



PALESTINE POLYTECHNIC UNIVERSITY

College of IT and Computer Engineering

Department of Computer Science and Computer Engineering

Graduation Project

Campus Grid Computing System

Project Team

Ibrahim Qdemat & Muhammad Dwaib

Supervisor

Dr. Mohammed Al-dasht

According to the system of the College of IT and Computer Engineering, and to the recommendation of the Project Supervisor, this project is presented to Computer Science and Computer Engineering Department as a part of requirements of B.Sc. degree in Computer System Engineering.

Hebron-Palestine

Jun-2014

Signatures

Project Supervisor signature

.....

Testing Group signature

.....

Department Headmaster signature

.....

Dedication

We dedicate this work to our parents, brothers, sisters and friends. Without their continuous support, endless patience, understanding and encouragement, this project wouldn't have been completed successfully.

Acknowledgment

Our deepest gratitude and thanks go to our supervisor, Dr. Mohammed Aldesht for his great efforts, enlightening guidance, continuous supports and encouragement throughout the entire period of this project. Also we would like to thank labs' supervisors, lecturers and students for their valuable assistance during our experiments. We wish to thank the examining committee for their support and guidance.

Special thanks to our parents, brothers and sisters, who always encourage, support us throughout our life. We are also grateful to all our colleagues in the computer engineering field. We would like to thank everyone who, in way or another, contributed in the preparation and the completion of this project.

Abstract

Grid computing is a service for sharing computer power and data storage capacity over the Internet. This project aims to increase the utilization of the available computing resources in PPU. Computing resources at different facilities of the university especially those in labs are almost idle for long periods of time or for scattered periods. These resources can be exploited in another place or for other computational tasks in the university.

The goal of this project is to develop a grid computing system that provides the research units with a large computational power to enable the working on a computational-intensive experiments that may not be able to handle previously.

In addition, this project aims to operate a grid computing system with minimal costs. Hardware costs are eliminated since the project will use the hardware components that are already exist in the university, no additional components are needed. Software costs approaches to zero since all the dedicated software needed to develop the system are free open source software. So we will get a system that will provide a large computing power with a minimal cost.

الملخص

الحوسبة الشبكية هي احد مجالات الانظمة الموزعة والتي تهدف الى استغلال القدرة الحسابية للأجهزة الموزعة على الشبكة لتشكيل حاسوب ظاهري بقدرة معالجة فائقة. يتم ذلك عن طريق تقسيم المشاريع الضخمة الى اجزاء مستقلة بحيث يتم توزيع هذه الاجزاء على الاجهزة التي تكون في حالة الخمول والتي تعمل بدورها على معالجة هذه الاجزاء وتعيد النتائج.

هناك العديد من المرافق في الجامعة التي تحتاج الى قدرات حسابية هائلة لكي تتمكن من انجاز مهامها بالشكل الصحيح بالإضافة إلى العديد من الأبحاث التي تحتاج إلى وقت طويل إذا تم تنفيذها على أجهزة حاسوب عادية. ويمكن توفير هذه القدرات الحسابية العالية عن طريق تزويد هذه المرافق بالحواسيب ذات القدرات الهائلة (super computers). ولكن هذه الحواسيب ذات كلفة باهظة وفي اغلب الاحيان ليست متوفرة لنا في فلسطين. وفي المقابل يمكن ايضا توفير القدرات الحسابية الهائلة عن طريق بناء نظام الحوسبة الشبكية الذي يعمل على تجميع القدرات الحسابية لأجهزة الحاسوب الموجودة في مختبرات الحاسوب المختلفة بحيث نحصل على جهاز حاسوب ظاهري فائق القدرة يمكنه تزويد المرافق المختلفة بالقدرات الحسابية التي تحتاج اليها.

اثناء هذا المشروع سنعمل على اجراء دراسة عملية لحساب نسبة استغلال القدرة الحسابية للأجهزة الموجودة في مختبرات الجامعة. ايضا سنعمل على بناء نظام الحوسبة الشبكية على مجموعة من الاجهزة . وفي النهاية سنقوم بتزويد النظام بالقدرة على استقبال وظائف حسابية من المستخدمين ليقوم بمعالجتها وارجاع النتائج.

Project Contributions

This document summarizes the contributions of our project. Each of these contributions is described briefly below.

Studying CPU usage at PPU computer labs

The study applied to find the average CPU utilization at PPU computer labs. Two freeware programs were used for this purpose: Altra CPU monitor and CPU usage logger. These two programs installed on a sample of computers from different labs. The study applied for approximately one month.

Through this study we found that the average CPU usage at PPU computer labs is approximately around 9% for most of the time. This means that there is a huge computational power can be obtained from computers at PPU computer labs to perform important researchs. For more details about this study see section 7.2.

Configuring and installing the grid server and grid clients

We had configured a grid computing system which consists basically from grid server and grid clients. Grid server control and manage the overall system. Grid clients represent resources that can be attached to the server to perform computations.

This system provides the environment for researchers to perform their computations. It also allows developers to focus on advanced features in developing the grid computing system rather than starting from the beginnings.

We chose the BOINC to be our grid computing middleware. BOINC server runs on UNIX operating systems. We used Ubuntu12.04 LTS 64-bit Linux distribution. In addition, we installed the BOINC server software prerequisites and dependencies. Furthermore, we solved some problems that appeared during the deploying of BOINC server. The detailed explanation about the BOINC server setting up process is described in appendix A.

At the client side, BOINC client software was installed on each computer participated in our grid system. We installed BOINC client as a service by checking the **Service**

Install checkbox. In addition, we disabled screen saver option and prevented other users (usually students) from controlling BOINC client software. More details at section 6.2.2.

Customizing and automating the core functionalities of the grid system

The BOINC middleware implemented the core functionalities of the grid system. But if you want to handle any of these functionalities, you need to deeply understand the BOINC environment, you must be able to write shell scripts, you have to be familiar with Linux OS administration, you also need to do a lot of steps to perform any of these functions.

We implemented a set of shell scripts that simplify the process of executing any of these functions. You can, for example, create a whole BOINC project by executing a one terminal command. Using these shell scripts we can control the BOINC project and the BOINC clients. These scripts also enable us to build another level of the system which is the grid portal. For more details about the implementation of these scripts you can see section 6.2.

Lunching Single Job Project

BOINC is designed to handle streams of millions of jobs. It takes some work to set up a stream: you need to create applications and application versions, workunit (WU) and result templates, validators, assimilators, etc. we implemented a set of shell scripts that handle running a job without any of these hassles and handle also the results of the execution.

To enable single job submission, we created a special project called **single job project**. **Single job project** is a BOINC project that it is configured to use single job submission mechanism supported by BOINC; but we modified the source code of this mechanism to make it suitable for our purpose.

In addition, we created a set of shell scripts and PHP files that are responsible for submitting a job, monitoring its execution and handling the result.

By single job mechanism, the system user (usually the researcher) can submit any C/C++ program compiled for a particular platform to be executed on one of clients attached to the **single job project**.

Also, we configured the **single job project** to support the famous and well-known platforms: Windows, Linux and MAC operating systems with their 32-bit and 64-bit architectures.

Furthermore, we automate the process of creating and configuring **single job project** which contains a lot of hassle and requires you to follow a long list of steps. Thus, the **single job project** becomes one of the projects that are supported by our grid system. It is designed to support the single job submission mechanism.

Implementing the Grid portal

Without a grid portal all the work and interaction with the grid system is done locally at the grid server by executing terminal commands. To simplify the interaction with the grid system such that a grid user doesn't need to write terminal commands. Also, building the grid portal enables the interaction with the system remotely.

Thus, the grid portal enables users to interact with the grid system remotely through user friendly interfaces; they don't need to know anything about the underlying system. Grid portal make the use of the shell scripts implemented previously to support management and control over the grid system, enable the users to submit jobs for execution over the grid system and enable the control of the jobs execution. More details about the grid portal can be found in section 6.3.

Testing the Grid system performance

After investigating the grid computing system we performed an experiment to test its performance. This experiment was applied by running a BOINC project that contains a simple application that provides computations for execution over the grid clients. This application is

called Test application provided by the BOINC; it generates instances that read input text files, converts their contents to uppercase and write the results to the output files.

The sample size of grid clients used in this experiment is 40 PC. The experiment was applied for two weeks. Through this experiment we found that each PC provides approximately 3.45 GFLOPS as an average. See section 7.3 for more information.

List of contents

Title.....	I	
Signatures	II	
Dedication.....	III	
Acknowledgment.....	IV	
Abstract	V	
Arabic Abstract	VI	
Project Contributions.....	VII	
List of Contents.....	XI	
List of tables	XVI	
List of figures	XVIII	
List of Appendices	XXII	
Chapter One	Introduction	7
1.1 Overview		1
1.2 Idea Description		1
1.3 Problem/Motivation		2
1.4 Project Scope		3
1.5 Summary		7
Chapter Two	Literature Review and Theoretical Background	8
2.1 Overview		8
2.2 Theoretical Background		8
2.2.1 Definitions		8
2.2.2 Building a grid.....		9
2.2.3 Classification of Grids		11
2.2.4 Grid computing portals		15
2.2.4 Grid-powered projects.....		16
2.3 Literature Review		16
2.3.1 PC Grid Computing In Higher Education Institutions.....		17
2.3.2 How is grid computing different from the World Wide Web?.....		18
2.3.3 Grid versus Volunteer Computing		18

6.2.8 Update Attached Projects	122
6.2.9 Project Deletion	123
6.2.10 SingleJob Project.....	125
6.2.10.1 Single job submission mechanism.....	125
6.2.10.2 SingleJob Project	125
6.2.10.3 Automatic SingleJob Project Configuration.....	131
6.2.10.4 Job Submission	133
6.2.10.5 Job Submission Automation	136
6.3 Portal Implementation.....	141
6.3.1 Software development tools and programming languages	141
6.3.2 Portal subpages	141
6.3.3 Portal main page	152
6.4 Security Issues	155
6.4.1 Securing the Server and the Clients	155
6.4.2 Client/ Server Authentication and Authorization.....	156
6.4.3 Protecting Administrative web interface	157
6.4.4 Securing the Web Portal	158
6.5 Testing	159
6.5.1 Testing system functionalities (Testing lower level).....	159
6.5.2 Testing Portal functionalities (Testing higher level).....	174
6.6 Summary.....	193
Chapter Seven	Experiments and Results
	194
7.1 Overview	194
7.2 Average CPU usage at PPU computer labs	194
7.1.1 Environment specification and work difficulties.....	194
7.1.2 Practical Work.....	198
7.3 Examination the performance of PPU Environment	201
7.4 Summary	204

Chapter Eight	Conclusion and Future Work	205
8.1 Overview		205
8.2 Conclusion		205
8.3 Challenges		206
8.4 Future Work		208
8.5 Summary		214
References		215

Appendix A	Server and Client Software Installation	220
A.1 BOINC server Pre-installation requirements		220
A.1.1 Hardware requirements		220
A.1.2 Software requirements		221
A.2 BOINC server installation process		221
A.3 Trouble Shooting		222
A.4 BOINC Client Installation		225
A.4.1 Microsoft Windows		225
A.4.2 Linux		229
A.4.3 Other Platforms		231
A.4.4 BOINC Client Security		231

Appendix B	Project Creation	234
B.1 BOINC Project		234
B.1.1 Project DB		234
B.1.2 Project Directory		235
B.1.3 Project configuration file		236
B.2 Project creation pre-requirements		237
B.3 Project Creation Process		237
B.3.1 Creating an Empty BOINC Project		238
B.3.2 Creating project having a test application example.....		251
B.4 phpMyAdmin Installation		253

