsers. The important operations that a user can make are: **authentication;** **browse directories**; and **download files**. The tests check whether all the operations are supported.

Table – List of tests to validate the HTTP access to data

|  |  |  |  |
| --- | --- | --- | --- |
|  | # | *Test description* | *Tools* |
| Web Browser | 1 | Authentication with username and password | Mozilla Firefox  Google Chrome  Internet Explorer  Safari |
| 2 | Authentication with X.509 Certificate |
| 3 | Authentication with X.509 Proxy Certificate |
| 4 | Authentication with GSI Proxy Certificate |
| 5 | Explore directories |
| 6 | Download files |

The forth set of tests focuses on accessing the data by making use of different command line tools such as *wget* and *curl*.

Table – List of tests to validate the curl and wget

|  |  |  |  |
| --- | --- | --- | --- |
|  | # | *Test description* | *Platform* |
| Other | 1 | Authentication with username and password | Linux |
| 2 | Authentication with X.509 Certificate |
| 3 | Authentication with X.509 Proxy Certificate |
| 4 | Authentication with GSI Proxy Certificate |
| 5 | Download files |

The results of the evaluation are presented in the chapter 8.2.1

#### B. Performance

The performances of the current project must be seen from two different perspectives. First, the regular user that connects from his/her personal computer to SUD and requires data, and second, the Worker Node (WN) that connects to the WebDAV repository.

##### Regular user

In the first case, the assumption is that the users that connect from their personal computers to the SUD do not demand huge amounts of data, because of the limited resources that they have. The users need fast exploration of files and directories, and fast access to relatively small files.

**Evaluation process:**

To evaluate the performances of the system from this perspective, a set of tests is developed, and they simulate different scenarios that might happen during the real exploitation of the SUD. The main parameters that vary during this evaluation are the number of concurrent users (from 1 up to 200), and the type of request (*browse directory* and *open file*).

Table – List of request types that a user can make

|  |  |
| --- | --- |
| *Req. type* | *Description* |
| Browse directory | 1. Authenticate the user; 2. Query the file catalogs (such as LFC or SE) for files contained by the required directory; |
| For this type of request, the number of files contained by the directory impacts the time measured. For test purposes, all the directories contain ***200 files*** each. |
| Open file | 1. Authenticate the user; 2. Identify the SURL of the LFN; 3. Check the user’s permissions to access the file; 4. Deliver the file; |
| For this type of request the time spent to transfer the file is not taken into account because in reality it depends on the network connection and other external factors. However, for test purposes, the file has less than ***1KB*** and the transfer time is assumed to be 0s. |

The outcome of the tests highlights for how long the user must wait until all his/her requests are solved by the SUD.

Table – List of tests performed to measure the overhead introduced by the SUD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *Concurrent users* | | | | |
| *Request type* | 1 | 10 | 50 | 100 | 200 |
| Browse a directory | *Average duration time to resolve the request* | | | | |
| Open a small file |

For each of the tests presented in the table above, the memory-cache will be once active, and once inactive. The comparison between the two cases shows the improvement in performance (due to the usage of memory-cache) when the users open or browse the same sets of files or directories. The tests are performed from the same machine where the WebDAV Server is running in order to eliminate the variations and overhead introduced by the network connection.

All the tests are performed with the Apache Benchmark (ab)[[31]](#footnote-31) tool, and the full description of the scripts is presented in the *Appendix I*.

##### Worker Node

In the second case, when the Worker Nodes connects to the WebDAV repository (SUD) and requires data, the data-transfer is essential in measuring the performances of the system. In this scenario, the assumption is that the jobs that are executed by the WN usually know the file that must be processed and they are not performing extensive file searching / directory browsing. Instead, the size of the files is rather large and then the scenario becomes *data-intensive*.

The results of this case are highly dependent by the network connectivity and the hardware where the SUD is deployed (this is presented in *Deployment* sub-chapter).

The performance and the scalability of the SUD are measured as follows:

**Performance**: is quantified as the inverse of the time needed by the job to completely finish its execution by using the SUD. The value is compared with the execution inverse time of the same job that does not use the SUD. The time difference represents the gain or the loss that is related with the usage of the current project.

|  |
| --- |
| *-* Performance of the SUD  *-* Performance of the current system  - Gain in performance |

**Scalability:** is defined as the possibility to add multiple resources to increase the performance and robustness of the system. There is no standard definition of scalability as a quantifiable value. However, in this report, the quantifiable value of scalability is represented as the relative gain in performance of the system with more resources, compared with the system with fewer resources (further on it is called *efficiency*).

|  |
| --- |
| N < M; *N and M represents the number of Cache Nodes*  - *Performance of the SUD that has N Cache Nodes*  - *Performance of the SUD that has M Cache Nodes*  - *Efficiency:* *the relative gain in performance if the number of Cache Nodes is increased from N to M.* |

The performance of the SUD is highly coupled (1) with the number of Cache Nodes that are installed, (2) with the number of concurrent jobs, (3) with the size of the files used by the jobs and (4) with the hardware configuration of each of the system components.

**Evaluation process:**

In order to be able to evaluate the performance and scalability achievements of the SUD, several tests were implemented with the attempt to depict different situations that happen in real life.

From process perspective, there are two interesting real situations that might occur:

**Scenario 1**: Multiple jobs require the same set of data from the SUD. This is the case when the SUD should give the best performance due to the fact that the data is cached once (if not previously cached) and used several times after.

**Scenario 2**: Multiple jobs require all different files. This is the case when the SUD cannot perform better than the current system, because all the data must be copied from the Storage Element to Cache Node (if not previously cached) and from the Cache Node to the client, instead of copying the data directly from the Storage Element to the client (as it currently works). However, it is interesting to determine the differences between the two architectures.

From SUD perspective, there are two possible scenarios:

**Case 1 (C1)**: The data required by the jobs is already cached by the SUD, and it is served directly to the Worker Node. This is the scenario when SUD gives the best results, due to the fact that the interaction with the grid components is minimum, and the SUD can deliver data directly to the Worker Node.

**Case 2 (C2)**: The data required by the jobs is not cached by the SUD, thus it needs first to be cached, and then served to the Worker Node. This represents the worst case for the SUD, because instead to serve data to the Worker Node, it must copy it first from the Storage Element, cache it, and only after send the required data to the Worker Node.

Table – Parameters changed during the performance tests

|  |  |
| --- | --- |
| *Parameter* | *Values* |
| Numbers of concurrent jobs | 1 up to 104 |
| Size of the input files | 100 MB / 1GB / 4 GB |
| Number of Cache Nodes | 1 up to 3 |
| Type of the transfer | HTTPS (encrypted) / HTTP (unencrypted) |

All cases and scenarios previously presented are combined and covered by multiple tests. The main parameters (that are presented in the table above) that influence the results are sequentially changed in order to provide a complete and correct overview of the SUD.

Table – List of tests performed to calculate the Performance and the Scalability of the SUD (✔ - Test is performed; ✖ - Test is NOT performed)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | *Concurrent jobs* | | | | | | | *Transfer type* | | *Case* | |
| *File size* | 1 | 10 | 20 | 40 | 60 | 80 | 104 | HTTP | HTTPS | C1 | C2 |
| *1 Cache Node* | 100 MB | Scenario 1; Scenario 2; | | | | | | | ✔ | ✖ | ✔ | ✔ |
| **1 GB** | Scenario 1; Scenario 2; | | | | | | | ✔ | ✔ | ✔ | ✔ |
| 4 GB | ✖ | | | | | | | ✖ | ✖ | ✖ | ✖ |
|  |  |  | | | | | | |  |  |  |  |
| *2 Cache Nodes* | 100 MB | Scenario 1; Scenario 2 | | | | | | | ✔ | ✖ | ✔ | ✔ |
| **1 GB** | Scenario 1; Scenario 2; | | | | | | | ✔ | ✔ | ✔ | ✔ |
| 4 GB | Scenario 1; | | | | | | | ✔ | ✖ | ✔ | ✔ |
|  |  |  | | | | | | |  |  |  |  |
| *3 Cache Nodes* | 100 MB | ✖ | | | | | | | ✖ | ✖ | ✖ | ✖ |
| **1 GB** | Scenario 1; Scenario 2; | | | | | | | ✔ | ✖ | ✔ | ✔ |
| 4 GB | ✖ | | | | | | | ✖ | ✖ | ✖ | ✖ |

The table above shows all the tests that are interesting to evaluate the SUD. As obvious, the tests focus on the case with **1GB** file, which is considered medium file size. The other sizes of the input files are used to identify better the reasons for which the SUD performs as it does. Most the tests involve unencrypted data transfer due to the fact that the network between the SUD and Worker Nodes is private and secure. However, a test measures the performances of the SUD when the transfer is encrypted and compares it with the current system.

The results are presented in the *Results* chapter and they are compared with the performances of the current system, without using the SUD.

###### Jobs description

In order to execute the tests, three types of jobs are implemented. The first two sets use the current framework of the Grid (gLite). The difference between them is the type of the data transfer: encrypted or unencrypted. The third type is implemented to use the SUD, and it contains several operations that are required to mount and to use the WebDAV repository (SUD).

1. Job that uses *lcg-cp* (unencrypted data transfer)

Table – Description of the job that uses *lcg-cp*

|  |  |  |
| --- | --- | --- |
| *#* | *Operation* | ***Description*** |
| 1 | **date** +%H:%M:%S | Start timer |
| 2 | **lcg-cp** […] | Copy the file from Storage Element |
| 3 | **cksum** […] | Calculates the checksum of the file |
| 4 | **date** +%H:%M:%S | Stop timer |
|  | ***Total time composition*** | **Copy + Checksum** |

The table above presents the logical description of a job that uses *lcg-cp* command to copy data from the Storage Element (the transfer is unencrypted). The first and the last operations calculate the total duration of the job. The “lcg-cp” operation copies the data to the Worker Node. Once the data is copied, it is passed to the third operation that calculates the checksum of the copied file.

After competition, the file that is copied from the SE is removed automatically by the WN, and thus, the total duration of the job does not include the time required for cleanup.

1. Job that uses *globus-url-copy* (encrypted data transfer)

Table – Description of the job that uses *globus-url-copy*

|  |  |  |
| --- | --- | --- |
| *#* | *Operation* | ***Description*** |
| 1 | **date** +%H:%M:%S | Start timer |
| 2 | **lcg-gt […]** | Transform the LFN to TURL |
| 3 | **globus-url-copy** […] | Copy the file from Storage Element (encrypted) |
| 4 | **cksum** […] | Calculates the checksum of the file |
| 5 | **date** +%H:%M:%S | Stop timer |
|  | ***Total time composition*** | **LFN->TURL + Copy + Checksum** |

As in the first case, the table above describes the logic of a job that uses *globus-url-copy* command to copy the data from the Storage Element. The only difference is that the lcg-cp command is replaced by two other commands: *lcg-gt* that returns the TURL (Transfer URL) of a file; and the *globus-url-copy* command that copies securely the data from the Storage Element to the Worker Node. Again, after completion, the downloaded file is removed automatically by the WN.

1. Job that uses the SUD (WebDAV Repository)

Table – Description of the job that uses the SUD

|  |  |  |
| --- | --- | --- |
| *#* | *Operation* | ***Description*** |
| 1 | **date** +%H:%M:%S | **Start timer** |
| 2 | **htproxyput** […] | **Delegate credentials to SUD** |
| 3 | **wdfs-redirect** […]& | **Mount the WebDAV repository (SUD)** |
| 4 | **sleep** 2 | **Wait 2 seconds for mounting to be done** |
| 5 | **cksum** […] | **Calculate the checksum of the file** |
| 6 | **fusermount** […] | **Unmounts the WebDAV repository (SUD)** |
| 7 | **sleep** 5 | **Wait 5 seconds for unmounting to be done** |
| 8 | **date** +%H:%M:%S | **Stop timer** |
|  | ***Total time composition*** | **Delegation + Copy + Checksum + 7 seconds** |

Unlike the other two job types, the job that uses the SUD is more complex and multiple operations that in the first cases are handled automatically by the WN’s framework now are handled manually. Because the current Grid’s frameworks do not delegate automatically the user’s credentials to the SUD, this step is performed manually by the *htproxyput* command. The WebDAV Repository (SUD) must be mounted manually to the Worker Node with *wdfs-redirect* command. Because a job cannot start a daemon-application (as the wdfs-redirect WebDAV client is by default), the job starts the WebDAV client as a normal application and it spawn the new process. Before executing the new command, the job must wait around 2 seconds until the connection to the SUD is established. The 2 seconds waiting time is an estimated value determined by several experimental tests. Once the connection is established, the *checksum* command is executed and it gets as input directly the location of the file defined as LFN or SURL. The file is copied automatically to the WN, without any extra command required.

After the file is processed, the job unmounts the WebDAV repository manually. This is equivalent with the cleanup of the downloaded files from the previous cases handled automatically by the WN. This is a mandatory step because the WN cannot unmount automatically the repository. To unmount the repository, *fusermount* is executed. An extra 5 seconds amount of time is required to complete the command. This value is again determined experimentally, and it proved to be sufficient under stress condition of the system.

However, all the results related to the SUD performances presented in the *Result* (8.2.2) chapter, contain the duration of all commands described before (including the *sleep* command).

#### Robustness

The robustness of the current project must be interpreted from two perspectives: when the load of the system is very high and it reaches its physical limits; and its capability to recover from unexpected situations and errors.

**Evaluation process:**

For the first perspective, the robustness of the entire system is quantified as the difference between the number of jobs that fail to execute when using the SUD, and the number of jobs that fail to execute when the SUD is not used. There are multiple reasons for a job to fail, but for the robustness evaluation, only the jobs that fail because of the input data are counted. The robustness measurement, in this case, is a straightforward process that goes along with the performance tests. While running the tests, the SUD reaches its limit, moment when the robustness is challenged, thus the results of the performance tests will reveal how robust is the current project.

For the second perspective, in order to assure high levels of robustness of the SUD, each of its components must be evaluated separately, to check whether in case of errors the system recovers or not. A set of tests is defined in order to simulate different expected and unexpected scenarios that can happen. The main expectations are that the system will not crash and it will recover to the working state again. The description of each of the tests is presented in the table below, and the results are presented in the *Results* chapter.