Table 3.9: Hardware Components..	41
Table 3.10: Software Components	42
Table 4.1: Adding/deleting project use case template	46
Table 4.2: Adding/deleting user use case template.....	47
Table 4.3: Attaching/deleting resource use case template.....	48
Table 4.4: Update resource use case template	49
Table 4.5: Stopping/starting/restarting project use case template	50
Table 4.6: Modifying user information use case template	51
Table 4.7: Managing computing preferences use case template	52
Table 4.8: Monitoring project status use case template.....	53
Table 4.9: Upload a new job use case template.....	54
Table 4.10: Monitor jobs execution status use case template.	55
Table 4.11: Abort jobs while it is in progress stat.....	56
Table 4.12: Download the results of completed jobs.....	57
Table 4.13: Delete the results of completed jobs	58
Table 4.14: Modify account information	59
Table 4.15: Generate instance of job use case template	60
Table 4.16: Send instance to a client use case template	61
Table 4.17: Validate results use case template	62
Table 4.18: Prepare results for download by users	63
Table 4.19: Job Execution use case template	64
Table 4.20: DBConnection CRC.....	68
Table 4.21: ProjectCreation CRC	68
Table 4.22: ProjectManagment CRC	69
Table 4.23: UserCreation CRC	69
Table 4.24: JobSubmission CRC	70
Table 5.1: DBConnection class.....	85
Table 5.2: ProjectCreation class	85
Table 5.3: Project Management class	86
Table 5.4: User Creation class	87
Table 5.5: JobSubmission class	88
Table 6.1: Supported Platforms.....	131
Table 7.1: Computers specifications.....	195

Table 7.2: Average CPU usage at AL-Beruni(I) lab for one day.....	198
Table 7.3: Average CPU usage in PPU computer labs	199

List of figures

Figure 2.1: Grid Architecture	10
Figure 2.2: A typical form of cluster computing	11
Figure 2.3 Example of Enterprise Grid Infrastructure	12
Figure 2.4 Local PC Grid at the University of Westminister	19
Figure 3.1: Figure 3.1: Project time line.....	27
Figure 3.2: BOINC Architecture	29
Figure 3.3: Alchemi layer architecture	31
Figure 4.1: Admin use-case diagram.....	65
Figure 4.2: User use-case diagram	66
Figure 4.3: system and client use-case diagram	67
Figure 4.4: Class Hierarchies and relationships diagrams.....	71
Figure 5.1: BOINC architecture	74
Figure 5.2: BOINC Client.....	75
Figure 5.3: Installing BOINC Server Software.....	77
Figure 5.4:Creating And Running A BOINC Project	79
Figure 5.5: Deploying BOINC Clients.....	83
Figure 5.6: Project creation.....	89
Figure 5.7: stopping/starting/restarting project	90
Figure 5.8: attaching/detaching client to a BOINC project	91
Figure 5.9: disable/enable account creation	92
Figure 5.10: adding new user to a BOINC project	93
Figure 5.11: Users management	94
Figure 5.12: job submission	95
Figure 5.13: jobs management	96
Figure 5.14: Job Execution	97
Figure 5.15: The About Page	98
Figure 5.16: Contact page	99
Figure 5.17: Login Page Interface	100
Figure 5.18: Admin home page interface	101

Figure 5.19: Users page interface	102
Figure 5.20: Project management page interface	103
Figure 5.21: Modify user page interface	104
Figure 5.22: Job submission page interface	105
Figure 5.23: modify account page interface	106
Figure 5.24: Grid System DB ERD	107
Figure 5.25: Hardware interface design	108
Figure 6.1: Create Test Project Algorithm	116
Figure 6.2: Create Account Algorithm	117
Figure 6.3: Attachment/Detachment Algorithm	120
Figure 6.4: Update Attached Projects Algorithm	123
Figure 6.5: Project Deletion Algorithm	124
Figure 6.6: Single Job Configuration Algorithm.....	132
Figure 6.7: Submit Job Algorithm	139
Figure 6.8: Output Handler Algorithm	140
Figure 6.9: Admin page left part	142
Figure 6.10: User page left part	142
Figure 6.11: Project creation form	143
Figure 6.12: Project creation form	144
Figure 6.13: Add new user form	145
Figure 6.14: Add new user form	146
Figure 6.15: Project management form	147
Figure 6.16: Client attachment	148
Figure 6.17: Header subpage	151
Figure 6.18: Footer subpage	151
Figure 6.19: Download result	151
Figure 6.20: Results subpage	152
Figure 6.21: Admin home page	153
Figure 6.22: Users page	154
Figure 6.23: Create Project case 1	159
Figure 6.24: Project home page	160
Figure 6.25: Project creation case 2(wrong usage).....	160
Figure 6.26: Project creation case 3(wrong usage)	160

Figure 6.27: Create Test Project	161
Figure 6.28: Create Account	162
Figure 6.29: Attach client	162
Figure 6.30: Attached client BOINC manager	163
Figure 6.31: Update attached projects	163
Figure 6.32: BOINC manager after executing updateAttachedProject.sh	164
Figure 6.33: Detach a client	164
Figure 6.34: BOINC manager after executing deattach.sh	165
Figure 6.35: Stop project	166
Figure 6.36: Start project	166
Figure 6.37: Restart project	167
Figure 6.38: Customize project to single Job project	168
Figure 6.39: Project status executing singleJob.sh	168
Figure 6.40: Submit job	169
Figure 6.41: Abort job	169
Figure 6.42: Output handler	170
Figure 6.43: Enable account creation	171
Figure 6.44: Project home page after executing enable_account_creation.sh	171
Figure 6.45: Disable account creation	172
Figure 6.46: Project home page after executing disable_account_creation.sh	172
Figure 6.47: Project status	173
Figure 6.48: Project deletion	173
Figure 6.49: Project creation test 1	174
Figure 6.50: Project creation test 2	175
Figure 6.51: Successful Project Creation	175
Figure 6.52: Error handling test for adding new user	176
Figure 6.53: Test adding new user	177
Figure 6.54: Error handling test for attaching client	177
Figure 6.55: Before attaching a client	178
Figure 6.56: After attaching a client	178
Figure 6.57: BOINC manager of the updated client	179
Figure 6.58: Project status page after project stop	180
Figure 6.59: Project status page after start/restart project	180
Figure 6.60: Project management page after deleting the project	181

Figure 6.61: Invalid account modification	182
Figure 6.62: Valid account modification	182
Figure 6.63: User information before modification	183
Figure 6.64: User information after modification	183
Figure 6.65: Enabling account creation	184
Figure 6.66: Disabling account creation	185
Figure 6.67: Checking users for deletion	185
Figure 6.68: Users deletion	186
Figure 6.69: Error job submission	186
Figure 6.70: Valid job submission	187
Figure 6.71: Aborting Job confirmation	188
Figure 6.72: Aborting job	188
Figure 6.73: Job execution (state 1)	189
Figure 6.74: Job execution (state 2)	189
Figure 6.75: Job execution (state 3)	190
Figure 6.76: Job execution (state 4)	190
Figure 6.77: Download window	191
Figure 6.78: Check results to delete	192
Figure 6.79: Home page after click Delete Jobs	192
Figure 7.1: CPU usage logger program	196
Figure 7.2: Ultra CPU monitor program	197
Figure 7.3: Sample of the log file generated by CPU usage logger	199
Figure 7.4: Average CPU usage at PPU labs	200
Figure A.3.1: BOINC installation problems 1	141
Figure A.3.2: BOINC installation problems 2	142
Figure A.3.3: BOINC installation problems 3	143
Figure A.3.4: Proper BOINC installation	144
Figure A.4.1: BOINC Client Deployment Step 1	200
Figure A.4.2: BOINC Client Deployment Step 2	201
Figure A.4.3: BOINC Client Deployment Step 3	202
Figure A.4.4: BOINC Client Deployment Step 4	203
Figure A.4.5: BOINC Client Deployment Step 5	204
Figure A.4.6: BOINC Client Deployment Step 6	205

Figure A.4.7: BOINC Client Deployment Step 7	206
Figure A.4.8: BOINC Client Deployment Step 8	207
Figure B.1: Project home page	200
Figure B.2: Adding project name 1	201
Figure B.3: Adding project name 2	202
Figure B.4: Adding copy rights 1	203
Figure B.5: Adding copy rights 2	204
Figure B.6: Setting admin. Account 1	205
Figure B.7: Invalid login to admin. page	206
Figure B.8: Setting admin. Account 2	207
Figure B.9: Administrative page	208
Figure B.10: Project status 1	209
Figure B.11: Project status 2	210
Figure B.12: Project status 3	211
Figure B.13: Project status 4	212
Figure B.14: Project forum 1	213
Figure B.15: Project forum 2	214
Figure B.16: Test example application project status	215
Figure B.17: phpMyAdmin login page	216

List of Appendices

Appendix A	Server and Client Software Installation	220
Appendix B	Project Creation	234
Appendix C	Security	255
Appendix D	Boinccmd tool	263
Appendix E	Glossary	268

Chapter One

Introduction

1.1 Overview

In this chapter we provide an introduction to our project concentrating on three main topics. First, we briefly provide a description of the project idea. Second, we talk about the problems and motivations that lead to the idea of this project. Finally, we talk about the scope of the project.

1.2 Idea Description

Grid computing is a form of distributed computing in which an organization (business, university, etc.) uses its existing computers (desktop and/or cluster nodes) to handle its own long-running computational tasks [1]. In our project we want to make use of grid computing to utilize the computing resources available in our university by building our own grid.

The PPU grid will make use of the computing power that is supplied by computing lab machines, desktop and laptops belonging to faculty, staff, students, and any computing resource belonging to our university.

The project will be done into two stages. First, we have to gather the information about all the computing resources in the university, analyze the financial costs of upgrading some important resources to new ones with more computing capabilities and make a comparison between the current situation and the new one, supposing

the grid computing is applied. Secondly, we have to achieve the technical work and apply the grid computing on some facilities of the university.

There are some basic components and technologies that are needed in the project. We have the grid resources, such as computers, electronic data storages and any other computing resource connected to the network. These grid resources need to be connected using a network, so we need to have a fast and reliable network. In addition, we need a middleware layer that provides the tools that enable the various elements (servers, computers, storage, networks, etc.) to participate in a grid and communicate with each other.

This project will basically concentrate on distributed systems and grid computing to deal with workloads and computing resources. This is done by dividing large projects into smaller independent tasks and assigning these tasks to the available idle computing resources.

1.3 Problem/Motivation

While the computing resources are utilized during the teaching periods, there are large periods of time during the night or holidays or for scattered periods when these resources like computer labs are idle. These unused computing resources can be employed to run computation-intensive tasks for different university units.

Also, we know many of the PPU services sometimes suffer from latency due to the large number of users and poor resources, e.g. e-registration and e-learning. Grid computing can help solving this problem and enhancing these services at a very low cost.

Many researches use computing-intensive algorithms and may need a large amount of data sets to solve their research problems. This large computing power and memory capacities can be provided at little or no cost to university researchers using our grid. This resource enables previously infeasible researches or shortens the needed time for others. This leads to the publicity of the university and its researches.

Sometimes researchers realize that they need to use distributed computing; but they lack the expertise and resources to do it themselves. The basic idea behind this is to divide a huge work which may take days to be computed on a standalone computer, into many pieces which can be done concurrently by the available computing resources at their idle time. Finally gather the work done by these computers to produce the result.

In addition, the project team has many challenges and new things to be learned. Some of these are: learning new technologies, looking deeper in the networks functionalities, dealing with middleware layer and learn about distributed systems, grid computing and network's programming.

1.4 Project Scope

Basically this project will concentrates on utilizing the computing resources of PPU labs to fill the shortage in some facilities, like the research unit or to improve some services like E_Learning or E_Registration. The main features of the project scope are listed below with more details:

1. System Input:

- Computational projects or tasks.
- Clients demands.

2. System Outputs :

- Results for a running project and tasks to the clients demands. This is done by achieving balanced load distribution and concurrency execution using the available computing resources to get results in more efficient manner.

3. Project requirements :

- a) Utilize the available computing resources.
- b) Supply some facilities of the university with the available computational resources to be able to perform large computational tasks.
- c) Solve some problems that come from the limited performance of the available computing resources.

4. System components :

- a) Server: represents the system hardware backbone. It supplies some coordination functionalities in the system.
- b) Computing resources: the main supplier for these components is the university labs. These computing resources will be configured to accept the computational tasks that are assigned by the server and supply results back.
- c) Network: an excellent network bandwidth to connect system components together.
- d) Middleware: represents the system software backbone. It will provide the suitable grid environment for all components. It provides the main system functionalities, management and control.

5. Project Deliverables

We will provide a study about the project visibility by analyzing different factors on different aspects:

- Analysis of the performance of the available computing resources at the university.
- Analysis the need of some researches in the university to a high computational power.
- Analysis of the utilization of some computing resources especially those are in labs.