Table – List of tests to check the robustness of the SUD when unexpected scenarios happen; or when unexpected errors occur

|  |  |  |  |
| --- | --- | --- | --- |
|  | # | *Test description* | *Expected behavior* |
| WebDAV | 1 | Modify files from the mounted repository | - Error: *Operation not permitted*  - Rollback all the changes |
| 2 | Delete files/directories from the mounted repository | - Error: *Operation not permitted*  - Rollback all the changes |
| 3 | Require files that not exist | - Error: *File not found* |
| 4 | Browse folders that not exist | - Error: *File not found* |
| Cache Node | 1 | Exceed the maximum size of the cache | - Remove oldest accessed files to make space for the new file |
| 2 | Amount of the required data at a moment exceeds the size of the cache | - Error: *File not found*  - Remove all the partial downloads |
| 3 | None of the replicas of a file specified by LFC exist | - Error: *File not found* |
| 4 | Some of the replicas of a file specified by the LFC exist | - Try each of the replicas until a valid one is found |

## Verification

Verification is a process that makes sure the software product is developed in a proper way. Although this may be a vague definition, there are various techniques, which assist in examining the quality of the software, and preventing potential bugs, which may cause software failures.

The implementation of the SUD is rather simple and it tries to combine as much as possible already existing technologies that prove to be reliable and bugs-free. The amount of code that is written is used mainly to interconnect all these technologies. However, part of the code that composes the SUD, is meant to implement the logic that stands at the base of the design, and parts of the functional requirements.

The verification is in general done manually and by making use of unit-tests. The manual verification is done in the cases when a unit-test is very hard to implement or when the necessary time to implement the test would be much higher than testing it manually. The unit-tests are implemented only for the code that is highly reused within the implementation. Because most of the code is written in Python and the amount of code is rather small, the number of unit-tests is small. The unit-tests and the manual verification are performed anytime major changes are applied to the code.

However, there are two modules implemented in C++ (the *wdfs-redirect* and the *mod\_authn\_myproxy*) that are modified. They are verified with the *valgrind[[32]](#footnote-32)* tool for memory leaks. The results show that the modified implementations have a good quality and no memory leaks or any other errors are introduced.

# Results

*The chapter presents, in the first part, the deployment and test setup that was used in order to evaluate the acceptance tests defined in the previous chapter. In the second part, the experimental results of the test designed in order to interrogate the solution with respect the stakeholders’ needs are discussed, and short conclusions are formulated after each of the tests.*

## Deployment and test setup

To ensure that the tests defined in the previous chapter are executed in optimal conditions and no external factors affect the results, a number of sixteen computers are isolated from the Grid’s environment and prepared for the SUD to be deployed and tested. These computers compose the deployment and test environments of the current project, and they are presented in the next sections.

### Deployment environment

Three machines are dedicated only to host the SUD components and their configuration is presented in the table below.

Table – Configuration of the machines where the SUD is deployed

|  |  |  |
| --- | --- | --- |
| *Deployment machines (tbn21.nikhef.nl, tbn22.nikhef.nl, tbn28.nikhef.nl)* | | |
| 3 x | *CPUs* | 2 x Intel(R) Xeon(R) CPU L5520 @ 2.27GHz |
| *Total cores number* | 8 |
| *Memory (RAM)* | 24 GB |
| *Local disk* | 493 GB |
| * *Reading throughput* | 120MB/sec |
| * *Read-write throughput* | 40MB/sec |
| *Network connectivity* | 10 Gbps |

The *WebDAV Server* is in all the cases deployed on only one machine due to its low resource demands. The base deployment configuration uses the remaining two machines as *Cache Nodes*. During the tests (defined in the previous chapter), the deployment configuration changes in order to test the scalability and performances of the current project.

### Test environment

Fifteen dedicated machines compose the testing environment for the SUD. They are divided in *Worker Nodes* (13 machines) and *Disk Servers* (2 machines).

Table – Configuration of the Worker Nodes

|  |  |  |
| --- | --- | --- |
| *Worker Nodes (tbn23.nikhef.nl…tbn27.nikhef.nl, tbn29.nikhef.nl…tbn36.nikhef.nl)* | | |
| 13 x | *CPUs* | 2 x Intel(R) Xeon(R) CPU L5520 @ 2.27GHz |
| *Total cores number* | 8 |
| *Memory (RAM)* | 24 GB |
| *Local disk* | 463 GB |
| * *Reading throughput* | 120MB/sec |
| * *Read-write throughput* | 40MB/sec |
| *Network connectivity* | 10 Gbps |

As presented in the table above, the Worker Nodes have all 10Gb/sec network bandwidth and in total they can all demand data with a speed up to 130Gb/sec. They have in total 104 cores, which means that 104 jobs can be executed in the same time.

Table – Configuration of the Disk Servers that compose the Storage Element

|  |  |  |
| --- | --- | --- |
| *Storage Element – Disk Servers (hooimaand-01.nikhef.nl, hooimaand-02.nikhef.nl)* | | |
| 2 x | CPUs | 2 x Intel(R) Xeon(R) CPU L5520 @ 2.27GHz |
| Total cores number | 8 |
| Memory (RAM) | 72 GB |
| Local disk | 240 GB |
| * *Reading throughput* | 600MB/sec |
| * *Read-write throughput* | 100MB/sec |
| Network connectivity | 10 Gbps |

The difference between these two computers and the others is the size of the available memory (**74 GB** RAM each compared with **24 GB** for the other machines), and their hard disks configuration, that can read with up to **600 MB/sec** unlike the others where the speed is around **140 MB/sec**.

### Test and deployment environments overview

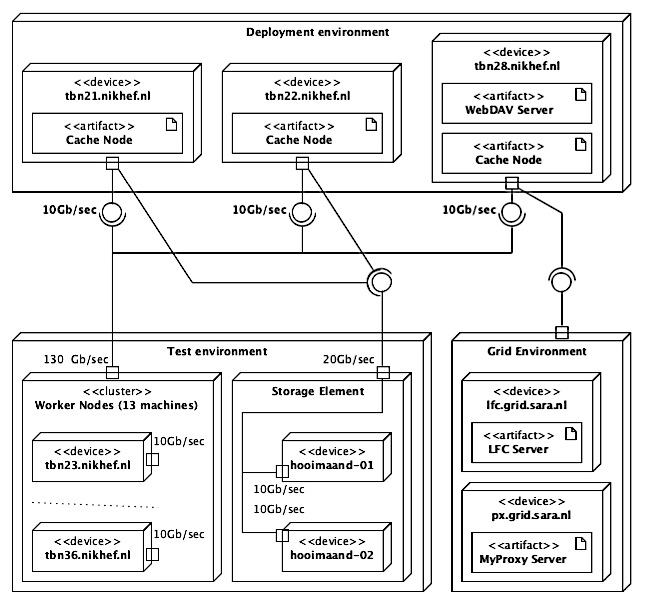


Figure – Test and deployment environments overview

## Experimental results

### Usability

The results from Usability point of view (as defined in 7.1.2) are concerned only about the interaction between the user and the SUD. Aspects such as authentication methods and operations that are supported by different tools, such as WebDAV Clients, web browsers and command line tools, are presented in the following sections.

#### WebDAV clients

The main functionality provided by the current project is the possibility of the user to connect to the SUD by making use of the WebDAV protocol. In order to use the protocol, the user must use aWebDAV client to the SUD. As described in the previous chapter, several WebDAV clients are investigated and their capabilities and compatibility with the SUD are presented below.

**Supported authentication methods:**

Table – Authentication methods supported by the WebDAV Clients (✔ - authentication method is accepted; ✖ - authentication method is NOT accepted)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operating System | WebDAV Client | Authentication | | | |
| Username / password | X.509 Certificate | X.509 Proxy Certificate | GSI Proxy Certificate |
| Windows XP | MiniRedirector | ✔ (partially) | ✖ | ✖ | ✖ |
| Windows 7 | MiniRedirector | ✔ (partially) | ✖ | ✖ | ✖ |
| Linux | davfs2 | ✔ | ✔ | ✖ | ✖ |
| wdfs | ✔ | ✔ | ✖ | ✖ |
|  | wdfs-redirect[[33]](#footnote-33) | ✔ | ✔ | ✔ | ✔ |
| Mac OS X | DavFS | ✔ | ✖ | ✖ | ✖ |

The table above shows the compatibility matrix between the different types of authentication and the WebDAV clients. All the investigated WebDAV clients support the *username and password* authentication method without requiring any changes or special configurations, with two exceptions. The *Windows XP* and *Windows 7* default WebDAV clients do not accept by default basic authentication with username and password over the HTTP or even HTTPS. This is a known issue with both clients and can be solved by changing the value of a key in Windows Registry. The required changes are described in the *Appendix D*. The *X.509 Certificate* authentication method is supported only the Unix WebDAV clients, but they do not support the *X.509 Proxy* or *X.509 GSI Proxy Certificate.* None of the tested WebDAV clients support GSI Proxy Certificate authentication method (except the *wdfs-redirect*).

**Supported operations and capabilities:**

Table – Operations supported by the WebDAV Clients (✔ - operation is supported; ✖ - operation is NOT accepted)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Operating System | WebDAV Client | Operations | | | |  |
| Mount | Browse directories | Read file | Search files | Redirect |
| Windows XP | MiniRedirector | ✔ | ✔ | ✔ (partially) | ✔ | ✖ |
| Windows 7 | MiniRedirector | ✔ | ✔ | ✔ | ✔ | ✖ |
| Linux | davfs2 | ✔ | ✔ | ✔ | ✔ | ✖ |
| Wdfs | ✔ | ✔ | ✔ | ✔ | ✖ |
|  | wdfs-redirect | ✔ | ✔ | ✔ | ✔ | ✔ |
| Mac OS X | DavFS | ✔ | ✔ | ✔ | ✔ | ✔ |

The table above shows the operations and capabilities of the different WebDAV clients investigated. The capability to mount the WebDAV Repository (SUD) to the local computer is supported by all WebDAV clients. All the investigated WebDAV Clients support the *browse directories*, *read file* and *search file* operations, with one exception. The Windows XP’s default WebDAV Client, cannot open a file directly. Instead, it starts the web-browser, asks again for username and password, and starts downloading the file as it would be downloaded from a web site. This is a common issue of this particular client.

###### Example of usage:

1. WebDAV Repository exploration with Finder (the Mac OS X’s file explorer)

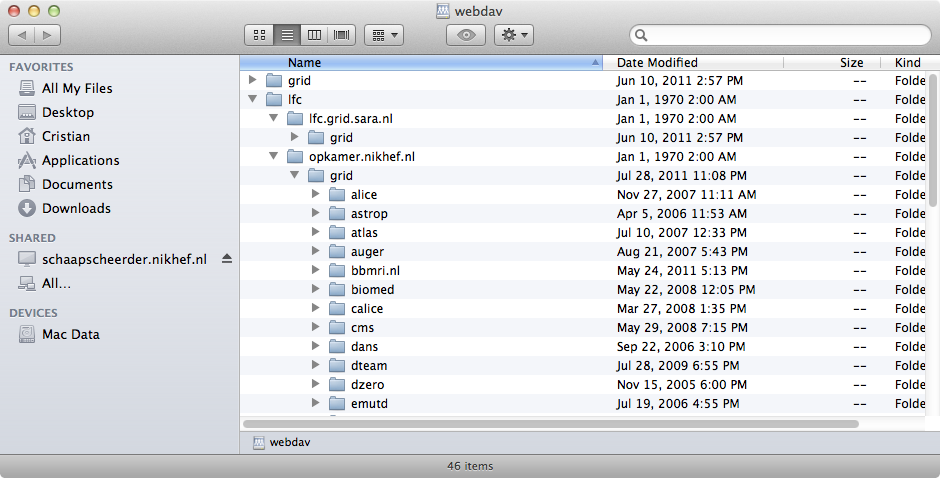


Figure – Mounted WebDAV Repository on a Mac OS X system

Once the WebDAV Repository (SUD) is mounted to the local computer, all the files defined by different LFCs (or SEs) are visible to the user. In the figure above, the current user is allowed to access *lfc.grid.sara.nl* and *opkamer.nikhef.nl* LFC servers. The user can browse through the directories defined by each of the catalogs, and open or copy files on the personal computer.

1. WebDAV Repository exploration with Terminal (the Mac OS X’s console)

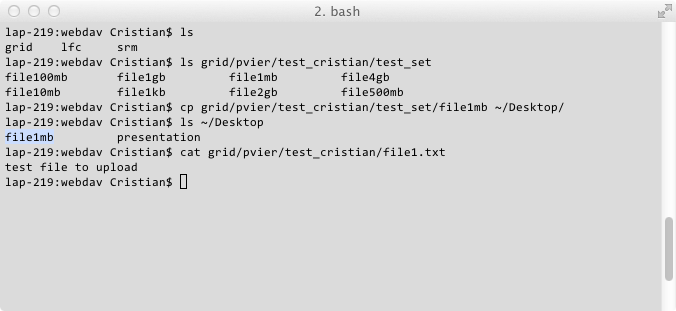


Figure – Mounted WebDAV Repository explored with the Terminal

The mounted WebDAV repository (SUD) can be accessed with a console (Terminal). The user can execute from his/her computer commands such as *ls*, *cp*, *cat* or any other commands that do not modify the content of the files or directories.

According to the results, the WebDAV repository can be accessed from any operating system, with the default client without installing any kind of applications or tools.

#### Web browsers

Four (Mozilla Firefox, Internet Explorer, Safari and Google Chrome) web browsers are tested in order to check their compatibility with the SUD.

Table – Compatibility matrix between web browsers and SUD

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Web Browser | Authentication | | | | Operation | |
| Username / password | X.509 Certificate | X.509 Proxy Certificate | GSI Proxy Certificate | Browse directories | Download file |
| Mozilla Firefox | ✔ | ✔ | ✖ | ✖ | ✔ | ✔ |
| Internet Explorer | ✔ | ✔ | ✖ | ✖ | ✔ | ✔ |
| Safari | ✔ | ✔ | ✖ | ✖ | ✔ | ✔ |
| Google Chrome | ✔ | ✔ | ✖ | ✖ | ✔ | ✔ |

All the investigated web browsers, presented above, are able to connect to the SUD. The authentication methods supported by all of them are *username and password,* and *X.509 Certificate*. *Browse directories* and *download file* operations are also supported by all the web browsers.

###### Example of usage

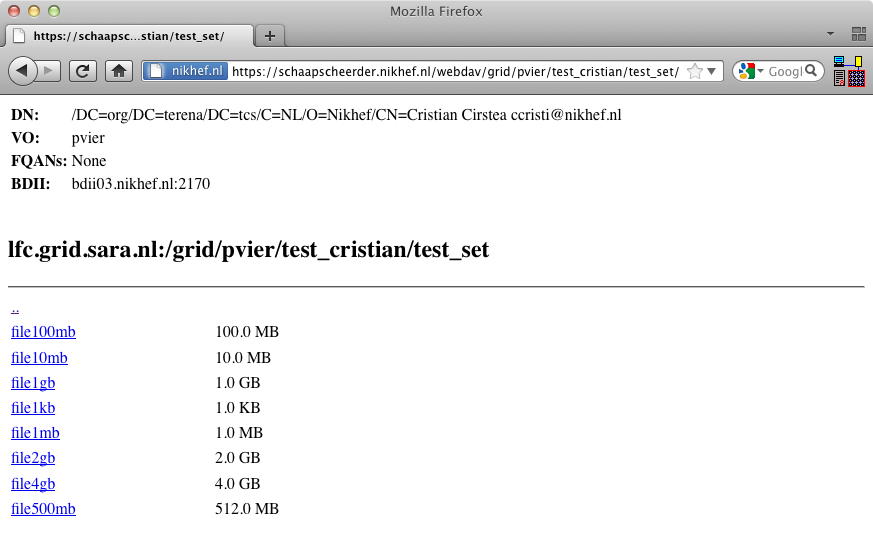


Figure – Browsing the WebDAV Repository with Firefox

#### Command line tools

Table – Compatibility matrix between curl and wget, and SUD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Web Browser | Authentication | | | | Operation |
| Username / password | X.509 Certificate | X.509 Proxy Certificate | GSI Proxy Certificate | Download file |
| curl | ✔ | ✔ | ✔ | ✔ | ✔ |
| wget | ✔ | ✔ | ✖ | ✖ | ✔ |

Both *curl* and *wget* can connect to, authenticate and download data from the SUD. The difference is that the *wget* only supports username and password, and X.509 Certificate authentication methods. Instead, *curl* can use all the methods. The reason why the *wget* cannot use *X.509 Proxy Certificate* and *GSI Proxy Certificates* is explained in the Appendix F.

###### Example of usage:

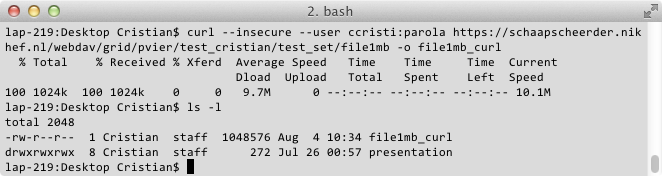


Figure – Downloading a file from SUD, by using *curl*

To conclude, the *Usability* of the current project has been proven by the results described above. The user can mount or browse the WebDAV repository with a large variety of tools. There is no reason why other than the investigated tools would not work.

### Performance

#### Regular User

As explained in the *Validation & Verification* chapter, the users need fast exploration of files and directories, and fast access to relatively small files. To measure the performances of the SUD and to calculate how many requests per second can be managed by it, two sets of test are performed.

The first set, simulate the scenario when multiple concurrent users (form 1 up to 200) utilize the SUD and the *memory-cache* is not active, and the results are presented in the next table.

Table – Time required by the WebDAV server to resolve requests when the memory cache is disabled

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | *Concurrent users* | | | | | | |
|  | 1 | 10 | | 50 | 100 | | 200 |
| *Request type* | *Average duration per request (sec)* | | | | | | |
| Browse a directory | 0.49 | | 0.71 | 3.7 | | 7.36 | 9.15 |
| Open a small file | 0.5 | | 0.84 | 4.74 | | 9.10 | 11.48 |

The results present the average time required by a number of concurrent users to wait until their request to browse a directory or the open a file is resolved. Because the memory cache is inactive, for each request the SUD connects to the LFC and SE. The results are as expected, and they prove that the performance of the current project are limited by the grid’s components[[34]](#footnote-34). In the case when **200** users utilize the SUD, the time that they must wait for a request is around 10sec.

Table – Time required by the WebDAV server to resolve requests when the memory cache is enabled

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | *Concurrent users* | | | | | | |
|  | 1 | 10 | | 50 | 100 | | 200 |
| *Request type* | *Average duration per request (sec)* | | | | | | |
| Browse a directory | 0.038 | | 0.15 | 0.67 | | 1.83 | 3.6 |
| Open a small file | 0.036 | | 1.14 | 0.63 | | 1.1 | 2.8 |

The results show a very consistent improvement in performances. In average, the time required to resolve a request is around **4** times shorter. Thus, in the case when there are **200** concurrent users that browse the same directories, the duration of the request drops from **10** to **3.6** sec.

To conclude, a relatively good machine that hosts the WebDAV server can handle around 200 concurrent users with reasonable performances. If the *memory-cache* is active, the required time to resolve a request reduces with up 75%. This comes with the cost that the responses received from the SUD might be outdated (during the time when for example, a directory is cached in memory, someone might remove or add more files to it). However, usually the files and directories are not removed from LFC or SE. When files are added to some directories from LFC or SE that are cached by the SUD, they are not visible till the cache expires. Anyway, usually the data is cached in memory for a short period of time (order of minutes or hours).

#### Worker Node

As explained in the *Validation and Verification* chapter, to evaluate the performances of the SUD when the Worker Nodes use it, there are two main scenarios likely to happen, and for each of them there are two cases.

|  |  |
| --- | --- |
| **Scenario 1**  All the jobs require the same set of data at the same time. | **Scenario 2**  All the jobs require different set of data at the same time. |
| **Case 1**  The data is not cached by the SUD (*worst case for SUD*) | **Case 2**  The data is already cached by the SUD (*best case for SUD* |

Most of the tests execute jobs that process files with the size of **1GB**. All the jobs consist in calculating the checksum of a file that is found on the Storage Element, and check whether the result is correct. However, there are additional tests with input files of **100MB** and **4GB** and these are meant to determine what is the limiting factor in the entire system, e.g., network bandwidth, memory size or hard disk speed.

**Scenario 1: All the jobs require the same set of data at the same time**

Firstly, the performance of the current system (without using the SUD) is measured to have the parameter of comparison that serves as a reference. This test is called “lcg-cp” due to the fact that the gLite “*lcg-cp*” command is used to copy data from the Storage Element to the Worker Nodes.

**Test 1:** Average execution time of jobs that are using *lcg-cp*

For the first test, the file size that is analyzed is 1GB and only one disk server (out of two) stores it.

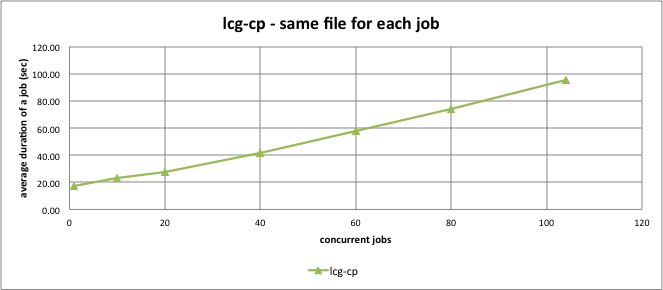


Figure – Average execution time of jobs that need the same file and use lcg-cp

The graph shows a linear increase of the average time needed by the jobs to complete, with the number of jobs that are running in parallel. This proves that the Storage Element is running normally and it does reach only the limit of the network bandwidth (10Gb/sec).

To prove this, in the case when 104 jobs run in parallel, the total amount of data that needs to be transferred from the SE to the Worker Nodes is around 104 GB and the average duration of each job is 98 sec. If the time spent to calculate the checksum is considered 0 sec (in reality it is around 3 sec << 98 sec), then the transfer speed of the Storage Element can be approximated as the total amount of required data, divided by the average execution time for each job (104GB/98 sec). It means the average transfer speed of the SE is around 1.1GB/sec, which is relatively close to the maximum theoretical speed that can be achieved on a 10Gb/sec network (1.25GB/sec).

To conclude, in the case when multiple jobs run in parallel and use the *lcg-cp* to access the same set of data from the Storage Element, the network bandwidth of the SE is the only limitation.

Once the current system (without using the SUD) is evaluated, the SUD is assessed and the results are compared with the previous test.

**Test 2**: Average execution time of jobs that are using the SUD (**1 Cache Node**)

The first test in this configuration reproduces the case when a number of parallel jobs are demanding the same file from the SUD, file that was not previously cached. In this case, the SUD downloads the file once from the Storage Element, writes it to the disk (or memory) and serves it over the HTTP protocol to the Worker Nodes.

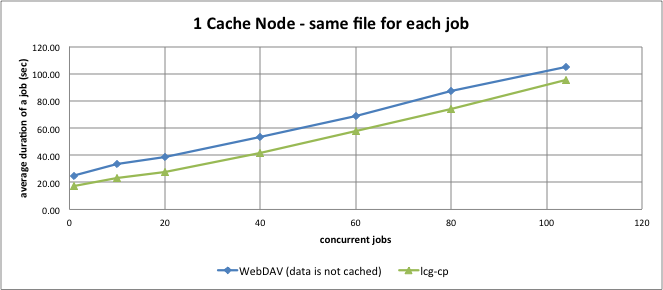


Figure – Comparison between average duration of jobs, those require the same file, and use the lcg-cp and the SUD that has 1 Cache Node (in total 24GB RAM) and the file is NOT cached

As expected, the graph above shows that the average execution time of the jobs that are using the SUD is higher than average execution time of the jobs that are not using SUD. The difference is approximately constant and it represents the time required to cache the file from the Storage Element (*around 13 seconds*).

The same test is executed again, but with the difference that the data is already cached by the Cache Node.

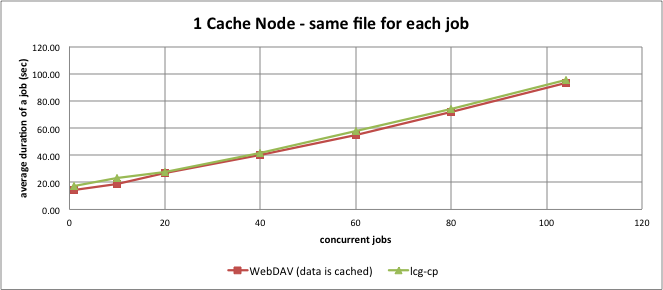


Figure – Comparison between average duration of jobs, those require the same file, and use the lcg-cp and the SUD that has 1 Cache Node (in total 24GB RAM) and the file is cached

As expected, due to the fact that both systems (SUD and SE) have the same network connectivity (10Gb/sec), the results are almost identical. The small difference in the favor of SUD shown in the graph above represents the overhead that the SRM interface introduces before starting to transfer the file (*< 7 seconds*).

In conclusion, the usage of SUD that is composed by only 1 Cache Node that has the same network connectivity as the Storage Element, does not seem to increase the overall performances of the system. Actually, for the case when the data is not cached, it is less efficient and the performances are worst than the *lcg-cp* usage.

**Test 3**: Average execution time of jobs that are using the SUD (**2 Cache Nodes**)

The number of Cached Nodes is increased from 1 up to 2 and all the previous tests are executed again. This test will determine whether there are any overall improvements, and what is the efficiency of the SUD if new resources are added.

Due to the fact that the SUD has more than 1 Cache Node, the load distribution policy of it becomes important. In fact, this policy is relatively simple but effective. The requests for data are sent randomly to the Cache Nodes. Because initially none of them have the required file cached, they both start to download it from the Storage Element. Once downloaded, both Cache Nodes can serve the file and then, the available overall network bandwidth of the SUD becomes **20Gb/sec** (2 Cache Nodes x 10Gb/sec).

The graph below shows the differences between the performances of the current system (lcg-cp) and the SUD that has 2 Cache Nodes.

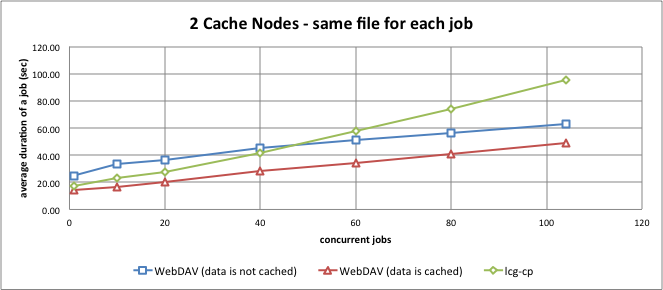


Figure – Comparison between average duration of jobs, those require the same file, and use the lcg-cp and the SUD that has 2 Cache Nodes (in total 48GB RAM)

In the graph above, it can be observed that the performances of the SUD are in general better than in the previous case (SUD with only 1 Cache Node), and at some point even better than current system (*lcg-cp*).

In the first case, when the SUD does not have the file cached, the average duration of jobs is longer than in the case *lcg-cp* in the first part of the graph, until 40 concurrent jobs. After this number, the performances of the SUD become better that the ones of lcg-cp*.* These results are expected, and they are explain by the fact that in the first part of the graph, the time required to download the file from the Storage Element to the Cache Nodes is relatively comparable with the time required to copy the file to the Worker Nodes directly. However, when the number of concurrent jobs exceeds **20**, the time required to download the file from the Storage Element to the Cache Node is the same, but because the amount of data is much higher and the SUD has a larger network bandwidth (20Gb/sec) compared with the Storage Element (10Gb/sec), the SUD performances improves compared with the SE. It overtakes the *lcg-cp* performances when the number of concurrent jobs exceeds 40.

Obviously, in the other case, when both Cache Nodes already have the file cached, the performances of the SUD are better than those of the *lcg-cp* in all tests. This is explained by the fact that the two Cache Nodes serve data to the Worker Nodes, each having 10Gb/sec network connectivity.

The relative improvement (regress) in performance, generated by the usage of the SUD (2 Cache Nodes), compared with the *lcg-cp* is calculated.

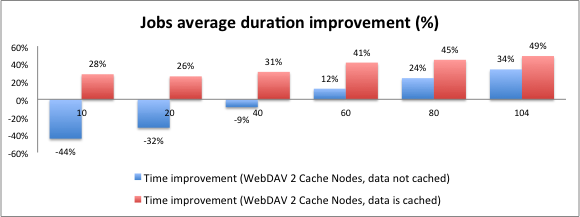


Figure – Job average duration improvement when SUD has 2 Cache Nodes

As the graph above shows, in the case the data is already cached by the Cache Nodes, the improvement of the performances is very high, reaching a top of 4**9%** (maximum 50%). In other words, a job can run almost twice faster that in the case of *lcg-cp*. However, in the other case, when the data is not already cached by the SUD, the average duration of a job is improved only when the number of concurrent jobs is higher than **40**. For less than **40** concurrent jobs, the SUD proves to be inefficient, and in the worst case it can slow down the performances with up to -44**%**, as compared with the *lcg-cp*.

From scalability perspective, the efficiency (as defined in the *Validation & Verification* chapter) of the system when the number of Cache Nodes is increased from **1** up to **2** in general reasonable high.

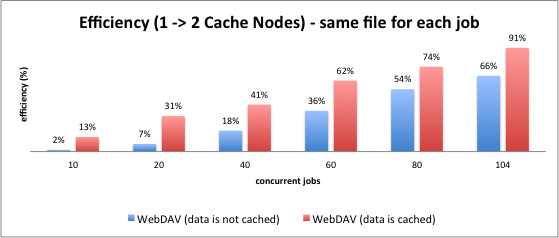


Figure – SUD efficiency when the number of Cache Nodes increases from 1 to 2

As the chart above shows, in the case of 10 concurrent jobs, the SUD composed by 2 Cache Nodes compared with the SUD composed by 1 Cache Node, increases the overall performances with 2% in case the date is not cached and **13%** in case the data is cached. However, when the number of concurrent jobs increases, the improvement increases as well. When 104 jobs run in parallel, the improvement reaches **66%** when the data is not cached and **91%** when the data is cached. Due to the fact the real value of scalable system shows up when the load is high, the results showed before are as expected and they are considered good. They prove that the SUD is scalable, and adding new resources to it, increases the performances.