After achieving the technical work and building the grid system, the final output of the project will be as follow:

A user can login to the grid portal using a username and password. The user will be able to upload his computational task to the server. Now, the server is responsible to find the available idle computing resources and send this task to these resources which execute it and return the result back to server. Finally, the result will be returned to that user.

6. Project assumptions, boundaries and constraints

Grid computing system needs a good network bandwidth and infrastructure to work well. This means that the access to the grid system from the web will not be efficient until upgrading the overall network capabilities.

The process of dividing the large computing task into smaller parallel tasks is not a part of this project. We assume that the server will get tasks which will be executed concurrently on different computational resources. Projects of multi-tasks must be partitioned to independent tasks before they are delivered to the server.

Partitioning projects to independent tasks needs additional effort which is out of the scope of this project.

In this project will use one computer as a grid server and a set of computers in the university labs as grid clients. The main limitations and assumptions regarding this project can be stated and summarized in the following points:

1. The project considers computers available in one of computer labs in wad-Alharria branch at PPU. This sample is small compared to available computers in all PPU branches.

2. The project is limited to PCs available in computer laboratories and does not include servers or computers used by academic/administrative staff for security matters.

4. The project is performed on these PCs during work hours (08:00 – 16:00) from Sunday to Thursday only.

5. Most of experiments in this study will be performed on PCs with Microsoft Windows OS (XP Professional and x86 Windows 7 Professional x86).

6- In our project we assume that the project to be executed is ready for deployment on the grid; i.e. the project is gridified and all jobs are identified.

7. Initial project organization:

- Project Team:

- Group of 2 students.

- Stakeholders:

- Computer center staff.
- University teaching staff.
- Students.
- Researchers

1.5 Summary

In this chapter we talked about the project idea. Then, we presented some of the project motivations. Finally, we have discussed the project scope which addressed these topics: system input and output, project requirements, components, deliverables, assumptions, boundaries constraints, and finally the Initial project organization.

Chapter Two

Literature Review and Theoretical Background

2.1 Overview

The purpose of this chapter is to clarify some of the concepts and components associated with grid computing. In addition, this chapter presents an overview of grid technologies and compares it with several similar technologies. Also, we will discuss some of the previous related projects and studies related to grid computing.

2.2 Theoretical Background

In this section we discuss the grid computing technology and its related issues in more details.

2.2.1 Definitions

The "Grid" takes its name from the analogy with the electrical "power grid" [2]. The grid power provides you with electricity just by plugging-in your device into wall socket; you do not have to care how the electricity was generated or how it reaches to your home. The same thing happens in grid computing technology which provides you with required computing power and storage resources without any need to know how you get these resources or from where it is originated, the grid hides all of these details.

Since the grid computing is emerging and evolving technology, there are many definitions to grid computing according to the different understanding of this term, but all of them share the same concept. Here we show some of these definitions from different sources:

- According to [3], the first and most cited definition of Grid Computing was suggested by Foster and Kesselman (1998):
"A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities".
- Grid computing is a service for sharing computer power and data storage capacity over the Internet [4].
- A grid is a collection of machines typically referred to as “nodes”, “resources”, “clients”, “hosts”, and other similar terms [5].

2.2.2 Building a grid

There are three things that should be available in order to set up a grid: architecture, hardware, and middleware layer. These components are described briefly below.

1. Architecture:

The overall design of the grid is called the grid architecture; it identifies the fundamental components that should be taken in consideration when a grid is to be built.

A grid's architecture can be divided into layers, where each layer has a specific function. These layers are described below from the lowest layer to the highest:

- **The network layer:** which connects grid resources and deals with different components like routers, switches, etc.
- **The resource layer:** actual grid resources, such as computers or storage systems that are connected to the network [6].
- **The middleware layer:** provides the tools that enable the various elements (servers, storage, networks, etc.) to participate in a grid. The middleware layer represents the environment where the grid can be built and work [6].
- **The application layer:** This includes applications in science, engineering, business, finance and more, as well as portals and development toolkits to support the applications. This is the layer that grid users see and interact with [6].

2. Hardware

The hardware forms the physical infrastructure of a grid. A grid depends on underlying hardware like computers and networks which are the backbone of the grid.

3. Middleware

Middleware is the software that organizes and integrates the resources in a grid [7]. Middleware layer is made up of many software programs that coordinate all the different grid resources. Middleware resides between the operating systems software (like Windows or Linux) and the applications software [7].

The grid architecture is depicted in figure 2.1:

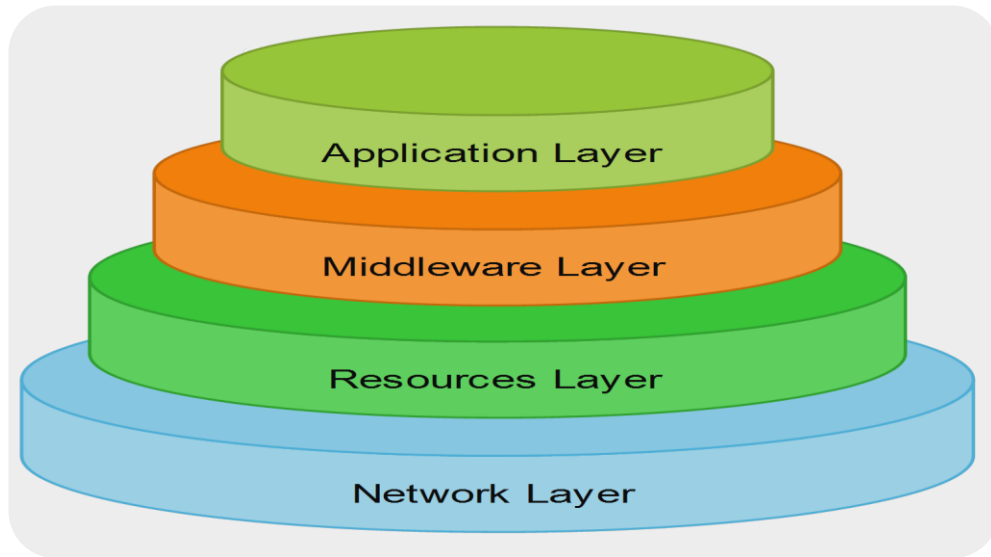


Fig. 2.1: Grid Architecture

2.2.3 Classification of Grids

Grid Architecture is still in the evolving stage. There are many variations and types of Grids. There are different classifications of grids based on type of classification or based on understanding of grid concept.

There is no standard in categorization of grids. Many Research Analysts, IT vendors, and Computer scientists began classifying the grid and grid variations based on their own understanding and vision. Some base it on the functionality, some base it on the architecture and others on the built-in components. Many organizations have different focuses, thus resulting in different classifications [8]. In this section we explain the classification of the grid based on scalability and function concentration. Grids can be classified based on the scalability as below:

1. Cluster Grids:

Cluster Grids, or clusters, are a collection of co-located computers connected by a high-speed local area network and designed to be used as an integrated computing or data processing resource (see Fig. 2.2) [9].

Cluster grids are the most popular and simplest form of a grid. Cluster grid consists of one or more systems, working together, to provide a single point of access to users. Cluster grid meets the need of most of the organizations. Typically used by a team of users such as a single project or a department, a cluster grid supports both high throughput and better performance for the jobs [5].

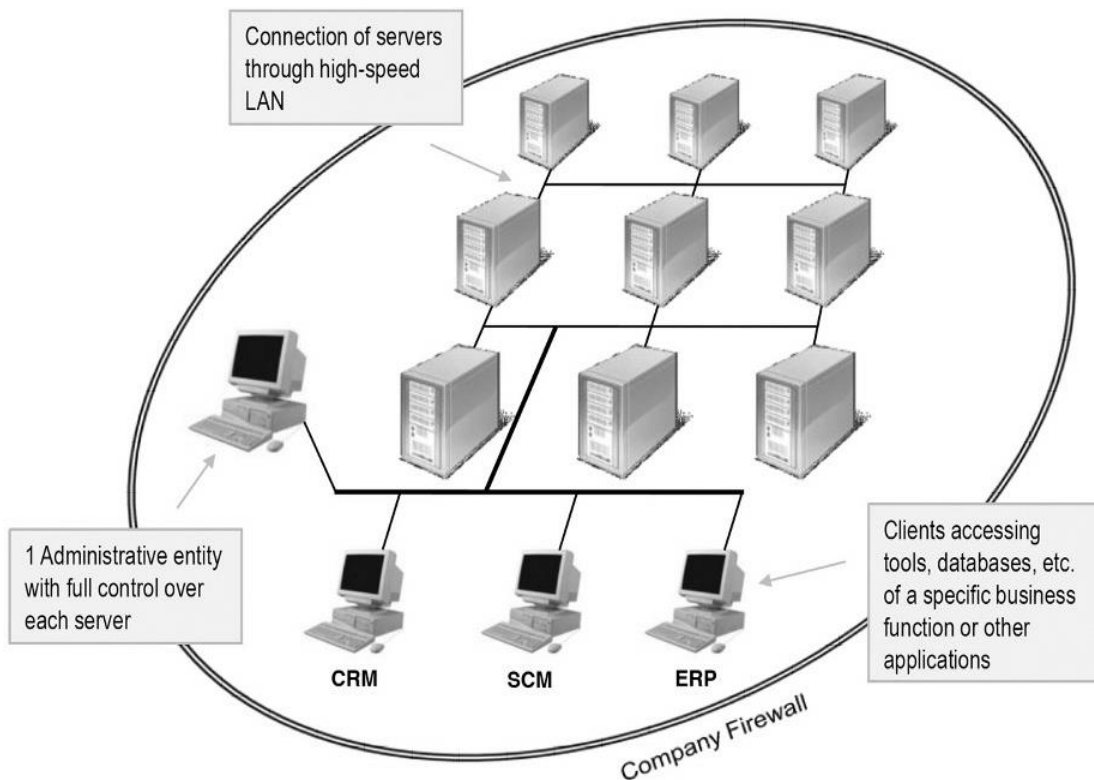


Fig. 2.2: A typical form of cluster computing, Source [15]

2. Campus/ Enterprise Grids

Campus grids enable multiple projects or departments to share computing resources in a cooperative way. It is also referred as the cooperative grid. Campus grids may consist of dispersed workstations and servers, as well as centralized resources located in multiple administrative domains, in departments, or across the enterprise [5].

The term Enterprise Grid is used to refer to application of Grid Computing for sharing resources within the bounds of a single company [10]. All components of an Enterprise Grid operate inside the firewall of a company, but may be heterogeneous and physically distributed across multiple company locations or sites and may belong to different administrative domains (see Fig. 2.3) [9].