To conclude, in the case of the SUD composed by two Cache Nodes, if the required file is used more than once (it is cached), the average duration time of the jobs can be reduced to half. Also, if the file is not previously cached, but multiple jobs demand it in the same time, again the SUD proves to improve the overall performances with up to one third. In the case when the file is not cached and there is not high demand for that particular file, the SUD decreases the overall performances with up to -44%.

**Test 4**: Average execution time of jobs that are using the SUD (**3 Cache Nodes**)

The number of Cached Nodes is increased from 2 up to 3 and all the previous tests are executed again. This test determines how much the overall performance of the system improves, while new resources are added to the SUD configuration.

In this case, the all three Cache Nodes that compose the SUD cache the required file, due to the load balancing policy (as defined in the previous test). This makes the SUD to have up to 30Gb/sec overall network bandwidth (3x10Gb/sec each Cache Node), which is superior to the Storage Element network bandwidth (20Gb/sec).

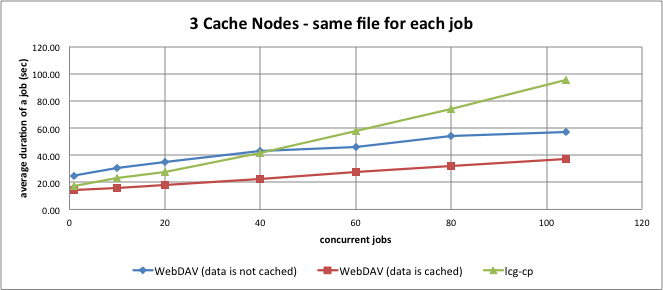


Figure – Comparison between average duration of jobs, those require the same file, and use the lcg-cp and the SUD that has 3 Cache Nodes (in total 72GB RAM)

As expected, the performances of the SUD composed by 3 Cache Nodes are better that the *lcg-cp* when the file is already cached during all the tests. However, for the case when the file is not cached, the same behavior as in the previous test happens. For a small number of jobs that run in parallel, the performances of the SUD are lower that the lcg-cp. Once the number exceeds 40 concurrent jobs, the performances of the SUD are higher and the increase of the average execution time of the jobs is much slower than the *lcg-cp*, while increasing the number of concurrent jobs.

The relative improvement (regress) in performance, generated by the usage of the SUD (3 Cache Nodes), compared with the *lcg-cp* is calculated.

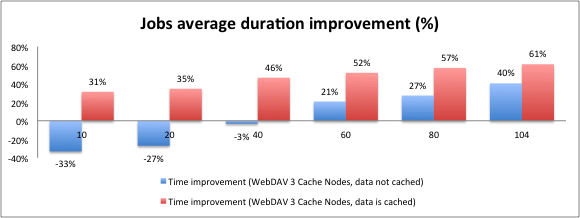


Figure – Job average duration improvement when SUD has 3 Cache Nodes

As the graph above shows, in the case the data is already cached by the Cache Nodes, the performance improvement is high, reducing the average execution time of the jobs with up to **61%** (maximum 66%). In other words, a job can run almost twice and a half faster that in the case of *lcg-cp*. However, in the other case, when the data is not already cached by the SUD, the average duration of a job is improved only when the number of concurrent jobs is higher than **40**, and the average execution time is reduced with up to **40%**. For less than **40** concurrent jobs, the SUD proves to be inefficient, and in the worst case it can slow down the performances with up to **-25%**, compared with lcg-cp. The behavior is similar with the case when SUD has 2 Cache Nodes.

From scalability perspective, the efficiency (as defined in the *Validation & Verification* chapter) of the system when the number of Cache Nodes is increased from **1** up to **3** in general high.

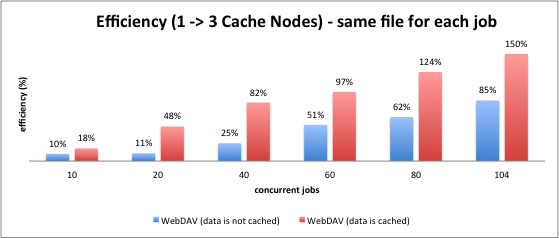


Figure – SUD efficiency when the number of Cache Nodes increases from 1 to 3

As the chart above shows, in the case of 10 concurrent jobs, the SUD composed by 3 Cache Nodes compared with the SUD composed by 1 Cache Node, increases the overall performances with **10%** in case the date is not cached and with around **20%** in case the data is cached. However, when the number of concurrent jobs increases, the improvement increases as well, and when 104 jobs run in parallel, the improvement reaches **85%** when the data is not cached and **150%** when the data is cached.

Another interesting comparison is the increase of the performances generated by the third Cache Node compared with the SUD with 2 Cache Nodes.

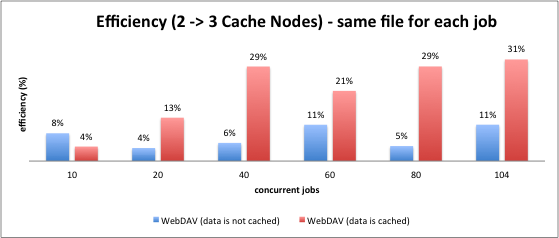


Figure – SUD efficiency when the number of Cache Nodes increases from 2 to 3

The chart above shows that using a third Cache Node in the tested case, the benefit introduced in the overall performances is only up to 31%. Probably for a higher load (i.e., more concurrent jobs), the improvement may become higher. However, due to the availability of resources, it was not possible to perform tests at higher load.

In the case of the SUD composed by three Cache Nodes, if the required file is used more than once (it is cached), the average duration time of the jobs can be reduced up to one third. Also, if the file is not previously cached, but multiple jobs demand it in the same time, again the SUD proves to improve the overall performances with around 70%. In the case when the file is not cached and there is not high demand for that particular file, the SUD decreases the overall performances with around one third.

To conclude, the results of the tests for the Scenario 1, when all the Worker Nodes require in the same time the same 1GB file from the SUD, shows that the overall performance of the system increases almost for all the tests. This proves that the SUD is a reliable and performant solution. Based on the usage statistics presented in the *Domain Analysis* chapter, around 30% of the files stored by a Storage Element are used more than once during a day. With reasonable large amount of space for caching, part of those 30% of the files can be theoretically be accessed faster if the SUD would be used.

**Scenario 2: All the jobs require different sets of data at the same time**

In the second scenario, the input file of each job is different, thus the hard disk transfer rate and the memory size become very important. Like in the previous scenario, the performance of the current system (without using the SUD) is measured and will serve as a comparison parameter (reference).

**Test 5:** Average execution time of jobs that are using *lcg-cp*

For this test, the size of each input file is 1GB and they are equally distributed across the two Disk Servers that compose the Storage Element.

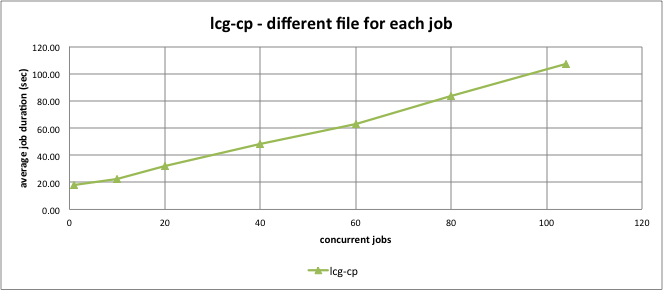


Figure – Average execution time of jobs that need different files and use lcg-cp

The graph shows again a linear increase of the average time needed by the jobs to complete, with the number of jobs that are running in parallel. This proves that the Storage Element is running normally and it does not reach its limit while running the tests. The same trend has been observed when using *lcg-cp* if the same file is requested by each job (*Test 1*).

However, compared with *Test 1 (lcg-cp)* from Scenario 1, the average time needed to complete the jobs is higher, even if now, the load per Disk Server is half compared to the other test. This behavior is highly related with the reading speed of the Disk Server’s hard disks. Unlike in the first scenario, where the entire amount of data needs to be read from the disk is the size of the file (1GB) and then kept in the memory, now a much higher amount of data is read from the disk. Each job demands a different file, so for each job the required file is read from the hard disks and kept in memory if the space allows.

In this test, the maximum number of jobs that are executed concurrently is 104, which means that each of the Disk Servers serves 57 jobs and around 57 GB of data. The approximate transfer rate of each of the disk is calculated as the total amount of data served by one disk server divided by the average duration of the jobs (again, as in the *Test 1*, the time spent from checksum is considered 0). The transfer rate would then be 57GB/105sec, which is **530MB/sec**. It is clear that when each of the jobs requires different file from the Storage Element, the network is not a problem anymore (like in *Test 1*), but the **hard disks** limits the entire performances. In other words, in this case a file is copied from the Storage Element more than two times slower.

In the case of the Storage Element, each of the Disk Servers has **72GB RAM** that theoretically allows the full storage of data required by the jobs in memory. However, this is not the case for the Cache Nodes that have **24GB RAM** each.

Next tests shows some interesting results, influenced by the better hardware specifications of the Disk Servers compared with the Cache Nodes.

**Test 6**: Average execution time of jobs that are using the SUD (**1 Cache Node**)

The first test in this configuration reproduces the case when a number of parallel jobs are each demanding different 1GB files from the SUD.

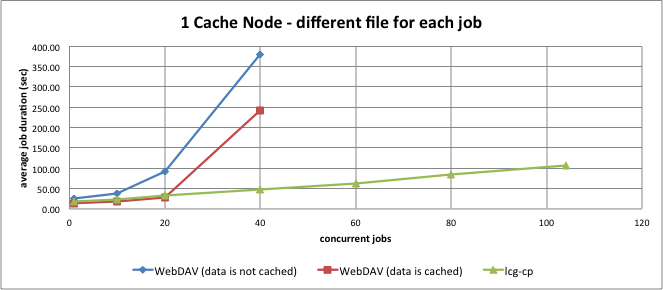


Figure – Comparison between average duration of jobs, that require different files, and use the lcg-cp and the SUD that has 1 Cache Nodes (in total 24 GB RAM)

The graph above shows for the first time a limitation of the SUD. In the case when the data is not previously cached, the average execution time of jobs that use SUD increases steeply between **1** and **20** concurrent jobs, up to **100 sec**, while the lcg-cp shows **32 sec**. More that **20** concurrent jobs make the average execution time to increase dramatically up to **400** seconds for **40** concurrent jobs, and after this limit the jobs fail (they take more than **600 sec**). This behavior is explained by the large amount of data that must be cached (wrote on the disk) by the SUD. When the number of concurrent jobs is up to **20**, the entire amount of data that is cached is around **20 GB**, and it fits in the Cache Node’s internal memory (**24 GB**). Once the amount of cached data exceeds the memory size of the Cache Node, it must be written directly on the disks, process that is extremely slow compared with writing in memory. This drives the dramatic increase of the job’s execution time.

In the other case, when the Cache Node already caches the files, obviously the performances of the SUD are better than in the case when they are not cached. However, compared with the lcg-cp, when the number of concurrent jobs is less than **20**, the average execution time of the jobs that uses the SUD are similar with the performances of the *lcg-cp*. But, when the number of concurrent jobs exceeds **20**, the average duration of the jobs increases dramatically up to **250 sec**, while the *lcg-cp* shows around **50 sec**.

This behavior is generated mainly by the two factors: *memory size* and the *hard disks* of the Cache Node. Firstly, the memory size of the Cache Node that must handle **104GB** of data is **24GB**, while the memory size of a Disk Server that needs to handle **57GB** of data is **74GB**. Secondly, unlike the Storage Element, which has two Disk Servers and each of them has multiple sets of hard disks; the SUD consists in only one Cache Node that has only one set of disks. More than this, the hard disk specification of the Disk Servers is superior compared with the Cache Node.

To conclude, in the case of high demand of varied data from the SUD that has only one Cache Node, the performances compared with a strong Storage Element, are extremely low if the amount of data demanded at a time exceeds the memory size of the Cache Node. However, if the amount of data demanded at a moment is less than the available memory and the data is cached, then the performances of the SUD are similar with the performances of the Storage Element (*lcg-cp* test).

**Test 7**: Average execution time of jobs that are using the SUD (**2 Cache Nodes**)

The number of Cached Nodes is increased from 1 up to 2 and all the previous tests are executed again. This test will determine whether are there any overall improvements, and what is the efficiency of the SUD if new resources are added.

According with the load balancing policy, the requests for data are distributed randomly to the Cache Nodes. Due to the fact that now all the jobs requires different files, each of the Cache Nodes caches only the required data (half of the entire set of input files). For the evaluation of SUD when the data is not previously cached, it is not important which of the Cache Nodes caches a file, as long as it is delivered to the Worker Node. In the other test, when the data is already cached by the SUD, the load balancing policy remains the same: the requests for data are sent randomly to the Cache Nodes, regardless the real location of the cached data. The request might end up to the wrong Cache Node that does not have the file cached. In this case, due to the SUD implementation, it redirects the user’s request to the Cache Node that has the file. All this time spent by the user to actually find the cached data is comprised by the test results.

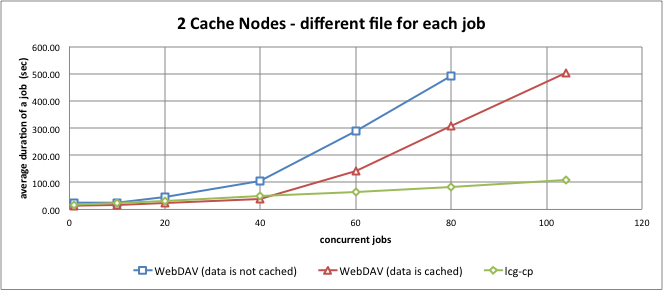


Figure – Comparison between average duration of jobs, that require different files, and use the lcg-cp and the SUD that has 2 Cache Nodes (in total 48GB RAM)

As expected, the results are better than in the previous case. Now the limit until the SUD has relatively good performances compared with the *lcg-cp* test increases up to **40** concurrent jobs. This is double than in the previous case, and the reason is that the overall available memory of the SUD increases from **24 GB** to **48 GB** (2 Cache Nodes x 24GB or memory). However, even if the number of concurrent jobs is less than 40, if the data is not cached by the SUD, the average execution time is higher, and it increases faster, than the *lcg-cp*. For 40 concurrent jobs using the SUD, the average duration is around **100 sec** when the data is not cached, and around **40 sec** when the data is cached. The last is comparable with the lcg-cp that shows **50 sec**. After **80** concurrent jobs, if the data is not cached by the SUD generates jobs failure.