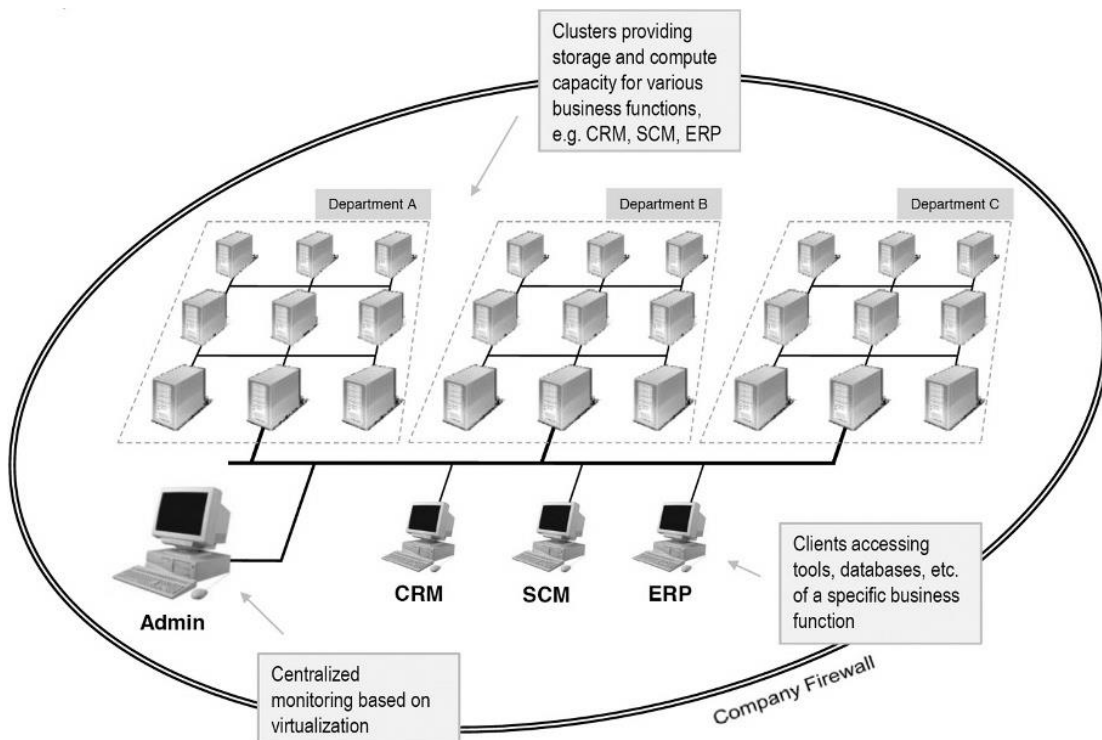


Fig. 2.3: Example of Enterprise Grid Infrastructure, Source [15]

3. Global Grids

When application needs exceed the capacity of a campus grid, organizations can tap partner resources through a global grid. It is designed to support and address the needs of multiple sites and organizations, global grids provide the power of distributed resources to users anywhere in the world for computing and collaboration. Individuals or organizations sending overflow work to a grid provider or by multiple companies working together and sharing data - crossing organizational boundaries with ease can use the global grid [5].

Also, grid can be categorized based on the grid infrastructure and functions concentration. In this classification we have two categories; data grids and computational grids which are described briefly below:

1. Computational Grids

A computational grid aggregates the processing power of a distributed collection of heterogeneous systems [5]. In this category, the emphasis in Grid infrastructure is given on the computation. A large computing problem is divided into sub-problems and then solved over the nodes of the grid independently. Large scale problems in Science and Engineering are being solved on the computational grid. The computing environment of a computational grid provides a demand driven, reliable, powerful and yet an inexpensive power for its customers [11]. Thus a computational grid environment consist of one or more hardware and software enabled environments that provide dependable, consistent, pervasive and inexpensive access to high end computational capabilities [12].

2. Data Grids

A data grid focuses on secure access to distributed, heterogeneous pools of data [5]. In data grid, the emphasis is over the management of the data that is being held in a variety of data storage facilities in geographically dispersed locations. The data sources may be databases, file systems and storage devices. The grid must also provide data virtualization services to satisfy various transparency issues e.g. transparency for data access, integration, and processing. Security and privacy is very important requirements of data in a grid system and is very complex [11].

2.2.4 Grid computing portals

Web-based Grid computing portals, or Grid portals [13], have been established as effective tools for providing users of computational Grids with simple, intuitive interfaces for accessing Grid information and for using Grid resources [14].

Grid middleware such as the Globus Toolkit provides powerful capabilities for integrating a wide variety of computing and storage resources, instruments, and sensors, but Grid middleware packages generally have complex user interfaces (UIs) and Application Programming Interfaces (APIs). Grid portals make these distributed, heterogeneous compute and data Grid environments more accessible to users and scientists by utilizing common Web and UI conventions [15].

Grid portals are now being developed, deployed, and used on large Grids including the National Science Foundation (NSF), Partnership for Advanced Computational Infrastructure (PACI) TeraGrid, the NASA Information Power Grid, and the National Institute of Health (NIH) Biomedical Informatics Research Network [15].

The software used to build Grid portals must interact with the middleware running on Grid resources. The portal software must also be compatible with common Web servers and browsers/clients [15].

2.2.5 Grid-powered projects

There are hundreds of computer grids around the world. Many grids are used for e-science enabling projects that would be impossible without massive computing power [16].

Grid computing is changing the way the world is doing science, as well as business, entertainment, social science and more. Below are a few examples of grid-powered projects [16].

- The **WISDOM** project is using grid computing to speed the search for a cure for malaria, a disease that affects millions of people all over the developed world.
- **MammoGrid** is building a grid for hospitals to share and analyse mammograms in an effort to improve breast cancer treatment.
- The **MathCell** project aims to create a grid-managed multi-purpose environment for further research in biology and bioinformatics.
- The **P12S2** project uses grid computing to learn more about the spread of plant diseases.

2.3 Literature Review

We have reviewed and studied several references, scientific papers and books concerning grid computing and its related issues like volunteer computing and local desktop grid. In this section we summarize the most important and relevant topics.

2.3.1. PC Grid Computing Environment In Higher Education Institutions:

One of the recent researches in grid computing field was introduced by Bader Ahmed Bader Ajrab in 2013 at Al-Quds University as a master thesis. It had stated the importance of grid computing especially in high education institutions in Palestine.

Al-Quds Open University was chosen as a test bed of his research which focuses on local PC grid computing. Since AlQuds Open University has a lot of geographically distributed branches, reliable network communications and ample resources available in computer labs, it is considered the most suitable environment to be the test bed for local PC grid computing project [3].

An overall study for Al-Quds Open university different branches was introduced. This study encompasses the overall communication network, computer labs and specifications of computers in these labs, in addition to the authentication and authorization issues.

A study for percentage of computer utilization was performed on computer labs at Jerusalem and Bethany branches by running some programs that logs the CPU utilization for some periods of time. The researcher found that” the average CPU utilization doesn’t far exceed 10% for 90% of the time” [3].

Finally, some real experiments to build a grid were performed in Jerusalem and Bethany branches. These experiments used two middleware frameworks, which BOINC and Alchemi. A detailed study of the operating system and the network effect, in addition to the CPU utilization of grid computers is performed. This study

applied two times; one time using BOINC as a middleware and the other time using Alchemi as a middleware.

The result of the study proved that the computing resources at Al-Quds Open University are not utilized well. Also it shows that the grid computing is very useful to utilize these resources and fill the shortage of high performance supercomputers. This thesis shows some experiments on grid only. In contrast, our work will focus on the PPU environment.

We will provide a study about the available computing resources in PPU and the percentage of utilization of our resources. Then we will use one of the computer labs to build our grid using specific middleware. The main difference between our work and the previous one that he didn't make any real or users' computations, he just made specific tests using specific projects to prove the grid functionality while in our case we will provide a portal to upload user tasks to be executed on the grid.

2.3.2. How is grid computing different from the World Wide Web?

Grid computing uses the Internet to help us sharing computer power, while the Web uses the Internet to help us sharing information. Grid computing is making big contributions to scientific research, helping scientists around the world to analyze and store massive amounts of data [4].

2.3.3. Grid versus volunteer computing

Grid and volunteer computing are both forms of distributed computing and both aims to utilize the CPU usage but the main difference is the type of computing resources involved in the grid which leads to new other differences.

In volunteer computing the available computing resources (such as processing power and storage) are being donated by their owners; it is a matter of volunteering, anyone can donate his own computer to be part of the volunteer grid by downloading a special software for this purpose and it starts receiving jobs in its idle time. While you're not using your computer, someone else is using it for their research, perform simulations and otherwise contribute to some projects [17].

In Grid computing, an organization (business, university, etc.) uses its existing computers (desktop and/or cluster nodes) to handle its own long-running computational tasks. This differs from volunteer computing in several ways [18]:

- The computing resources can be trusted; i.e. one can assume that the PCs don't return results that are intentionally wrong, and that they don't falsify credit. Hence there is typically no need for replication.
- There is no need for screensaver graphics; in fact it may be desirable to have the computation be completely invisible and out of the control of the PC user.
- The deployment of middleware on clients is typically automated since the resources are under your authorization.
- In volunteer computing trust between resource providers and users is essential, especially when they don't know each other [19].

In addition, volunteer computing systems must deal with problems related to correctness [20]:

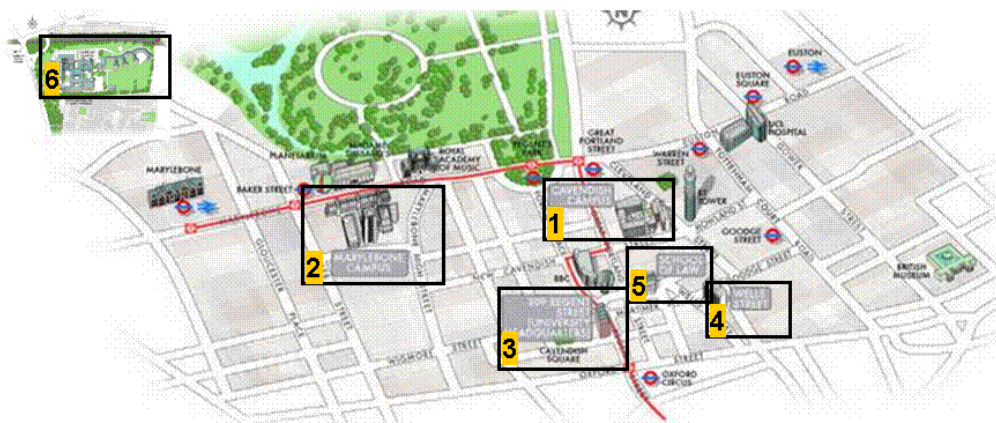
- Volunteers are unaccountable and essentially anonymous.
- Some volunteer computers occasionally malfunction and return incorrect results.
- Some volunteers intentionally return incorrect results or claim excessive credit for results.

2.3.4. The University of Westminster desktop grid system

The Centre for Parallel Computing at University of Westminster in UK built a local desktop grid system that currently includes over 1500 laboratory PCs which represents about half the total number of PCs in the university. These machines are available for desktop grid computations whenever they are switched on but not utilized by students for teaching or other purposes.

The university is set over four main campuses and some additional smaller locations in Central- and North-West-London each of them offering a variable number of mainly windows based PCs for teaching purposes. The table below summarizes the approximate number of machines offered by these locations and the figure gives an overview of the geographical location of the campuses [21].

- Over 1500 Windows PCs from 6 different campuses



Lifecycle of a node:

1.	New Cavendish Street	576 nodes
2.	Marylebone Campus	559 nodes
3.	Regent Street	395 nodes
4.	Wells Street	31 nodes
5.	Little Tichfield Street	66 nodes
6.	Harrow Campus	254 nodes

1. PCs basically used by students/staff
2. If unused, switch to Desktop Grid mode
3. No more work from DG server -> shutdown (green solution)

Fig. 2.4: Local PC Grid at the University of Westminster, Source [3]

The periods of time when PCs are idle were employed to run computation-intensive tasks for university researchers such as powerful simulation programs. Researchers using the Westminster Local Desktop Grid have found that they can shorten a typical execution time from weeks to hours.

The desktop grid system implemented at the University of Westminster is estimated to be equivalent, in raw computational power terms, to a £500,000 cluster procurement or supercomputer. Also the grid keep evolved and improved with no direct cost since the PCs themselves participated in the grid are continuously replaced from existing budgets. In other words, old PCs are always replaced with much better, higher performance ones, so the desktop grid always keep pace with PC performance improvements with no direct costs while other installed clusters need to be replaced or evolved on a three or four year cycle incurring all of the costs associated with this [22].

2.4. Summary

This chapter presented an overview of grid computing technology and its related issues. In addition, we discussed some of previous grid-based projects and studies and analyzed their results.

From the above discussion we can conclude that organizations can create their grids to match their special requirements and there is no fixed grid size can fit all organizations.

We can notice that there is no two grids are the same and each organization has its own environment and resources. Thus, we want to study our case in Palestine Polytechnic University (PPU) in order to be able to build our own suitable grid that can give an indication about what can be done in future in order to enhance our storage and computing capabilities.

Chapter Three

Project Management Plan

3.1 Overview

This chapter states the project plan, which includes all the sets of project tasks in addition to the project timeline. It also talks about the methodology of the work during project implementation. Project methodology consists of two parts, the first part introduces in general the steps of work flow in the project, and the second one states and analyzes the available options that can be used to implement the project. In addition, this chapter introduces some of project risks and an information sheet for each risk. Finally, it includes an analysis to all the project components with their estimated costs, these costs considered as the overall project cost.

3.2 Project Plan

This section will contain two basic subsections which are sets of project tasks and the project time line. These two subsections are stated as follow:

3.2.1 Sets of Project Tasks

- **Communications And Meetings : 2 weeks**

We will meet different units in the university and listen to their computational problems.

- We meet the computer center supervisors to get information about the available resources and their specifications.
 - We communicate with the scientific research deanship.
- **Data Collection and Data Analysis: 4 weeks**
- **Data Collection:**
 - ❖ Identify the available computing resources.
 - ❖ Determine the running time and workloads of these resources.
 - ❖ Determine the buying, maintenance and periodic costs of these resources.
 - ❖ Gather information about the underlying network.
 - **Data Analysis:**
 - ❖ Identify the amount of wasted (unutilized) computing powers when the computers are idle or run with minimal tasks.
 - ❖ Compute the utilization percent of our computing resources.
 - ❖ Make sure that the underlying infrastructure can support the grid.
 - ❖ Make a comparison between the current situation and the new one, supposing the grid computing is applied.
- **Project Analysis And Design: 8 weeks**
- **Analysis task set**
 - 1) Review requirements that have been collected during the data collection and analysis stages.
 - 2) Refine the user scenarios
 - Define all actors who deal with the grid.

- Represent how actors interact with the software.
 - Extract the functions and features from the user scenarios.
 - Review refined scenarios for completeness and accuracy.
- 3) Model the information domain (Data)
- Identify all major information objects.
 - Define attributes for each information object.
 - Determine the relationships between objects.
- 4) Model the functional domain(functions)
- How functions modify data objects.
 - Refine functions to provide elaborative details.
 - Describe each function and subfunction.
 - Review functional model.
- 5) Model the behavioral domain(behavior)
- Determine **events** that cause behavioral changes within system.
 - Identify the **states (modes)** that the system goes through in its response to particular event.
 - Describe how an event causes the system move from one state to another.
 - Review the behavioral models.
- 6) Modeling the graphical user interface(GUI)
- Determine the GUI Layout and basic user interaction requirements.
- **Design task set:**
 - 1) Review all analysis models for completeness and consistency.
 - 2) divide the analysis model into design subsystems

- Make sure the each subsystem is functionality cohesive.
 - Design subsystem interfaces.
 - Design appropriate data structures using the information domain model.
- 3) Based on interface analysis, design user interface.
- Review GUI analysis done in analysis activity.
 - Using user scenarios; specify action sequence.
 - Create behavioral model of the interface.
 - Define interface objects, control mechanism.
 - Review the interface design and revise as required.
- 4) conduct component-level design
- Specify all algorithms at a relatively low level of abstraction.
 - Refine the interface of each component.
 - Define component level data structures.
 - Review the component level design.
- **Prepare The Environment : 2 week**
- Choosing the suitable middleware software environment after investigating the available ones.
 - Setup the middleware and any other software on the server side and on the resource computing involved in the grid.
- **Middleware Investigation And Learning : 3 weeks**
- Read the middleware software documentation
 - Learn how to deal with middleware layer.

- Read more about grid computing and its implementation.
 - Learn server programming.
- **Project Implementation and coding: 10 weeks**
- 1) Build architectural infrastructure
 - Review the architectural design
 - Code and test components that enable architectural infrastructure.
 - Acquire reusable architectural patterns.
 - Test the infrastructure to ensure interface integrity.
 - 2) Build a software component
 - Review the component-level design.
 - Create a set of unit tests for the component.
 - Code component data structures and interfaces.
 - Code internal algorithms and related processing functions.
 - Review code as it is written
 - Look for correctness.
 - Ensure that coding standards have been maintained.
 - Ensure that the code is self-documenting.
 - 3) Unit test the component
 - Conduct all unit tests.
 - Correct errors uncovered.
 - Reply unit test.
 - 4) Integrate completed component into the architectural infrastructure.
- **Project Testing And Modification: 2 weeks**
- Test all project components, and modify when detecting errors.

3.2.2 Time estimation (Gantt chart): The following figure shows the overall project time estimation that starts from September of 2013 and continues to May of 2014.

ID	Task Name	Start	Finish	Duration	2013/9/14				2014/1/1					
					2013/9/14	2013/10/1	2013/11/1	2013/12/1	2014/1/1	2014/2/1	2014/3/1	2014/4/1	2014/5/1	2014/6/1
1	Communication and meetings	2013/9/14	2013/10/1	2.3 w.										
2	Data collection and analysis	2013/9/24	2013/10/22	3.6 w.										
3	Analysis Task set	2013/10/22	2013/11/23	4.0 w.										
4	Design Task set	2013/11/24	2013/12/24	3.9 w.										
5	Prepare the environment	2014/1/15	2014/2/1	2.1 w.										
6	Middleware investigation and learning	2014/2/3	2014/2/24	2.7 w.										
7	Project implementation and coding	2014/2/25	2014/5/10	9.1 w.										
8	Project Testing and Modification	2014/5/10	2014/5/25	2.0 w.										
9	Documentantion	2014/5/1	2014/6/1	4.0 w.										

Figure 3.1: Project time line.

3.3 project methodology

Firstly, we will state the steps that will be followed to evaluate the project, and then we will discuss the available options in the project.

3.3.1 Flow of the project work

Firstly, we will evaluate the need of some facilities of PPU University to more powerful computational resources. This will be done by making meetings with the staff of these facilities.

Secondly ,we will evaluate the utilization percentage of some resources in the university labs during their work ,by installing some dedicated software programs to log

the CPU and memory usage at small scattered periods then we will averaging these values to get a more general term that represents the usage percentage of these resource.

Thirdly, we will study and make use of the previous studies to choose the most suitable middleware to use in our project.

Fourthly, we will use this middleware to build our grid on a sample of one or two labs. Finally we will provide a portal that users can load there computational jobs through. These jobs will be computed using the grid resources.

3.3.2 Options and analysis

➤ Middleware:

Middleware provides the tools that enable the various elements to participate in a grid. “The middleware layer is the brain behind a computing grid” [23], there are many middleware frameworks that can be used, some of them are:

- ❖ **Berkeley Open Interface for Network Computing (BOINC):** according to reference [24] we state the following description about BOINC:
 - BOINC is a set of software modules that enable the use of idle CPU cycles on a personal computer. It is an open source middleware platform for PRC. It provides one of the most powerful supercomputers in the world.
 - BOINC is a client-server architecture: the server generates tasks and distributes them to clients then collects their results. The following figure shows BOINC architecture.

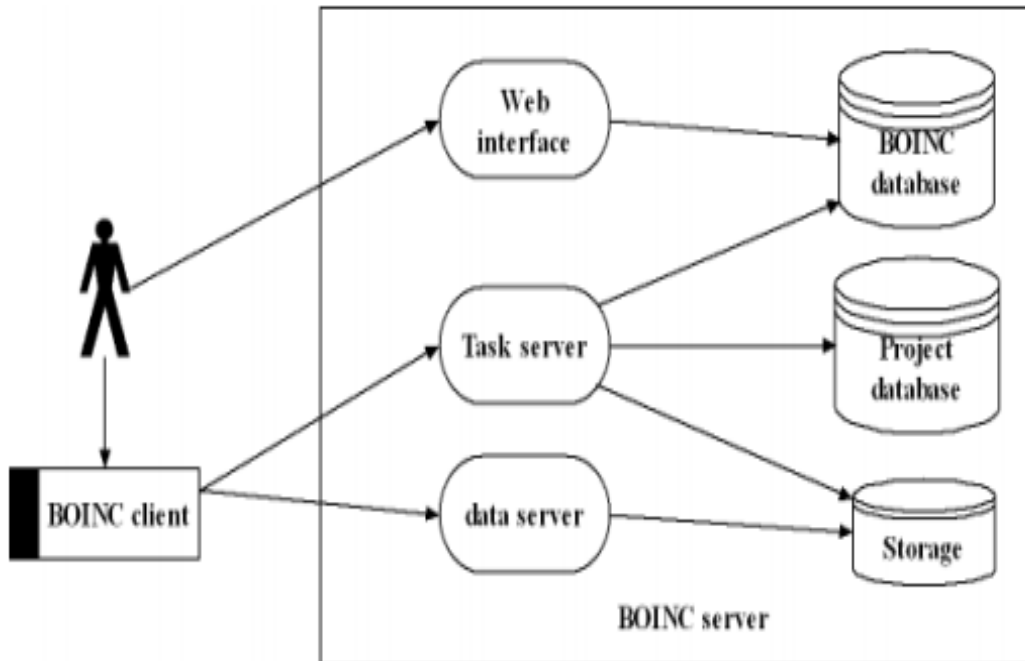


Figure 3.2: BOINC Architecture, Source [3].

❖ Entropia:

- Commercial product.
- Supports windows desktop grid system by aggregating desktop resources into a single logical resource.
- Depends on a central job manager that administers various desktop clients.
- Provides a centralized interface to manage all of the clients on the Entropia grid.

This description stated according to reference [3].

- ❖ Distributed.net:
 - Volunteer computing middleware.
 - Focuses only on two specific projects.
 - Its server code cannot be obtained and thus used for any other projects [3].

- ❖ Grid MP:
 - Commercial distributed computing software package.
 - It is centralized architecture: a Grid MP Service represents the manager. It accepts tasks from the user and schedules them on the resources having Grid MP agents. The Grid MP agents can be deployed on clusters. Grid MP agents receive tasks and execute them on resources, advertise their resource capabilities on Grid MP services and return results to the Grid MP services [25].

- ❖ Alchemi .NET: according to references [26, 27] we stated the following about Alchemi.
 - Dedicated to build PC Grid projects.
 - Layered architecture : as shown in Figure 3.2
 - Alchemi follows the master-worker parallel programming paradigm: central component dispatches independent tasks for parallel execution to workers and manages them.
 - Object oriented .NET programming environment.

- It was shut down after 2006 means that support is currently unavailable.
- Aneka is the successor of Alchemi: it is commercial package.

Alchemi.Net architecture is shown in the figure 3.3 below.

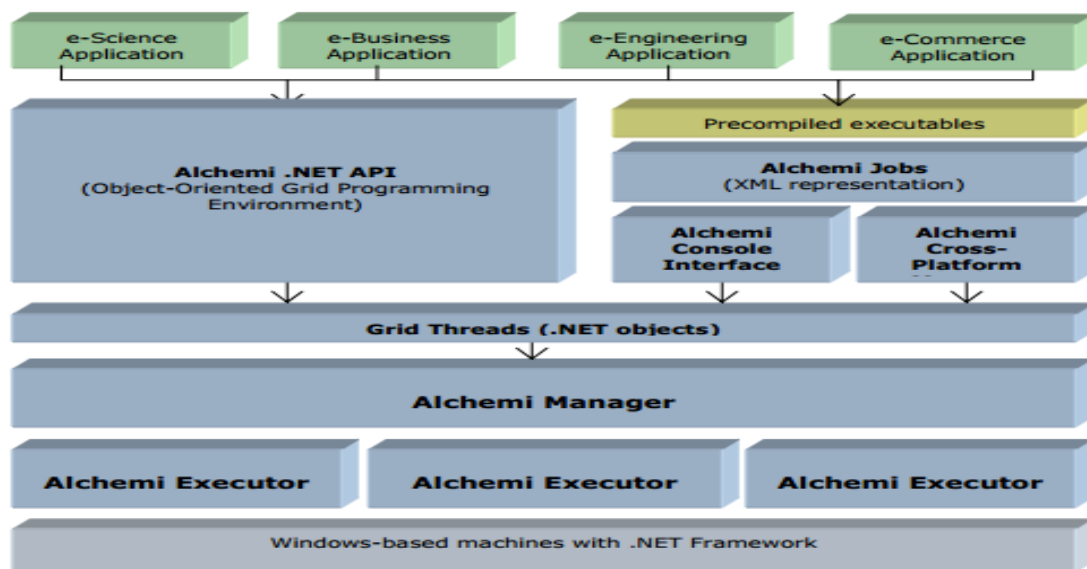


Figure 3.3: Alchemi layer architecture, Source [23].

All of the middleware frameworks that were stated previously are commercial products except BOINC and Alchemi .NET that were free software packages. BOINC provides many features that are related to our project.