The performances of the SUD, in the first part of the graph, when the data is cached are slightly better than the performances of the *lcg-cp* due to the fact that the data is kept in the memory of the Cache Nodes. The difference between them is almost constant (around 7-10 sec), and it is generated by the overhead introduced by the SRM interface of the Storage Element before start sending data. When the amount of data exceeds the size of the available memory, then the Cache Nodes must read the files from the disk, process that is slow compared with reading from memory.

The relative improvement (regress) in performance, generated by the usage of the SUD (2 Cache Nodes), compared with the *lcg-cp* is calculated.



Figure – Job average duration improvement when SUD has 2 Cache Nodes

As described in the figure above, the performances of the entire system are not increased drastically by the usage of the SUD with two Cache Nodes. It performs better than the *lcg-cp* only when the number of concurrent jobs is less than **40**, and the data is already cached (eventually in memory). The performance improvement in this case is around **+25%**. However, when the data is not cached by the SUD, the overall performances decrease with up to 480%, which makes the system extremely slow and inefficient.

**Test 8**: Average execution time of jobs that are using the SUD (**3 Cache Nodes**)

The number of Cache Nodes is increased from 2 up to 3 and all the previous tests are executed again. This test determines how much the overall performance of the system improves, while new resources are added to the SUD configuration.

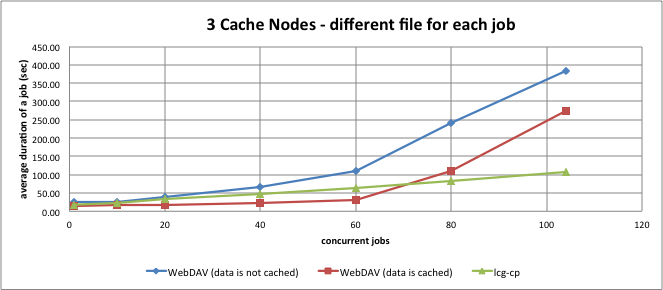


Figure – Comparison between average duration of jobs, that require different files, and use the lcg-cp and the SUD that has 3 Cache Nodes (in total 72GB RAM)

Again, the results above indicate that the SUD has good performances compared with the current system when the number of concurrent jobs is up to **60**. This is also explained by the increased overall memory of the SUD, up to **72 GB**. Over this limit, the SUD performs worse than *lcg-cp*, and the average duration of the jobs increases significantly up to **370 sec**, compared with the performances of the *lcg-cp* that are around **110 sec**.

The relative improvement (regress) in performance, generated by the usage of the SUD (3 Cache Nodes), compared with the *lcg-cp* is calculated.

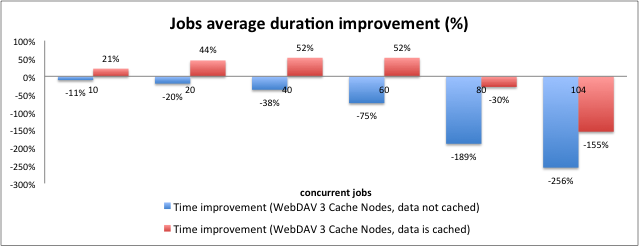


Figure – Job average duration improvement when SUD has 3 Cache Nodes

As in the previous test, the graph above highlights that better performances than the lcg-cp when the data is cached. In this case, the improvement of the average execution time of the jobs is up to 50%. This is possible due to the fact that now, three Cache Nodes that compose the SUD have a larger memory compared with the previous case (72 GB), and three sets of disks that read data in parallel. This configuration shows relatively good results even if the number of concurrent jobs exceeds 60. In the case the data is not cached, it performs worse than the *lcg-cp*, but compared with the previous configuration of the SUD, is still good. During this test, none of the jobs failed, that proves to be reliable enough to sustain up to 104 concurrent jobs.

**Test 9**: Average execution time of jobs that use the SUD (**1 Cache Nodes - 12 GB**)

The test tries to highlight the dependency between the memory of the SUD and its performances. To check whether the memory size is the reason, the test consists in reducing the memory of one Cache Node from 24 GB to 12 GB, and to execute again the *Test 6* (all jobs require different files from SUD that has one Cache Node).

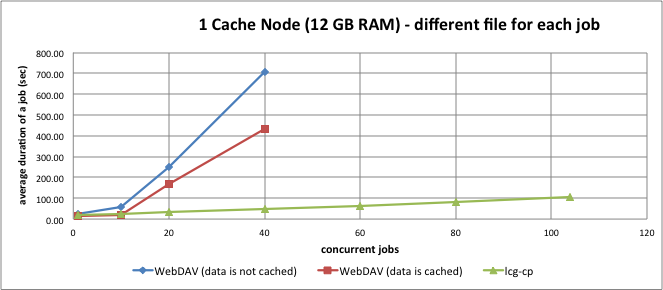


Figure – Comparison between average duration of jobs, that require different files, and use the lcg-cp and the SUD with 1 Cache Node that has only 12 GB of RAM

The result shows very clear that once the memory size of the Cache Node is reduced to half, the number of concurrent jobs that have relatively good results drops to **10**. This value is half of the number of concurrent jobs with good results for the case when 1 Cache Node (with 24 GB RAM) composes the SUD.

Now, it is clear that the performances of the SUD are closely related with the overall amount of available memory and the hard disks configuration of each of the Cache Nodes. Faster disks would probably increase the performances of the SUD when the amount of available memory is not sufficient.

To conclude, the SUD performs at least as good as the *lcg-cp* when the same set of data is used more than once (in a relatively short period of time, until the cache is flushed), and much worse as compared to the *lcg-cp* when the set of data is used only one time.

**Additional tests: encrypted data transfer and different input file sizes**

**Test 10**: Average execution time of jobs that use the SUD and encrypt the data traffic

The test simulates the case when multiple jobs demand the same 1GB file from the SUD with 1 and 2 Cache Nodes and the data is transferred encrypted. In this case, because of the encryption, the CPUs of the Cache Nodes and Storage Elements are highly utilized, thus they become the bottleneck of the entire system.

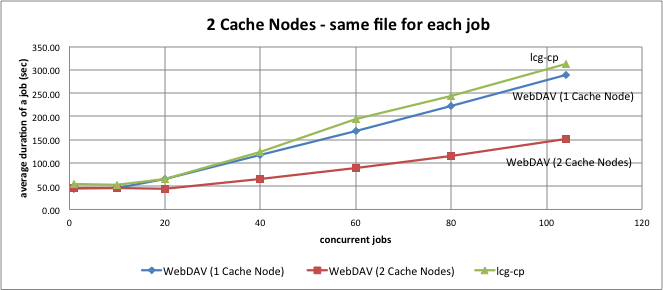


Figure – Comparison between the average duration of a job that uses the lcg-cp and the SUD with 1 and 2 Cache Nodes when the data transfer is encrypted (HTTPS)

As expected, the figure above shows good performances of the SUD compared with the *lcg-cp* when the data transfer is encrypted, same as in the case of unencrypted traffic. However, the average duration time for the jobs when that data is encrypted is more than **3** times higher than in the case when data is unencrypted, for both the SUD and the *lcg-cp*. This is caused exclusively by the huge overhead that is introduced by the encryption process, which is very CPU-intensive. The performances of the SUD with 2 Cache Nodes are twice as good as *lcg-cp* or SUD with 1 Cache Node, due to the higher number of cores that are used to encrypt data (2x8=**16**). In this case the network bandwidth is not the bottleneck anymore. Probably the usage of weaker algorithms for encryption would increase the performances, but with the cost of data security.

As a remark, the CPU usage for a Cache Node, in the case when the data transferred is unencrypted does not exceed **20%**, whereas it goes up to **100%** when data is encrypted.

**Test 11:** Average execution time of jobs that use the SUD and the file size is **100MB**.

Once the size of the file is reduced up to 100MB, then for 104 jobs the requested amount of data is around 10GB, instant amount that can easily be handled by 1 Cache Node. The test is performed and the results are presented above.

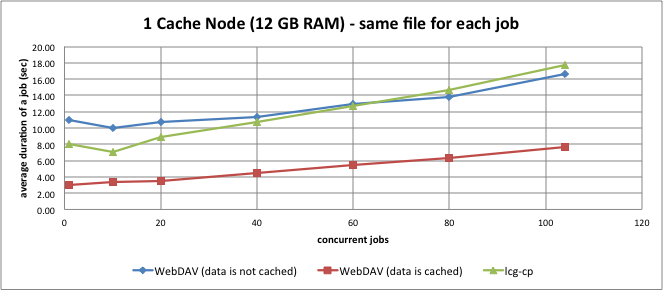


Figure – Comparison between average duration of jobs that require the same 100MB file and use the lcg-cp and the SUD with 1 Cache Node

The results show that even if the file is not previously cached, the SUD has relatively good performances and it gets even better than the lcg-cp when more than 60 concurrent jobs run. The reason for this is that once the network connectivity is not a problem anymore, the overhead introduced by lcg-cp and the SUD becomes important. The results prove that the SUD introduce a lower overhead when the data is copied. If the file is already cached, the results of the SUD are much better than the lcg-cp even if only 1 Cache Node composes the SUD.

The test is performed again with the difference that now all the jobs require different 100MB files, and the results is presented below.

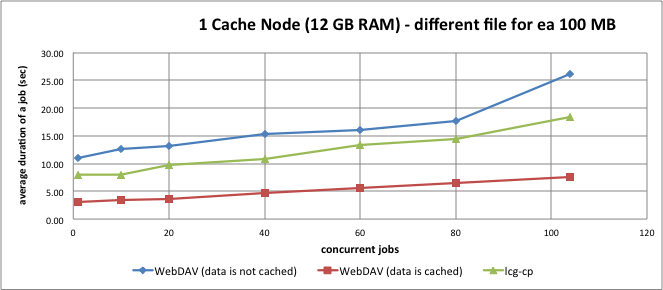


Figure – Comparison between average duration of jobs that require different 100MB files and use the lcg-cp and the SUD with 1 Cache Node

In the case when all the files are different, and they are not already cached by the SUD, the average duration of a job is, as expected, higher that the lcg-cp. However, when the files are cached, the SUD with 1 Cache Node performs much better (more that twice faster) than lcg-cp.

This proves again that once there is enough memory to accommodate all the data required at a moment, the SUD performs reasonably well and the jobs do not fail anymore even if they require different files.

To conclude, SUD proves to be extremely reliable and performant when the files are relatively small. The low overhead introduced makes it much more performant than the current system (lcg-cp).

### Robustness

The robustness evaluation of the current project when the load is high and the SUD reaches its physical limits, as explained in the previous chapter, is carried out while running the performance tests. It is defined as the number of failed jobs caused by the SUD utilization, compared with the current system (*lcg-cp* – SUD is not used).

The results show that depending on which scenario from the performance evaluation is tested, the robustness of the current project differs, as follows:

* In the first scenario, where all the jobs use the same file, the network bandwidth and the CPU (in the case of encrypted data transfer) are fully used, thus the physical limit is reached. It might happen sometime that some jobs fail, due to some unexpected errors (Segmentation fault) that are thrown by the gLite middleware (as explained in the feasibility study) randomly. The percentage is rather low (less than 5%), and it occurs only when the permission of the user to access a file is checked against the SE. However, if the *memory-cache* is active, and the permissions to the files are already cached by the SUD, then the number of jobs that fail is 0% (zero). The number of jobs that failed while using the grid’s tools (lcg-cp) is 0% (zero).

To conclude, in this first scenario, SUD proves to be a robust solution, which does not increase the jobs failure rate (as long as the gLite framework does not fail).

* In the second scenario, where all the jobs use all different files, the available memory (on the Cache Nodes), and the network bandwidth are fully used. In this case, the failure rate of jobs that use the SUD is high. The problems occur when the available memory of the Cache Nodes is fully used, as presented in the previous sub-chapter. Probably the jobs would have succeeded to finish but the time required would have been too long. A job is considered failed when the period between the time when job starts and the time when the SUD start sending data exceeds 10 minutes. The failure rate generated by the SUD varies between 0% and 80%, depending on the number of Cache Nodes that are used. However, the high failure rate is explained by the undersized hardware configuration where the Cache Nodes are deployed. The number of jobs that failed while using the grid’s tools (lcg-cp) is 0% (zero).

To conclude, in the second scenario, the SUD proves to increase the failure rate of the jobs with up to **80%**, but this is only due to the hardware limitations. In the case when the SUD has 3 Cache Nodes, none of the job fails, thus the failure rate is 0% (zero).

The other tests, where the robustness is evaluated as the SUD capability to recover from unexpected errors and situations, are all successfully passed. As a mention, during the performance tests, the SUD did never require an application restart or a machine reboot.

#### 

# Conclusions

*This chapter discusses the extent to which the goals of the project have been met. It presents in the first part the main conclusions according with the results from the previous chapter. In the second part, a set of future recommendations to improve the robustness and the performances of the current project are presented.*

## Conclusions

The main goal of current project is to provide a user-friendly and efficient alternative to access data that resides on the grid. The goal has been achieved and a WebDAV interface is developed and integrated with the grid technologies as part of the current project. The solution allows users to access the data from any operating system by using the web interface, or by mounting the grid’s data storages to their local computers. This increases the palette of applications that could be used for analyzing or processing, in the same time reducing the time required to reach the data. In order to access data, the user does not need to install any standard or grid-specific frameworks/tools, except the Grid Proxy Manager plugin for Firefox used for credential delegation.

To integrate the current components of the grid with the new solution, and in particular to access data from any Worker Node environment, proves to be an easy process. There is no need to install any single application or library on the Worker Node, in order to access data by making use of the current project. All the prerequisites can be easily shipped along with the jobs (as input sandbox). The only required modification is the addition of the user that sends jobs to the *fuse* group, which is an administrative decision.

The security of data is ensured, and the current solution provides the same authorization rules to access data as the storage elements of the grid currently provide. To improve the performances, trade-offs can be done, but they are completely configurable and they can be applied when needed.

A secondary goal of the project is to reduce the traffic of data between various components of the grid, especially between different sites which by definition is considered slow. This goal has been achieved as well and the current solution designs and implements a distributed caching mechanism that stores, for relatively short period of time, the data that is most used. In other words, once a file is copied from another site (by making use of the WebDAV interface), it is cached at the local site. Next time when the file is needed, it will be served directly from the (local) cache, much faster.

The extensive testing of the current solution, presented in Chapter 7 and Chapter 8, aims to determine the performance of the implemented project. In a worst case scenario the required data is located on the local storage of the site that needs it (by definition this is fast). The results show a rather small decrease in performance when the file is accessed the first time. This is normal behavior caused by the integration between the grid and standard technologies. When the file is already cached by the project, then performance is (highly) improved, depending only on the hardware configuration where the caching mechanism is deployed. In the other cases when the required data is located on a different site, the access time is even further reduced.