- Operating system on the server machine:
 - ❖ Windows.

- ❖ Linux.

The choice between Linux and windows depends on the middleware framework because different middleware frameworks show different performance on different operating systems. In general Linux has an advantage over windows which is providing more security levels. Security is very important characteristic to the grid system.

- Grid hosts: lab PCs that will be used as computing resources of the grid.
 - ❖ The grid hosts will be one of the labs of university. These labs may differ in their utilization percentage and the performance of their machines in addition to the availability and authorization issues.

3.2.3 Risk Management

The following table states the risks that may arise during the project implementation.

Table 3.1: Risk Management

Risks	Category	Probability	Impact
Lack of experience	Staff experience	70%	critical
Going beyond Schedule (late delivery)	Schedule risk	50%	marginal
Lack of testing experience	Performance risk	40%	critical
Low staff productivity	Schedule risk	35%	marginal
Lack of training on tools	Development environment	30%	negligible
Loss of team member	Staff size	20%	critical
Inaccurate Cost estimations	Cost risk.	15%	Marginal

Each of these risks has its own risk information sheet that clarifies the risk and show how to deal with it when happening. All of these sheets are shown in the following tables:

Table 3.2: lack of experience information sheet.

Risk Information Sheet		
Project Name < Grid Computing In PPU >		
ID <risk id:PO1>	lack of experience	Identified <12-10-2013>
Impact < high>	Risk Statement(Description)	
Probability <70%>	Lack of experience with network management, security and web programming may affect on the performance of the final software.	
<p>Context :</p> <p><u>Subcondition1:</u></p> <p>The system requires a high experience in network administration and management in addition to Linux experience.</p> <p><u>Subcondition2:</u></p> <p>Also a good web programming experience is needed in this project.</p>		
<p>Mitigation strategy</p> <ol style="list-style-type: none"> 1- The team adequate training on website design and such programming language. 2- An intensive work on server and network configuration with Linux scripts to get better experience which is very important to be able to build the grid. 		
<p>Contingency plan and trigger</p> <ul style="list-style-type: none"> - Taking intensive training courses in network configuration and web programming. 		
<p>Status / date</p> <p><1-1-2014>: Mitigation steps initiated.</p>		
Originator: Muhammad Dwaib		Assigned: Ibrahim Qdemat

Table 3.3: Late delivery information sheet.

Risk Information Sheet	
Project Name < Grid Computing In PPU >	
ID <risk id:PO2>	Late delivery Identified <16-10-2013>
Impact <high>	Risk Statement(Description) The degree of uncertainty that the project schedule will be maintained and the product will be delivered on time.
Probability <50%>	
<p>Context</p> <p>Subcondition1: poor experience of time estimation may cause an unrealistic end date</p> <p>Subcondition2: Ambiguity in the system requirements.</p> <p>Subcondition3: A need to unavailable software.</p> <p>Subcondition4: Lack of effective project team integration and project assembling problems</p>	
<p>Mitigation strategy:</p> <ol style="list-style-type: none"> 1- Define the scope accurately. 2- Create a realistic and achievable schedule. Some level of risk analysis is required. 3- Analyze risks and adjust the schedule. 	
<p>Contingency plan and trigger:</p> <ol style="list-style-type: none"> 1- Abandon (give up) some additional features that does not affect the whole system. 2- Increase the team productivity. 3- Evaluate the basic functional requirements of system. 	
<p>Status / Date</p> <p><During first semester>: Mitigation steps initiated.</p>	
Originator: Ibrahim Qdemat	Assigned: Muhammad Dwaib

Table 3.4: Lack of testing experience information sheet.

Risk Information Sheet		
Project Name < Grid Computing In PPU >		
ID <risk id:PO6>	Lack of testing experience	Identified <12-10-2013>
Impact < high>	Risk Statement(Description)	
Probability <40%>	This project may need to a good knowledge about quality assurance to be able to track the system and find problems to solve.	
Context : <u>Subcondition:</u> Many tests may be required to examine the overall system performance and to detect any problem that need to be solved.		
Mitigation strategy 1- Take a help from experts of quality assurance.		
Contingency plan and trigger 2- Offer the system to the quality assurance testers to help in problems detection.		
Status / date <10-10-2013>: Mitigation steps initiated.		
Originator: Ibrahim Qdemat		Assigned: Muhammad Dwaib

Table 3.5: low staff productivity information sheet.

Risk Information Sheet		
Project Name < Grid Computing In PPU >		
ID <risk id:PO3>	low staff productivity	Identified <14-10-2013>
Impact < high>	Risk Statement(Description)	
Probability <35%>	Project team may need longer time than expected to perform the required work.	
Context :		
<u>Subcondition1:</u>		
The project team may not be familiar with the project environment as fast as expected.		
<u>Subcondition2:</u>		
The project team may need to intensive efforts to be able to perform the work.		
Mitigation strategy		
<ul style="list-style-type: none"> - The team determine a period of time before the evaluation work start for training on website design and network management on Linux operating systems to get better experience which is very important to be able to build the grid. 		
Contingency plan and trigger		
Taking intensive training courses in network management and web programming.		
Status / date		
<1-1-2014>: Mitigation steps initiated.		
Originator: Muhammad Dwaib		Assigned: Ibrahim Qdemat

Table 3.6: Lack of training on tools information sheet.

Risk Information Sheet		
Project Name < Grid Computing In PPU >		
ID <risk id:PO5>	Lack of training on tools	Identified <12-10-2013>
Impact < high>	Risk Statement(Description)	
Probability <30%>	To perform this project; team members may need to deal with new tools which they don't have any previous experience with.	
<p>Context :</p> <p><u>Subcondition:</u></p> <p>The system requires a good knowledge on dealing with many software tools, like some middleware frameworks. The project team may need to work on some tools that never deals with previously</p>		
<p>Mitigation strategy</p> <p>1- Try to identify all the needed tools in the project to take an experience courses with these tools.</p>		
<p>Contingency plan and trigger</p> <p>2- Intensifies work on these tools with some experiments.</p>		
<p>Status / date</p> <p><10-10-2013>: Mitigation steps initiated.</p>		
Originator: Ibrahim Qdemat		Assigned: Muhammad Dwaib

Table 3.7: Loss of team member information sheet.

Risk Information Sheet		
Project Name < Grid Computing In PPU >		
ID <risk id:PO4>	Loss of team member	Identified <12-10-2013>
Impact < high>	Risk Statement(Description)	
Probability <20%>	The team consists of two students only, so any loss of members will be so critical.	
<p>Context :</p> <p><u>Subcondition:</u></p> <p>This project may need three members to be manipulated well. Since the team members are two only so they have to work intensively to evaluate this project correctly, and any loss of project team may cause unexpected results on the project.</p> <p>We have two types of loss of team member :</p> <ol style="list-style-type: none"> 1- Temporary absent which may come from illness or any other causes. 2- Permanent absent which may come from death. 		
<p>Mitigation strategy:</p> <p>Really this type of risk doesn't have any mitigation strategies.</p>		
<p>Contingency plan and trigger:</p> <ul style="list-style-type: none"> - Permanent absence: Refine the project scope. - Temporally absence: Try to work overtime and intensively to perform the work. 		
<p>Status / date</p> <p><1-1-2014>: Mitigation steps initiated.</p>		
Originator: Muhammad Dwaib		Assigned: Ibrahim Qdemat

Table 3.8: Inaccurate cost estimations information sheet.

Risk Information Sheet		
Project Name < Grid Computing In PPU >		
ID <risk id:PO7>	Inaccurate cost estimations	Identified <12-10-2013>
Impact < high>	Risk Statement(Description)	
Probability <15%>	Cost estimations may be determined inaccurately.	
<p>Context :</p> <p><u>Subcondition1:</u></p> <p>Lack of experience on costs estimations or an ambiguity in the required components may lead to inaccurate project estimation.</p>		
<p>Mitigation strategy</p> <ul style="list-style-type: none"> - Try to clarify all of the project requirements. 		
<p>Contingency plan and trigger:</p> <ul style="list-style-type: none"> - Search for free alternatives. - Search for financial support. 		
<p>Status / date</p> <p><10-10-2013>: Mitigation steps initiated.</p>		
Originator: Muhammad Dwaib		Assigned: Ibrahim Qdemat

3.4 Project components and resource cost estimations

This project aims to operate a grid computing system with minimal costs. Basically the cost is represented by three aspects, which are the hardware components, software components and human resources. These components are stated in the following subsections.

3.4.1 Hardware Components

Grid computing consists of basic HW components that represented as:

- Computational and storage resources:
 - Grid server: a certain computer will work as a grid server, it is cost will be approximately 1000\$. No need to use a dedicated hardware server in this project.
 - Grid clients: These are all the computers in the labs that will participate in the grid system.
- Communication:
 - Network: The network will connect the different components of the grid together.

These components are summarized in the following table:

Table 3.9: Hardware Components.

Component	Price
Grid server	1000\$
Grid clients	Available
Network	Available

3.4.2 Software components

- Middleware: as stated previously there are many options of middleware frameworks to work on, the preferred middleware is the free one. So the software components may also have no costs.
- PHP editor: for web page design.
- Mysql DB server: to build the needed DB for the project.
- Apache server: which is http server used to generate web pages as a response for users requests.

These components are summarized in the following table:

Table 3.10: Software Components.

Component	Price
Ubuntu 12.04 Linux distribution	Free
Middleware	Free
PHP editor	Free
MySQL DB server	Free
Apache server	Free
Shell scripting	Free

3.4.3 Human resource costs

- The team of this project consists of two under graduation computer system engineering (CSE) students.

3.5 Summary

In this chapter we defined all the project task sets then we generated the project Gantt chart. We briefly described how the work will go during the project manipulation, and then we stated the available options to use in the project implementation. Some of the project risks were stated and analyzed. Finally, we defined all the components that are needed for the project evaluation in addition to the cost estimations.

Chapter Four

Software Requirements Specification

4.1 Overview

In this chapter we clarify the system requirements by identifying the possible actors and their expected interactions with the system. Scenarios and use-case diagrams are used to describe the system actors and their interaction with the system. This chapter also presents the class responsibilities collaborator modeling that contains the functions and attributes of each class and its helping classes. Finally, it states the hierarchies of classes and their relationships.

4.2 Requirements Description

In this section we show all possible system's actors and describe the system requirements in terms of scenarios and use-case diagrams.

4.2.1 System Actors

There are four possible actors. Here we briefly describe them:

- Grid Administrator :

This is the person that has the absolute control over the grid system and its resources. He manages, maintains and monitors the grid and its overall functionalities.

- Grid User :

This is the end user of the grid system who can log into his account and upload his jobs that is to be performed on the grid resources and download results

back. He also can delete the results, abort jobs execution and modify his account information.

- **Grid System :**

This is the main actor, consists of the middleware and the configurations to evaluate the system. It is responsible about utilizing available resources, scheduling jobs and monitoring their execution, validate results and prepare them for download.

- **Grid Client :**

This represents the computer machines at the computer labs. These machines will not be involved in decision making; they are forced to perform submitted jobs in their idle time and return the results back.

4.2.2 Use-case templates

The interaction between the possible actors and the grid system can be clarified using expected scenarios; here we provide all expected scenarios that will be initiated by the system actors.

Grid administrator:

There are eight use-cases that are initiated by the grid administrator. These use-cases are:

- 1- Creating/deleting projects.
- 2- Adding/deleting users.
- 3- Attaching/deleting resources.
- 4- Updating resources.
- 5- Stopping/starting/restarting project.
- 6- Modifying user information.
- 7- Managing computing preferences.
- 8- Monitoring project status.

Analysis for each use-case:

Each use-case analyzed by use-case template as follow:

Table 4.1: Creating/deleting project use case template.

Use case template for creating/deleting project:
Use case: <ul style="list-style-type: none">- Creating/deleting projects.
Primary actor: <ul style="list-style-type: none">- Administrator.
Goal: <ul style="list-style-type: none">- Adding new project or deleting an existing one.
Precondition: <ul style="list-style-type: none">- Verifying the administrator authenticity.
Trigger: <ul style="list-style-type: none">- Administrator clicks on delete button.
Scenario: <ul style="list-style-type: none">- Creating project: Administrator login to the system, goes to project creation part, enter the project name, select the project type and finally click on create project button.- Deleting project: Administrator login to the system, select one of project at the system, click on delete button.
Exception: <ul style="list-style-type: none">- Connection loses during the process.