The design and the implementation of the current solution prove to be efficient in the sense that the SUD provides the maximum theoretical performances achievable with the resources provided by the deployment environment (such as network connectivity, CPU or hard disks). The design is highly scalable (both horizontally and vertically) and more resources can be added to the deployment environment with minimum integration effort. The implementation of the current project follows the model of so called Content Delivery Networks, which are the most efficient mechanism to serve large amounts of data as fast as possible.

Mostly, the implementation uses only unmodified (grid / standard) technologies, thus the possibility to use the most recent technological updates and the easy maintenance are ensured. However, some minor changes are required to some of the tools: the MyProxy Apache2 module and the wdfs WebDAV client.

The high-level requirements for design and implementation of the current solution were derived in Chapter 4. They are summarized below together with an evaluation of their achievement.

Table – High level requirements achievement

|  |  |  |
| --- | --- | --- |
| *Functional requirement* | *Achieved* | *Comments* |
| **FR-1**: The SUD must allow user to mount the files and directories defined on LFC and SE | Yes | - |
| **FR- 2**: The SUD must support PKI and username-password authentication mechanism | Yes | - |
| **FR- 3**: The SUD must preserve the same authorization rules to access files as defined on the SE. | Yes | - |
| **FR- 4**: The SUD must store (cache) the files that are used, for an undetermined time period, as long as space permits | Yes | - |
| *Nonfunctional requirement* | *Achieved* | *Comments* |
| **NFR- 1**: The SUD must provide a uniform and transparent access to data from user perspective | *Yes* | *-* |
| **NFR- 2**: The SUD should use as much as possible standard technologies | *Yes* | *-* |
| **NFR- 3**: The SUD should be scalable and additional resources can be easily added | *Yes* | *-* |
| **NFR- 4**: The changes required on the already existing Grid components must be minimum | *Partially* | The Worker Nodes need to add the user that executes jobs in the *fuse* group. |
| **NFR- 5**: The number of tools that the user needs to install in order to connect to the SUD should be 0. | *Yes* |  |
| **NFR- 6**: Failure rate of jobs that use the SUD, caused exclusively by failures of the SUD, should be less than 2% | *Partially* | When the SUD is deployed on machines that do not have enough resources, the failure rate of the jobs is rather high. |
| **NFR- 7**: The total execution time of a job that uses SUD should be less than the current system | *Partially* | When the files are already cached by the SUD then this requirement is achieved. However, when the files are not cached, and they need to be copied from the Storage Elements, the execution time of jobs is higher that the current system. |

## Recommendation

The critical factor for achieving better performances is the environment where the current solution is deployed, and in particular the memory size, the network connectivity, and the hard disk size and configuration of the deployment machines. It was proved that using machines with less memory available than the maximum required data the performance temporarily drops dramatically, sometimes causing job failure.

It is desirable to use the results of the project in situations when the same data is utilized more that once, in order to make use of the caching mechanism and the low overhead introduced. Making use of the project results when data is not used more than once proved to be inefficient. However, the results of the project can easily be used anytime the usability is important and the small decrease of the performance is not a major issue.

## Future work

During the project several ideas have come up on what further improvements can increase the usability and the performances of the developed solution. Some of them are described in the following paragraphs; they were considered to be outside the scope of this project because of lack of time and the foreseen complexity of the underlying work.

* *Write functionality* – it would be desirable that not only reading data must be a user-friendly process, but also writing. Once the storage elements of the grid are mounted to a computer, the user should be allowed to write and upload files directly to the Storage Elements.
* *Integration with other File Catalogs* – other grid file catalogs (such as ROOT[[35]](#footnote-35)) can publish the file throughout the WebDAV protocol implemented by the current project. By providing access to all types file catalogs provided by the grid, the current project would become the main standard gateway to all the data stored by the grid.
* *Eliminate the GSI proxy certificate* – during the feasibility study several problems were encountered while using the X.509 GSI proxy certificates. They are not entirely following any standard, thus none of the open technologies investigated were able to use them. Because the standard X.509 proxy certificates are supported as well by the grid technologies, the impact of eliminating the GSI proxies would be very limited.
* *HTTPS interface provided by the Storage Elements* – currently to transfer data from Storage Elements several protocols are implemented: RFIO, GSIFTP, etc. All these are grid-specific protocols cannot be integrated with any already existing standard tools. It would be desirable that along with the grid-specific transfer protocols, the HTTPS protocol would be provided as well. By providing a standard transfer protocol, a large palette of tools would be able to access data directly from the Storage Elements. Moreover, the current project would perform much better in the case the data is not cached, because even so, the Cache Nodes can act as proxies, thus the data will be transferred to the client in one step, and not two as it is implemented now (first copy the data from Storage Element to cache – GSIFTP, and then serve it to the client – HTTP [S]).

# Project Management

*The chapter introduces various issues relevant to project management. The process that was used to manage the project is described in the first part. Other related subjects like Breakdown structure, Milestone Trend Analysis, and Risk management are also presented in this section. A short retrospective of the project encloses the chapter.*

## *Process*

The current assignment was mainly a research project where the number of unknowns was high. The grid-computing domain was new for the author when the project started, thus research had to be done in order to realize the magnitude of the assignment. The management of this project aimed to be simple, in order to have a minimum overhead. Because only one person (the author) carried out the project and the stakeholders imposed no intermediary deadlines, the planning was not rigid and changes took place during the project.

The project planning followed an iterative approach with several iterations called sprints. Each of the iterations was defined in terms of deliverables and activities that need to be done in order to achieve the goals of the sprint. After the completion of every sprint, the resulted deliverables were compared against what was planned, and new actions were taken.

Project Steering Group (PSG) meetings were usually held once per month through- out the project. The purpose of these meetings was to inform the attendants about the status/progress of the project and to obtain feedback from supervisors. The output of PSG meetings was archived and used all the way through the project.

The communication within the company was rather informal and not many meetings were held with all of the supervisors, due to their busy schedule, and probably due to the lack of initiative of the author. However, an important part of the communication was carried out via email, where the problems encountered and the progress of the project, were presented. All these were sufficient to ensure that the project is on the right track and the goals of the stakeholders are the achieved.

## Breakdown structure

While carrying out almost any project, several steps need to be done: gathering the requirements, feasibility study, design, implementation, deployment, and testing. Due to the nature of the assignment (research-oriented and proof of concept), almost all the phases of the project were important as individual for the main stakeholder. The sprints are defined and they match all the phases of the project. While defining the sprints the deadlines and the deliverables are specified as well.

Table – List of all iterations, deadlines and deliverables

|  |  |  |  |
| --- | --- | --- | --- |
| *Iteration* | *Deadline* | *Deliverables* | *Motivation* |
| Domain analysis | 28/01/2011 | Knowledge | It was important for the author |
| Requirements | 21/02/2011 | Survey and  Requirements list | The outcome from the survey carried out is valuable for stakeholders and for the project |
| Feasibility study | 11/04/2011 | Prototypes | Ideas and reasons why to [not] use a design approach or a tool are important for Nikhef |
| Design | 02/05/2011 | Architecture | The decisions and motivations for the final design are important for all stakeholders |
| Implementation | 18/07/2011 | Prototype that provides entire functionality | This proves that the entire design and feasibility study is correct |
| Deployment and testing | 25/07/2011 | Deployment process + the results of using the SUD | This proves that the current project can be integrated into the production environment. It also checks the efficiency and performances of the SUD |
| Final report delivery | 05/09/2011 | This report | The ideas and findings will guide a further extension of the SUD |

However, because it was an exploratory project where the number of options and possible solutions that could have been investigated was high, the deadlines of the phases were relatively strict. At the end of each sprint decisions and trade-offs were made, because otherwise the project could not have been finished on time. After each of the iterations, fine-grained deliverables and deadlines were defined for the next ones based on the findings at that moment. A timeline chart was used to define the sprints and it is presented below.

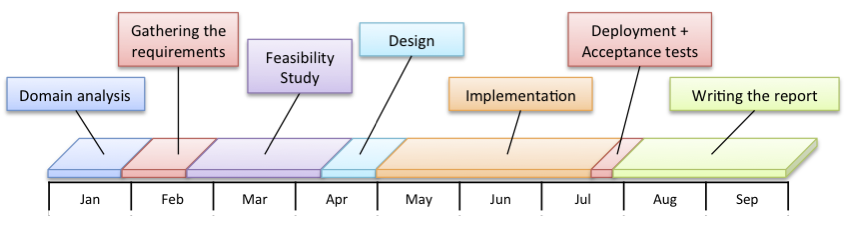


Figure – Planned timeline of the project

For a more detailed plan with all the deliverables, the reader can refer to [2].

## Milestone Trend Analysis

Although horizontal timeline chart was defined for the entire project (Figure 81) a Milestone Trend Analysis chars was created as well in order to keep a good track of deadlines for the important artifacts and deliverables, and it is presented below.

As it can be observed in Figure 82 the project suffered some deviations from the initial plan, especially at the beginning, when the requirements were defined. The cause for those deviations was that the initial requirements for the project were vague, and the process to elucidate and to formally define them was inefficient and the communication with the stakeholders was not ideal. A survey among the potential users of the current project needed to be performed in order to determine what exactly is required.

Fortunately, once the requirements are defined (around end of February), the project progress was quite straightforward and no significant deviations took pace. The most important deadlines that have changes are:

* Use-case definition – postponed from mid-February until mid-March. It was caused by the late elucidation of the requirements.
* Definition of the test plan - postponed from the beginning of May until the beginning of June. This is due to an unplanned holiday (for 1 week) that the author took.
* Deployment and testing – postponed with around two weeks. The reason is that more tests are performed with more resources than planned in order to correctly evaluate the performances of the project.

The design of the system was defined on time, at the end of April. Because of the high-level functional requirements, the implementation of the current project was done in three steps (two prototypes and one final release), which were delivered on time. Verification and validation was performed after each delivered prototype.

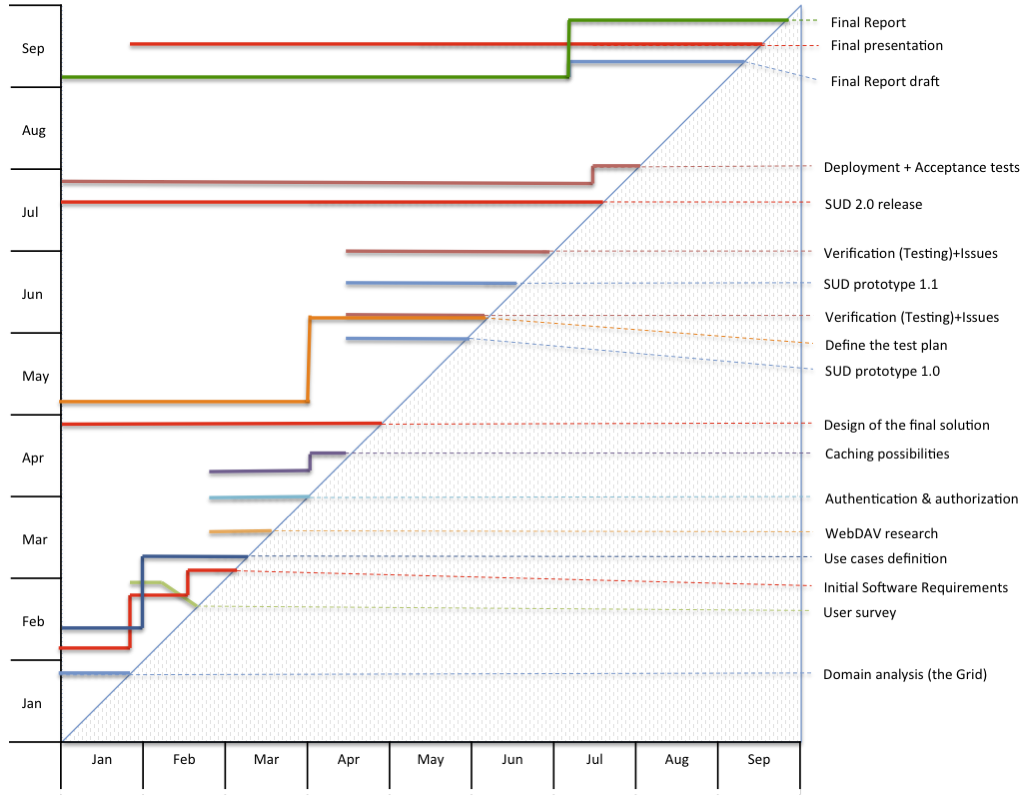


Figure – Milestone Trend Analysis chart

All the phases of the project (except the writing of this report) were fully completed at the end of July and all the functional requirements were delivered. The current report was written in the last two months of the project.

The used management practices have been proven successful for this project. All the functionality required was provided on time. More than that, the time planned for testing was extended in order to have a better view of the performances of the current project, even if this would not have been mandatory.

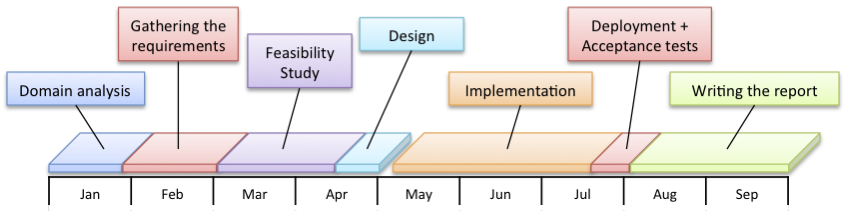


Figure – Real timeline of the project

If the initial timeline (Figure 81) is compared with the figure above (the real timeline) it is obvious that there are no major differences. This proves, that the initial planning was accurate and the project followed it almost entirely.

The small planning deviations from the beginning of the project probably could have been avoided if the communication with the stakeholders would have been more intense. However, the good practices defined on the risk mitigation were applied and the problems were solved relatively quickly.

## Risk management

Because the feasibility study was one of the most significant parts of the entire project, it tried to identify all the risks and problems that can occur during the implementation and deployment phases. The risks and problems were immediately communicated to the responsible stakeholders, and their mitigation was defined accordingly. Some of the identified risks which had an impact on the course of the project, accompanied by the mitigation strategies and experiences are listed in the table below.