Table 4.2: Adding/deleting user use case template.

Use case template for Adding/deleting user:
Use case: Adding/deleting users.
Primary actor: Administrator.
Goal: <ul style="list-style-type: none">- Adding a new user to the grid-users.- Or deleting a specific user from the grid-users.
Precondition: <ul style="list-style-type: none">- Verifying the administrator authenticity.
Trigger: <ul style="list-style-type: none">- Select adding a new user, or select one user to delete.
Scenario: <ul style="list-style-type: none">- Adding new user: A specific request for adding or deleting user received by the administrator, admin enters the user information and click add button.- Deleting user: Administrator select a user and click on delete button.
Exception: <ul style="list-style-type: none">- Losing connection during process.

Table 4.3: Attaching/deleting resource use case template.

Use case template for Attaching/deleting resource:
Use case: Attaching/deleting resources.
Primary actor: Administrator.
Goal: <ul style="list-style-type: none">- Attaching a resource to a project.- Or deleting a resource from serving a project.
Precondition: <ul style="list-style-type: none">- Verifying the administrator authenticity.
Trigger: <ul style="list-style-type: none">- Administrator decides to add or remove some resources from serving the project.
Scenario: <ul style="list-style-type: none">- Admin select a project- Enters the resource IP address.- Click on attach/delete button to attach or delete the resource.

Table 4.4: Update resource use case template.

Use case template for Update resource :
Use case: Update resource.
Primary actor: Administrator.
Goal: <ul style="list-style-type: none">- Refresh the connection between the resource and the project.
Precondition: <ul style="list-style-type: none">- Verifying the administrator authenticity
Trigger: Administrator select update resource function.
Scenario: <ul style="list-style-type: none">- Administrator logs into the grid portal.- Select a project.- Enter the resource IP address.- Click on the update function.

Table 4.5: Stopping/starting/restarting project use case template.

Use case template for Stopping/starting/restarting project:

Use case: Stopping/starting/restarting project.

Primary actor: Administrator.

Goal:

- Control the project status.

Precondition:

- Verifying the administrator authenticity.

Trigger:

- Administrator select stop/start/restart function.

Scenario:

- Admin select a project
- Click on the button start to start running a stopped project.
- Click on the button stop to stop running a project.
- Click on the button restart to restart the project running; needed for the cases when some diamonds are stopped while the project is running.

Table 4.6: Modifying user information use case template.

Use case template for Modifying user information :

Use case: Modifying user information.

Primary actor: Administrator.

Goal:

- Modify user information.

Precondition:

- Verifying the administrator authenticity.

Trigger:

- Select a specific user to modify his information.

Scenario:

- Admin show system users.
- Select one of them.
- Show his information.
- Modify the information and click on update information button.

Table 4.7: Managing computing preferences use case template.

Use case template for Managing computing preferences:

Use case: Managing computing preferences.

Primary actor: Administrator.

Goal:

- Control the resource usage of CPU, memory and other preferences.

Precondition:

- Verifying the administrator authenticity.
- Verifying the project account authenticity.

Trigger:

- Administrator selects computing preferences.

Scenario:

- Admin select a project
- Click on the computing preferences link.
- Modify the computing preferences and click on save button.

Table 4.8: Monitoring project status use case template.

Use case template for Monitoring project status :
Use case: Monitoring project status.
Primary actor: Administrator.
Goal: <ul style="list-style-type: none">- Monitor the project status.
Precondition: <ul style="list-style-type: none">- Verifying the administrator authenticity.
Trigger: <ul style="list-style-type: none">- Administrator click on the project status link.
Scenario: <ul style="list-style-type: none">- Admin select a project- Click on the project status link.

Grid-user:

Grid-user is responsible about initiating some use-cases in the system. These use cases are:

- 1- Upload a new job.
- 2- Monitor jobs execution status.
- 3- Abort jobs while it is in progress state.
- 4- Download the results of completed jobs.
- 5- Delete the results of completed jobs.
- 6- Modify account information.

Analysis for each use-case using use-case template:

Table 4.9: Upload a new job use case template.

Use-case template for Upload a new job:
Use case: Upload a new job.
Primary actor: User.
Goal: uploading new job for execution on the grid system.
Precondition: Verifying the user authenticity.
Trigger: a user open Job Submission page to submit a new job.
Scenario: <ol style="list-style-type: none">1- A user open job submission page.2- A user selects a job for uploading from his device.3- A user selects the platform for executing the job.4- Enters any arguments if needed by the executed job.5- Click on upload button.
Exception: <ul style="list-style-type: none">- A user selects invalid job type for submission.- A user doesn't select the platform.

Table 4.10: Monitor jobs execution status use case template.

Use-case template for Monitor jobs execution status:
Use case: Monitor jobs execution status.
Primary actor: User.
Goal: <ul style="list-style-type: none">- Monitoring jobs execution status.
Precondition: Verifying the user authenticity.
Trigger: <ul style="list-style-type: none">- User login to the system.- User goes the home page.
Scenario: <ol style="list-style-type: none">1- A user login to the home page.2- All uploaded jobs displayed at this page with their status.

Table 4.11: Abort jobs while it is in progress state.

Use-case template for Abort jobs while it is in progress state:
Use case: Abort jobs while it is in progress state.
Primary actor: User.
Goal: Aborting job execution.
Precondition: <ul style="list-style-type: none">- Verifying the user authenticity.- A user has a job in the progress status.
Trigger: A user click on abort button.
Scenario: <ol style="list-style-type: none">1- User enters to the portal.2- Show all his jobs.3- Click on abort button for one of in progress status job.
Exception: <ul style="list-style-type: none">- Display the status in progress for a job having a status not in progress; since the page is not updated such that the real status of the job is changed and the effect doesn't appear yet on the portal.

Table 4.12: Download the results of completed jobs.

Use-case template for Download the results of completed jobs
Use case: Download the results of completed jobs.
Primary actor: User.
Goal: Download the results of completed jobs.
Precondition: <ul style="list-style-type: none">- Verifying the user authenticity.- A user has a result for his uploaded job.
Trigger: A user click on download link.
Scenario: <ul style="list-style-type: none">4- User enters to the portal.5- Show all his jobs.6- Click on download link for a job result.
Exception: <ul style="list-style-type: none">- Result folder was deleted by the system or the server admin.

Table 4.13: Delete the results of completed jobs.

Use-case template for Delete the results of completed jobs
Use case: Delete the results of completed jobs.
Primary actor: User.
Goal: Delete the results of completed jobs.
Precondition: <ul style="list-style-type: none">- Verifying the user authenticity.- A user has a result for his uploaded job.
Trigger: A user click on download link.
Scenario: <ul style="list-style-type: none">7- User enters to the portal.8- Show all his jobs.9- Select jobs for deletion.10- Clock on delete button.
Exception: <ul style="list-style-type: none">- Result folder was deleted by the system or the server admin.

Table 4.14: Modify account information.

Use-case template for Modifying account information.
Use case: Modify account information.
Primary actor: User.
Goal: Modify account information.
Precondition: - Verifying the user authenticity.
Trigger: A user goes to "my Account" page.
Scenario: 11- A user enters to the portal. 12- Goes to "my account" page. 13- A user modifies his account information. 14- Click on modify information button.

Grid System:

There are four use-cases that are initiated by the grid system actor. These use-cases are:

- 1- Generating instance of job.
- 2- Sending instance to a client.
- 3- Validating results.
- 4- Preparing results for download by users.

Analysis for each use-case:

Each use-case analyzed by use-case template as follow:

Table 4.15: Generate instance of job use case template.

Use case template for Generate instance of job :
Use case: <ul style="list-style-type: none">- Generate instance of job.
Primary actor: <ul style="list-style-type: none">- System.
Goal: <ul style="list-style-type: none">- Generate instance of job for executing on one of the computing resources.
Precondition: <ul style="list-style-type: none">- Valid executable job submitted by one of the grid users.
Trigger: <ul style="list-style-type: none">- Job is submitted by user.
Scenario: <ul style="list-style-type: none">- System handles a new submitted job or handles fail of the previous instance of the job.- System generate instance of the job for execution on one of the computing resources.
Exception: <ul style="list-style-type: none">- Lose the result before arriving to the server. The job will fall in the status "in progress on host...". The solution here is just aborting the job by the user and submits the job again.

Table 4.16: Send instance to a client use case template.

Use case template for Send instance to a client:
Use case: Send instance to a client.
Primary actor: System.
Goal: <ul style="list-style-type: none">- Send the job instance for execution on a client.
Precondition: <ul style="list-style-type: none">- Valid executable job submitted by one of the grid users.- System generated an instance of the job for sending to a client.
Trigger: <ul style="list-style-type: none">- New instance of the job is available at the system.
Scenario: <ul style="list-style-type: none">- System handle new instance of the job waiting to submit to a client for execution.- System selects the available client and sends the instance to.
Exception: <ul style="list-style-type: none">- No clients are currently available.- No client has the suitable platform to execute the job.

Table 4.17: Validate results use case template.

Use case template for Validate results:
Use case: Validate results.
Primary actor: System.
Goal: <ul style="list-style-type: none">- Check the validity of the results.
Precondition: <ul style="list-style-type: none">- Job is executed by one of the clients or aborted while executing.
Trigger: <ul style="list-style-type: none">- Client uploads results of execution of the Job.
Scenario: <ul style="list-style-type: none">- Job is executed by a client.- A connection between the client and server is activated.- Result is uploaded.- System checks the validity of the results by a pre-determined mechanism.

Table 4.18: Prepare results for download by users.

Use case template for Prepare results for download by users :
Use case: Prepare results for download by users.
Primary actor: System.
Goal: <ul style="list-style-type: none"> - Prepare the result for download by the user.
Precondition: <ul style="list-style-type: none"> - Job execution is completed.
Trigger: a user requires downloading the completed job.
Scenario: <ul style="list-style-type: none"> - A user logs into the grid portal. - Select a completed job for download. - System does the appropriate compression mechanism to prepare the job for download.

Grid Client:

This actor has the Job execution scenario.

- Job-Execution :
When a job is to be executed the grid client machine and the grid server are involved in this process. They cooperate with each other to achieve the job.

The following template shows how the scenario happens:

Table 4.19: Job Execution use case template.

Use case template for Job Execution:
Use case: Job Execution.
Primary actors: Grid Client, Grid System (Server).
Goal: <ul style="list-style-type: none">- Executing an uploaded job on one of available computing resource (machine).
Precondition: <ul style="list-style-type: none">- The Job is submitted to the project and its associated input and output files are determined so the project is ready to execute.
Trigger: <ul style="list-style-type: none">- Job is added to a running project.- It is on the top of the server scheduler.
Scenario: <ul style="list-style-type: none">- The unutilized Grid Client (computer machine) gets a set of tasks from the project's scheduling server.- Grid Client downloads executable and input files from the project's data server.- Grid Client runs the application programs (jobs), producing output files.- Grid Client uploads the output files to the data server.- After a period of time the grid client reports the completed tasks to the scheduling server, and gets new tasks.- This cycle is repeated indefinitely while project running.

4.2.3 Use-case Diagrams

This subsection clarifies the interaction between actors and the grid system, as shown in figures below.