Table – Most important risks identified during the project

|  |  |
| --- | --- |
| *Risk* | *Description and mitigation* |
| Learning curve | The grid technologies were new for the author thus during such a short project the learning curve is desirable to be quite steep. Even if it is, there is always the risk to go in the wrong direction. For this reason it is essential to summarize the findings to the supervisors. |
| Extracurricular activities | Interviews and other activities important for the author in long term can delay the deliverables deadline. It is important to allocate extra time from the beginning of the planning for these activities. |
| Summer vacations | During the summer, when this report aims to be written, many of the supervisors take days of for holidays. This would reduce the frequency of the needed feedbacks in order to write a good quality report. To mitigate this risk it is desirable to start writing the report from an early stage of the project. |
| Defining the right requirements | In the first part of the project the initial requirements were vague. Due to the lack of communication with the company supervisors their elicitation can be difficult. To mitigate this problem other stakeholders should be investigated, and mockups need to be presented anytime possible to the main stakeholders. |
| Communication with supervisors | Because of language barrier and background differences there might be misunderstandings between the author and stakeholders. The use of meeting minutes and small demonstrators (when possible) are good practices. |

## Project retrospective

Looking back to the project, most of the parts worked out well while others could have been improved. The following list gives an overview, including the positive experiences as well as lessons learned.

* Extensive domain and feasibility analysis is important especially when the project is research-oriented. Discuss the options and possible solutions with the colleagues and supervisors and use the feedback to improve the solution.
* Finding experts within the organizational environment is very important. They can help to find potential problems earlier and not during the technical implementation. They proved extremely supportive and helpful for this project.
* Logging activities will help to keep track of the progress of the project, and will give to the supervisors a clear idea about how the project is handled. However, the weekly logging was not done during the entire project, but it would have been beneficial.
* Working in a structured way can highly improve the productivity. When the project is in early stage and many things are not yet clear, it is very important to involve as many persons as possible to have technical discussions.
* Writing the final report should be a continuous process during the entire project. It should be taken into account that during the summer, most of the supervisors are having holidays and the time for reviewing is very limited.

While designing a project, there are ten technological design criteria that can be taken into account: functionality, impact, possibility of realization, inventiveness, complexity, elegance, genericity, methodical approach, and documentation. Not all these criteria are applicable for all the projects, or even further their importance varies.

However, three criteria have been chosen as the most important for this project. A brief description of how they were achieved is presented below:

* Functionality – This project started as an idea without a fixed goal. It was very important to determine how that idea can be extended and what can be achieved. A set of functionalities that would be desirable was defined during the first phase of the project, with the help of scientists that currently are using the grid. The research focused on how the functionalities can be achieved and the designed tried to accommodate all of them. At the end of the project everything that was required was achieved.
* Possibility of realization – The project was research-oriented and it started with some vague requirements. It was not known whether it is possible to be implemented and what are the costs. The feasibility study was the most important phase of the project and it tried to discover if and how it is possible to implement the defined functionality. The design focused on the possibility to combine the findings from the feasibility study in order to provide the required functionality.

* Genericity – The design combines as much as possible standard technologies and tools that prove to be reliable and efficient. The idea of the implemented solution is to provide something simple, easy to maintain and performant without “reinventing the wheel” if possible. In the other hand, the design provides modules and components that can easily be reused in further developments. During the feasibility study, and the design elaboration a set of practices (possible solutions) were identified during the implementation of several prototypes, and during the research of several already existing similar solutions. The design uses the identified (best) practices.

The two out of the ten design criteria that are less important are:

* Inventiveness – The project tried to not reinvent the wheel and many standard tools and technologies are used. The design is inspired by different success-proof content delivery designs, thus the inventiveness of the implemented solution and its design is rather low.
* Elegance – The project aimed to be rather a proof-of-concept. For this reason, the elegance of the design was not considered very important criteria. However, the design of the implemented solution is extensible with well-defined layers and components.

Glossary

|  |  |
| --- | --- |
| **AB** | Apache Benchmark |
| **API** | Application Programming Interface |
| **BDII** | Berkley Database Information Index, the Information Service used in gLite |
| **CA** | Certification Authority |
| **CDN** | Content Delivery Network |
| **CE** | Computing Element |
| **CERN** | European Organization for Nuclear Research |
| **CGI** | Common Gateway Interface |
| **CPU** | Central Processing Unit |
| **DMS** | Data Management System |
| **DN** | Distinguished Name |
| **EGI** | European Grid Initiative |
| **FCGI** | Fast Common Gateway Interface |
| **FTP** | File Transfer Protocol |
| **FUSE** | File system in User space |
| **GFAL** | Grid File Access Library |
| **GSI** | Grid Security Infrastructure |
| **GSIFTP** | GSI File Transfer Protocol |
| **GUI** | Graphic User Interface |
| **HTML** | Hypertext Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Secure Hypertext Transfer Protocol |
| **JDL** | Job Description Language |
| **LCG** | LHC Computing Grid |
| **LFC** | LCG File Catalog |
| **LFN** | Logical File Name |
| **LHC** | Large Hadron Collider (high-energy particle accelerator at the CERN laboratory) |
| **MSS** | Mass Storage Systems |
| **PDP** | Physics Data Processing Group |
| **PKI** | Public Key Infrastructure |
| **POSIX** | Portable Operating System Interface for Unix |
| **PSG** | Project Steering Group |
| **RFC** | Request For Comments |
| **RFIO** | Remote File Input/Output |
| **SE** | Storage Element |
| **SRM** | Storage Resource Manager |
| **SSL** | Secure Sockets Layer |
| **SUD** | System Under Development |
| **SURL** | Storage Uniform Resource Locator |
| **UI** | User Interface |
| **URL** | Uniform Resource Locator |
| **VO** | Virtual Organization |
| **VOMS** | Virtual Organization Membership Service |
| **WebDAV** | Web-based Distributed Authoring and Versioning |
| **WLCG** | Worldwide LHC Computing Grid |
| **WMS** | Workload Management System |
| **WN** | Worker Node |
| **XML** | Extensible Markup Language |

# Bibliography

[1] Cirstea, C. T. *Software Requirement Specifications.*

[2] Cirstea, C. T. *Project Management.*

[3] Cirstea, C. T. *Grid Usability Survey.*

[4] Ricciardi, G. M. (n.d.). *Usage examples of LFC Python API*. Retrieved from grid.ct.infn.it: <https://grid.ct.infn.it/twiki/bin/view/GILDA/LFCPythonAPI> on 05-09-2011

[5] (n.d.). Retrieved from http://wiki.egee-see.org/index.php/Data\_Access\_with\_GFAL on 05-09-2011

[6] Calanducci, T. (n.d.). *Usage of the lcg\_util APIs*. Retrieved from grid.ct.infn.it: https://grid.ct.infn.it/twiki/bin/view/GILDA/LcgUtilAPI on 06-09-2011

[7] Wikipedia. (n.d.). *Public Key Infrastructure*. Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Public\_key\_infrastructure on 30-08-2011

[8] Globus. (n.d.). *Overview of the Grid Security Infrastructure*. Retrieved from Globus: http://www.globus.org/security/overview.html on 02-09-2011

[9] Burke, S., Campana, S., Lanciotti, E., Lorenzo, P. M., Miccio, V., Nater, C., et al. (n.d.). *GLite User Manual*. Retrieved from CERN: https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.html on 05-09-2011

[10] Nikhef. (n.d.). *About Nikhef*. Retrieved from Nikhef.nl: <http://www.nikhef.nl/en/about-nikhef/> on 12-08-2011

[11] Cirstea, C. T. *Use-cases document.*

[12] *Wikipedia. (n.d.). WebDAV.* Retrieved from Wikipedia: <http://en.wikipedia.org/wiki/WebDAV> on 05-08-2011

[13] Dierks, T., & Rescorla, E. (n.d.). The Transport Layer Security (TLS) Protocol. Retrieved from The Internet Engineering Task Force: <http://www.ietf.org/rfc/rfc5246.txt> on 25-08-2011

[14] *Globus. (n.d.). MyProxy.* Retrieved from Globus.org: <http://globus.org/toolkit/docs/5.0/5.0.0/security/myproxy/rn/> on 05-08-2011

Appendix A: Job example

The table below shows an example of a job that can be sent to the Grid. The job connects to the SUD in order to access data, and calculates the checksum of a file.

Table – Simple example of job that uses the SUD

|  |
| --- |
| ***Job.jdl*** |
| Executable = "/bin/bash";  Arguments = "**wdfs.sh** **work.sh**";  Stdoutput = "stdout";  StdError = "stderror";  InputSandbox = {"**wdfs-redirect**", "**htproxyput**", "**wdfs.sh**","**work.sh**"};  OutputSandbox = {"stdout","stderror"};  OutputSandboxBaseDestURI = "gsiftp://tbn15.nikhef.nl/tmp/"; |
| ***wdfs.sh*** |
| #!/bin/bash  chmod 755 ./**wdfs**  chmod 755 ./**htproxyput**  chmod 755 $1  export WEBDAV=$PWD/webdav  old=$PWD  #delegate WN’s credentials to the SUD  ./**htproxyput** --cert $X509\_USER\_PROXY --key $X509\_USER\_PROXY https://tbn28.nikhef.nl:6443/delegation-service/delegate  mkdir $WEBDAV  #mounting the WebDAV repository  ./**wdfs-redirect** -f -o direct\_io -o kernel\_cache -o accept\_sslcert -o certchain=$X509\_USER\_PROXY -o privkey=$X509\_USER\_PROXY https://tbn28.nikhef.nl:8444/webdav/ $WEBDAV &  sleep 2  ./$1  **fusermount** -u -z $WEBDAV  sleep 5 |
| ***work.sh*** |
| #!/bin/bash  LFN=/grid/pvier/test\_cristian/real\_test/file$1  cksum $WEBDAV/$LFN |

The **wdfs.sh** is the executable that mounts the WebDAV repository to the Worker Node and execute the script that is passed as parameter (*work.sh*). It also delegates the user’s proxy certificate to the SUD by making use of *htproxyput* and unmounts the WebDAV repository when the job is finished.

The **work.sh** is the script that executes the required analysis/processing of an input file. Before using the SUD, this file would have been specified as the main executable. The *wdfs.sh* executes this file once the Worker Node is connected to the SUD.

The **job.jdl** is the file that describes the job and it is usually send to the Grid to be executed. It specifies the main executable (wdfs.sh) and it defines the list of files that are transferred to the Worker Node (*wdfs*, *htproxyput*, *wdfs.sh*, *work.sh*). The **wdfs-redirect** is the WebDAV client for Linux specially modified to accept GSI authentication and to support REDIRECT. The **htproxyput** is the tool that delegates the proxy certificate from the Worker Node to the SUD and it is part of the GridSite project.

Appendix B: Usage of htproxyput and wdfs-redirect

**Htproxyput** is a tool provided by the GridSite project that makes possible the X.509 (proxy) certificates delegation to another machine that provides gridsite-delegation.cgi.

|  |
| --- |
| ./**htproxyput** –cert CERTIFICATE –key PRIVATEKEY URL |

* URL points to the *gridsite-delegation.cgi*
* -cert represents the certificate file path
* -key represents the private key file path

To use the proxy certificates that are currently used to interact with the grid’s components, the –cert and the –key needs to point to the same file ($X509\_USER\_PORXY) that contains the proxy certificate.

However, in order to delegate a Limited Proxy Certificate that resides on the Worker Node, the *htproxyput* was modified. In the original version, the limited proxy certificate is delegated as a normal proxy certificate. In other words, if the DN of a limited proxy certificate would be for example *“/DC=org/DC=terena/…/O=Nikhef/CN=Cristian/CN=limited proxy“*, the DN of the delegated proxy certificate would be *“/DC=org/DC=terena/…/O=Nikhef/CN=Cristian/CN=limited proxy/CN=proxy“*. This will make the newly created proxy certificate invalid.

The changes made on the htproxyput correct this behavior. The new DN looks like: *“/DC=org/DC=terena/…/O=Nikhef/CN=Cristian/CN=limited proxy/CN=limited proxy“*

**Wdfs-redirect** is a modified version of the wdfs[[36]](#footnote-36) that permits REDIRECT and accepts the GSI proxy certificates as authentication method. It is the WebDAV client that is used by the Worker Nodes to connect to the SUD.

The original version of wdfs does not support PKI authentication or the REDIRECT functionality. Because these were mandatory for the project, it was modified and the missing functionality was implemented.

The WebDAV client (wdfs-redirect) starts by default as a daemon. In order to run on a Worker Node, it needs to be started as a foreground application (not as a daemon), thus the “-f” parameter needs to be added as in the following example. Otherwise, the Worker Node will kill the client’s process.

|  |
| --- |
| ./**wdfs-redirect** **-f** -o certchain=… -o privkey=… URL MOUNT\_POINT |

To use the proxy certificates that are currently used to interact with the grid’s components, the *–o certchain*and the *–o privkey* need to point to the same file ($X509\_USER\_PORXY) that contains the proxy certificate.

Appendix C: PROPFIND response (for Windows 7)

As explained in the report, the only WebDAV method that needs to be implemented by the current project is PROPFIND. In order to work with Windows 7, the xml nodes need to be in the exact order as presented below.

Table – Example of HTTP-response generated by the SUD when the server receives a PROPFIND request

|  |
| --- |
| *Response* |
| STATUS 207 Multistatus  <?xml version="1.0" encoding="utf-8"?>  <D:multistatus xmlns:D="DAV:">  <D:response xmlns:lp1="DAV:" xmlns:lp2="http://apache.org/dav/props/">  <D:href>/wd5/</D:href>  <D:propstat>  <D:prop>  <lp1:resourcetype><D:collection/></lp1:resourcetype>  <lp1:creationdate>2011-06-24T06:55:19Z</lp1:creationdate>  <lp1:getlastmodified>Thu, 23 Jun 2011 10:22:51 GMT</lp1:getlastmodified>  <lp1:getetag>"21fb59-1000-4a65e77a0b4c0"</lp1:getetag>  <D:supportedlock>  <D:lockentry>  <D:lockscope><D:exclusive/></D:lockscope>  <D:locktype><D:write/></D:locktype>  </D:lockentry>  <D:lockentry>  <D:lockscope><D:shared/></D:lockscope>  <D:locktype><D:write/></D:locktype>  </D:lockentry>  </D:supportedlock>  <D:getcontenttype>httpd/unix-directory</D:getcontenttype>  </D:prop>  <D:status>HTTP/1.1 200 OK</D:status>  </D:propstat>  </D:response>  </D:multistatus> |

Appendix D: Windows XP & 7 WebDAV Client required changes

As presented in the previous chapters, the WebDAV clients provided by the Windows XP and Windows 7 are not able to connect to the SUD directly (or to any WebDAV repository that provides Basic Authentication). In order to fix the issue, the Windows registry needs to be changed as follows.

1. Open the Window Registry (Start -> Run and type *regedit* command)
2. Set the following registry key value to 1 or 2.

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\WebClient\Parameters\BasicAuthLevel

The BasicAuthLevel can be set to the following values:

* 0 - Basic authentication disabled
* 1 - Basic authentication enabled for SSL shares only
* 2 or greater - Basic authentication enabled for SSL shares and for non-SSL shares

1. Reboot the computer

Appendix E: mod\_authn\_myproxy required changes and compilation

The Apache2 module *mod\_authn\_myproxy* integrates the MyProxy credential repository with the web server.

Compilation steps:

1. Download the source code from <http://grid.ncsa.illinois.edu/myproxy/apache/>
2. Install apxs application provided by Apache2.

|  |
| --- |
| yum install httpd-devel |

1. Add to the *authn\_myproxy\_authenticate* method the following code.

|  |
| --- |
| int i=0;  for (i=0; i<strlen(user); i++)  if (user[i]==VO\_DELIMITER) {  apr\_table\_set(r->subprocess\_env, "USER\_VO",strdup(user+i+1));  break;  } |

1. Compile the source code

|  |
| --- |
| apxs -c -i -I /usr/lib64/globus/include/ -I /usr/include/globus/ -L /usr/lib64 -l myproxy mod\_authn\_myproxy.c |

1. The compiled module is saved at /usr/lib64/httpd/modules/mod\_authn\_myproxy.so
2. Copy the module to /etc/httpd/modules/

Appendix F: List of problems identified during the project

Table – Problems identified while using GridSite tool

|  |  |
| --- | --- |
| *#* | *Problem & Solution* |
| 1 | **The *htproxyput* does not delegate correctly a Limited GSI proxy certificate. It does not make difference between GSI proxy certificate and Limited GSI proxy certificate, thus it adds all the time to the DN the “CN=proxy”, instead of “CN=limited proxy”.** |
| *Solution* | The *htproxyput* needs to be modified in order to make the difference between the two types of GSI proxy certificates, and add accordingly to the DN the right type of the proxy: “CN=proxy” or “CN=limited proxy” |
| 2 | **The *gridsite-delegation.cgi* module does not handle correctly concurrent delegations of the same (proxy) certificate. The problem occurs when multiple concurrent jobs delegate in the same time the same proxy certificate.** |
| *Solution* | Some sort of synchronization need to be added to the delegation module in order to make the delegation an atomic operation. |

Table – Problems identified while using MyProxy

|  |  |
| --- | --- |
| 1 | **The *Grid Proxy Manager* plugin does not allow user to specify the VO (VOMS attributes).** |
| *Solution* | The certificate is uploaded to MyProxy Server without VOMS attributes. The SUD needs to add the attributes when the certificate is received from the MyProxy Server.  For Apache2: the *mod\_authn\_myproxy* module needs to be modified. |
| 2 | **The *mod\_authn\_myproxy* Apache2 module, once loaded by the web server will not allow to other CGI application to run properly. The user will receive the “Premature end of script” message when he/she tries to access the CGI.** |
| *Solution* | The source code of the *mod\_authn\_myproxy* needs to be modified as follows:  The error comes out of *globus\_module\_deactivate\_all ()* method. If the function is removed from the *deinitialize\_module\_child (void \*s)* method then everything works as expected. |

Table – Problems identified while using the gLite framework

|  |  |
| --- | --- |
| 1 | **The file path of the proxy certificate MUST not contain % character. Otherwise *Segmentation fault* error is generated.** |
| *Solution* | The “%” needs to be replaced with “%%” everywhere it is found in the path.  Reason: The error is generated due to the fact that the path to the proxy certificate file is passed as the format parameter of sprintf function, in the gLite middleware. |
| 2 | **The GFAL and LFC libraries cannot be used together in the same process, because *Segmentation fault* error is generated.** |
| *Solution* | Unknown |
| 3 | **The GFAL and LCG\_UTIL libraries cannot be used together in the same process, because *Segmentation fault* error is generated.** |
| *Solution* | Unknown |

Table – Problems identified while using WebDAV

|  |  |
| --- | --- |
| 1 | **Some of the WebDAV clients does not support REDIRECT** |
| *Solution* | For Linux, there are several open source WebDAV clients that can be modified.  *Wdfs-redirect* is a modified version of *wdfs* WebDAV Client that allows REDIRECT and GSI proxy certificates as authentication credentials. |
| 2 | **The WebDAV clients do not support partial download of the files. Before reading the file, the WebDAV client copies the entire file to the local computer, and only after this allows reading.** |
| *Solution* | - |
| 3 | **The Windows 7 WebDAV Client does not accept XML responses for PROPFIND requests if the order of XML-tags is not identical as in the Appendix C.** |
| *Solution* | - |

Table – Problems identified while integrating the standard and grid technologies

|  |  |
| --- | --- |
| 1 | **None of the web server supports the GSI proxy certificates for mutual authentication.** |
| *Solution* | * Modify the web servers in order to support the GSI proxy certificates.   Create new modules (when possible) that will validate the GSI proxy certificates (Apache2 already has such a module: *mod\_gridsite*). |

Table – Problems identified while integrating the SUD with the Worker Node

|  |  |
| --- | --- |
| 1 | **The Unix-user that runs jobs on the worker nodes needs to be added to the *fuse* group.** |
| *Solution* | - |
| 2 | **The WN does not unmount automatically the mounted WebDAV repository when the job is finished. Actually, the WN tries to remove the directory, thus it tries to remove all the files that are mounted. This task is not possible (the WebDAV repository is read-only), thus the mounted directory will never be removed from the WN. This leads to an overload of the WebDAV Server (that needs to respond to the delete requests) and the WN (that needs to send the requests).** |
| *Solution* | The *fusermount* command needs to be used (within the job) to unmount the WebDAV repository after the job is completed. |
| 3 | **If the proxy certificate that resides on the WN is renewed, it needs to be delegated again to the SUD! Otherwise, the SUD will respond with “Permission Denied” anytime when the WN sends requests to it.** |
| *Solution* | - |

Appendix G: wget and curl – GSI proxy certificates

The *curl* tool accepts [GSI] proxy certificates if it is linked against the *openssl* library when it is compiled.

The *wget* tool does not accept [GSI] proxy certificate in any case. In order to accept proxy certificates, it needs to be modified as follows:

*SSL\_CTX\_use\_certificate\_file* method used to load the certificate needs to be replaced with *SSL\_CTX\_use\_certificate\_chain\_file.*

Appendix H: Development environment

|  |  |
| --- | --- |
| *Operating system* | CentOS 5 |
| *EPEL repository* | http://download.fedora.redhat.com/pub/epel/5/x86\_64/epel-release-5-4.noarch.rpm |
| *Yum repository* | http://repository.egi.eu/sw/production/cas/1/current/repo-files/EGI-trustanchors.repo |
| *Yum repository* | http://grid-deployment.web.cern.ch/grid-deployment/glite/repos/3.2/glite-UI.repo |

Any applications, tools and frameworks from the repositories presented above are maintained and can be used during the development.

Appendix I: Apache Benchmark (ab) example of usage

|  |
| --- |
| ab –c $CONCURENT\_REQUESTS –n $NUMBER\_OF\_REQUESTS – A $USER:$PASSWORD $URL  Example of output:  Requests per second: 24.56 [#/sec] (mean)  Time per request: 4071.685 [ms] (mean)  Time per request: 40.717 [ms] (mean, across all concurrent requests)  Transfer rate: 102.41 [Kbytes/sec] received |

Appendix J: Tools and frameworks used for implementation

Table – List of main tools and frameworks used to implement the project

|  |  |
| --- | --- |
| *Tool/framework* | *Description* |
| Apache2 | Used to host the WebDAV and Cache Node implementation |
| NginX | Used to serve data over the HTTP. |
| mod\_gridsite | Is a module that adds GSI support for Apache2 web server |
| mod\_authn\_myproxy | Is a module that provides basic authentication and credential delegation for an Apache2 web server |
| memcached | Is used to provide short-term caching functionality. It is also used as shared memory within the implementation |
| Python-lfc | Python interface to communicate with LFC servers |
| Lcg\_util | Python interface to communicate with the SE |
| GFAL | Python interface to communicate with the SE |
| wdfs | WebDAV client for Linux |
| htproxyput | Delegation client provided by gridsite |
| gridsite-delegation.cgi | CGI application that provides support for credential delegation |

About the Author

|  |  |  |
| --- | --- | --- |
| |  | | --- | |  | | Cristian Traian Cirstea received his Computer Science Engineer's degree in September 2008 from the “Politehnica” University of Bucharest, Faculty of Automatic Control and Computer Science, Romania. He carried out his final thesis on the topic of "Business Process Functional and Performance Metrics” at Fraunhofer FOKUS Institute, Berlin, Germany.  During his studies he worked as software developer and technical leader at Axway, Romania. After graduation he worked for 1 year as a software developer at Total Soft – a company that develops an e-purchasing system.  From 2009 until 2011, Cristian Traian Cirstea worked at the Eindhoven University of Technology, where he followed the Software Technology (OOTI) program from the 3TU.School for Technological Design, Stan Ackermans Institute. During his final project, he worked at Nikhef on a project that aimed to make the access to Grid’s data user-friendlier. Upon completion of the project, he received her PDEng degree from Stan Ackermans Institute in September 2011. |

|  |  |
| --- | --- |
|  |  |
|  |  |
| |  | | --- | |  | |  |

1. Nationaal instituut voor subatomaire fysica (National Institute for Subatomic Physics), Amsterdam, The Netherlands [↑](#footnote-ref-1)
2. Large Hadron Collider [↑](#footnote-ref-2)
3. Communication protocol over HTTP [↑](#footnote-ref-3)
4. European Organization for Nuclear Research is an international organization that operates the largest physics particle laboratory situated on the Franco-Swiss border. [↑](#footnote-ref-4)
5. *gLite* – Grid middleware framework (presented in *Domain Analysis* chapter) [↑](#footnote-ref-5)
6. *CentOS* is a community-supported, free operating system based on Red Hat Enterprise Linux (<http://www.centos.org/>) [↑](#footnote-ref-6)
7. A *segmentation fault*(often shortened to segfault) or bus error is generally an attempt to access memory that the CPU cannot physically address (according to Wikipedia). [↑](#footnote-ref-7)
8. <http://tools.ietf.org/html/rfc4918> [↑](#footnote-ref-8)
9. File system in user space (FUSE) is a loadable kernel module for Unix-like computer operating systems that allows non-privileged users to create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a "bridge" to the actual kernel interfaces (source Wikipedia.org). [↑](#footnote-ref-9)
10. CGI application is any program designed to accept and return data that conforms to the CGI specification (RFC 3875), In other words, a CGI application generates dynamic content for web servers. [↑](#footnote-ref-10)
11. Depends on the quality of the WebDAV implementation according to the protocol specifications. [↑](#footnote-ref-11)
12. It refers to how easy is to create new views with the files and directories received from the LFC or SE. [↑](#footnote-ref-12)
13. In case of the Approach 1, the robustness is medium due to the fact that once the FUSE application crashes from various reasons, the request will not be reprocessed, thus the user will receive an error message. In this case, the FUSE application needs to be restarted “manually”. [↑](#footnote-ref-13)
14. In case of the Approach 2, the robustness is high due to the fact that if something happens within the [F] CGI application and it crashes, the web server will immediately start it again and reprocesses the request. Thus the user will not observe the crash, but in the worst case he/she will wait a bit more for the request to be solved. [↑](#footnote-ref-14)
15. mod\_gridsite is a module offered by GridSite project and it is available on Grid Unix-repositories. [↑](#footnote-ref-15)
16. GridSite (<http://www.gridsite.org/>) is a set of extensions to the Apache web server and a toolkit for Grid credentials. [↑](#footnote-ref-16)
17. *gridsite-delegation.cgi* is a CGI application with a WSDL description and signing functions that allows users to delegate their credentials to a service (http://www.gridsite.org/wiki/Delegation\_protocol). [↑](#footnote-ref-17)
18. *htproxyput* is a client application (provided by GridSite) used by the user to delegate the credentials to a service that provides the gridsite-delegation.cgi. It is a Linux application, thus it can be used only on Linux machines. [↑](#footnote-ref-18)
19. *mod\_authn\_myproxy* is an Apache2 module that allows clients to authenticate against a MyProxy server (http://grid.ncsa.illinois.edu/myproxy/apache/) [↑](#footnote-ref-19)
20. *Grid Proxy Manager* is a Firefox plugin that helps users to upload their credentials (X.509 certificate) to a MyProxy Server. The module can be found on <https://addons.mozilla.org/nl/firefox/addon/grid-proxy-manager/> and it works on any operating system that is supported by Mozilla Firefox (<http://www.mozilla.com/en-US/firefox/new/>) [↑](#footnote-ref-20)
21. *reverse proxy cache* is a type of server that retrieves resources on behalf of a client from one or more servers, and caches them (usually) for a short period of time, as long as the space permits [↑](#footnote-ref-21)
22. Varnish SSL support - <https://www.varnish-cache.org/docs/trunk/phk/ssl.html> [↑](#footnote-ref-22)
23. A content delivery network or content distribution network (CDN) is a system of computers containing copies of data placed at various nodes of a network. [↑](#footnote-ref-23)
24. An approach is cache efficient if the space required to cache a file does not exceed the size of the file [↑](#footnote-ref-24)
25. The control of the cached files refers to the possibility to manipulate from an external script/program the content or the policy of the cache and to check the content of the cache anytime from anywhere. [↑](#footnote-ref-25)
26. *OpenSSL* is an open source implementation of the SSL and TLS protocols. [↑](#footnote-ref-26)
27. Content Delivery Network (CDN) is a geographically distributed data delivery network. By deploying multiple Cache Nodes that cooperate at different sites, the SUD implements a CDN. [↑](#footnote-ref-27)
28. *RFC 4918* – HTTP Extensions for Web Distributed Authoring and Versioning [↑](#footnote-ref-28)
29. *davfs2* – WebDAV client for Unix (http://savannah.nongnu.org/projects/davfs2) [↑](#footnote-ref-29)
30. *cadaver* – command-line WebDAV client for Unix (http://www.webdav.org/) [↑](#footnote-ref-30)
31. *ApacheBench (ab)* – is a command line computer program for measuring the performance of HTTP web servers. [↑](#footnote-ref-31)
32. *Valgrind* is an instrumentation framework for building dynamic analysis tools. It has several modules that analyze the execution of applications such as memory error detector, thread error detector and heap profiler (for more details the reader should refer to <http://www.valgrind.org>) [↑](#footnote-ref-32)
33. *wdfs-redirect* - is a WebDAV client with special modifications to accept X.509 GSI Proxy Certificate kind of authentication, due to the fact that the Grid largely uses it. More details in *Appendix* B. [↑](#footnote-ref-33)
34. As described in the *Feasibility Study*, a request to LFC takes between 100 and 200ms; and the request to SE to check the permissions takes around 500ms. [↑](#footnote-ref-34)
35. ROOT is an object-oriented framework for large-scale data analysis. [↑](#footnote-ref-35)
36. http://noedler.de/projekte/wdfs/ [↑](#footnote-ref-36)