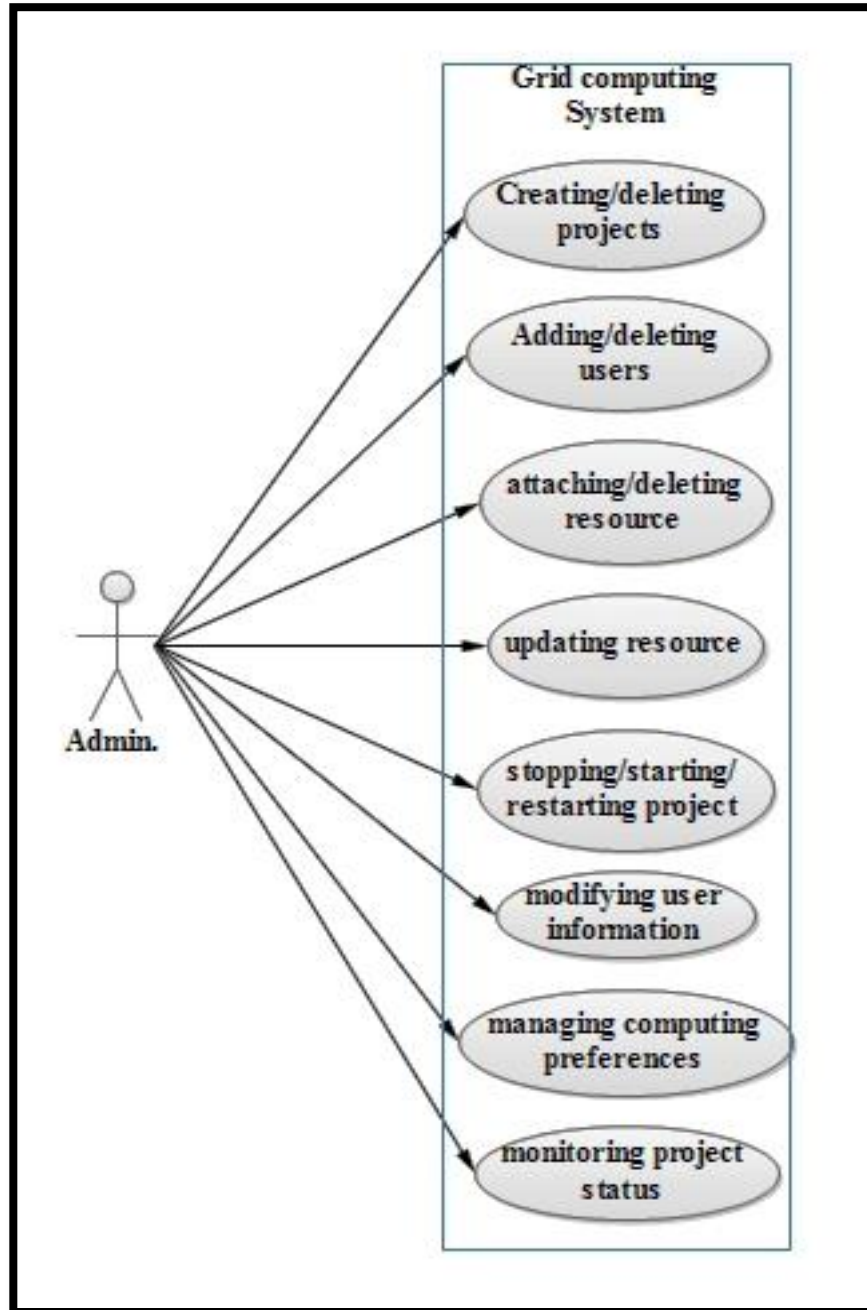


Figure 4.1: Admin use-case diagram.

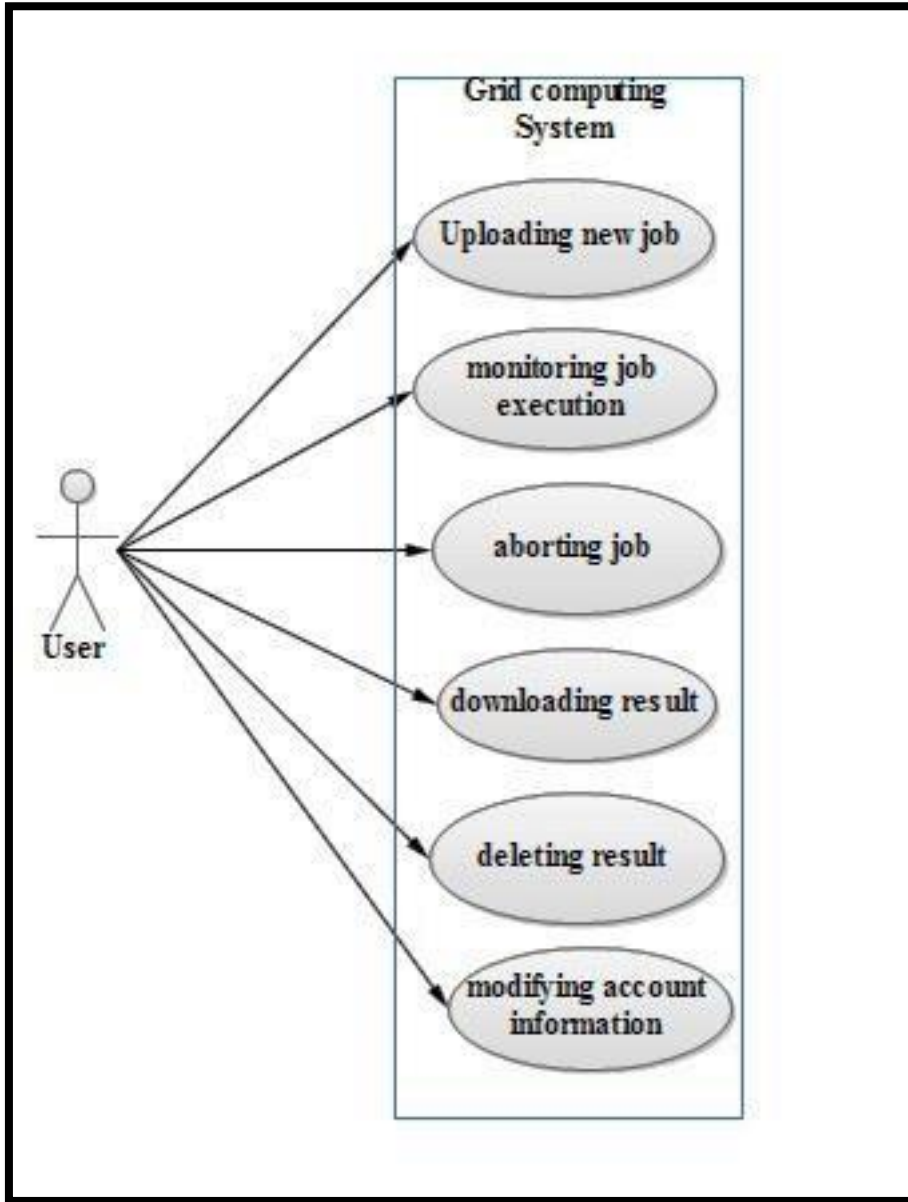


Figure 4.2: User use-case diagram.

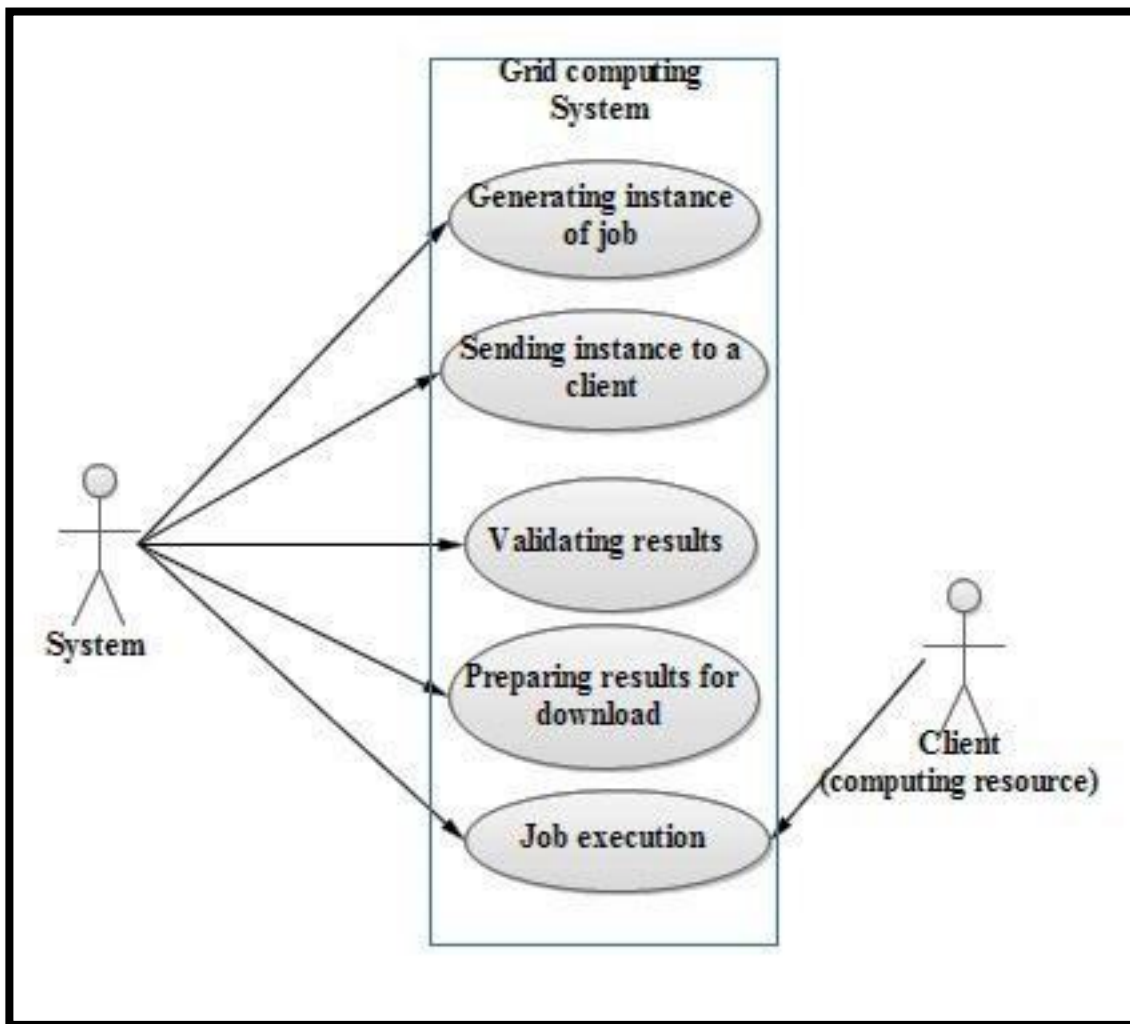


Figure 4.3: system and client use-case diagram.

4.3 Class-Responsibility-Collaborator Modeling (CRC)

This section identifies the classes required in the project implementation. Also, it provides a simple mean about the functionality of each class, and the class relationships. All the classes are represented using CRC modeling as follow:

- 1) DBConnection class:

Table 4.20: DBConnection CRC.

Class : DBConnection	
Description: this class has the responsibility of creating and managing connections with DB's.	
Responsibility:	Collaborator:
Setting the connection with DB following to a specific user at a specific server.	ProjectCreation class. ProjectManagment class.
Getting a DB connection.	UserCreation class.

- 2) ProjectCreation Class:

Table 4.21: ProjectCreation CRC.

Class : ProjectCreation	
Description: this class has the responsibility of creating different types of projects.	
Responsibility:	Collaborator:
Generate the form to enable the admin enter the inputs needed for project creation.	DBConnection class.
Validate the project name.	
Create project.	

3) ProjectManagment Class:

Table 4.22: ProjectManagment CRC.

Class : ProjectManagment	
Description: this class has the responsibility of managing the grid system project.	
Responsibility:	Collaborator:
Stop project.	DBConnection class.
Start project.	
Restart project.	
Attach clients.	
Delete clients.	
Update client.	
Validate IP.	
Delete project.	
Generate the form.	

4) UserCreation class:

Table 4.23: UserCreation CRC.

Class : UserCreation	
Description: this class has the responsibility of providing the functionality for adding new users to the system.	
Responsibility:	Collaborator:
Generate the form to enable adding new users.	DBConnection class.
Validate the new user name.	
Validate the new user email.	
Validate the new user password.	
Adding the new user.	

5) JobSubmission class:

Table 4.24: JobSubmission CRC.

Class : JobSubmission	
Description: this class has the responsibility of providing the functionality for submitting new jobs.	
Responsibility:	Collaborator:
Generate the form to enable submitting new jobs.	DBConnection class.
Validate the type of the job folder.	
Check all inputs.	
Uploading the job folder.	
Extracting the folder.	
Submitting job to a specific project.	
Adding the job to the DB	

4.4 Class Hierarchies and relationships

This section shows the relations between classes using class hierarchy and relationship diagram as shown in figure 4.4:

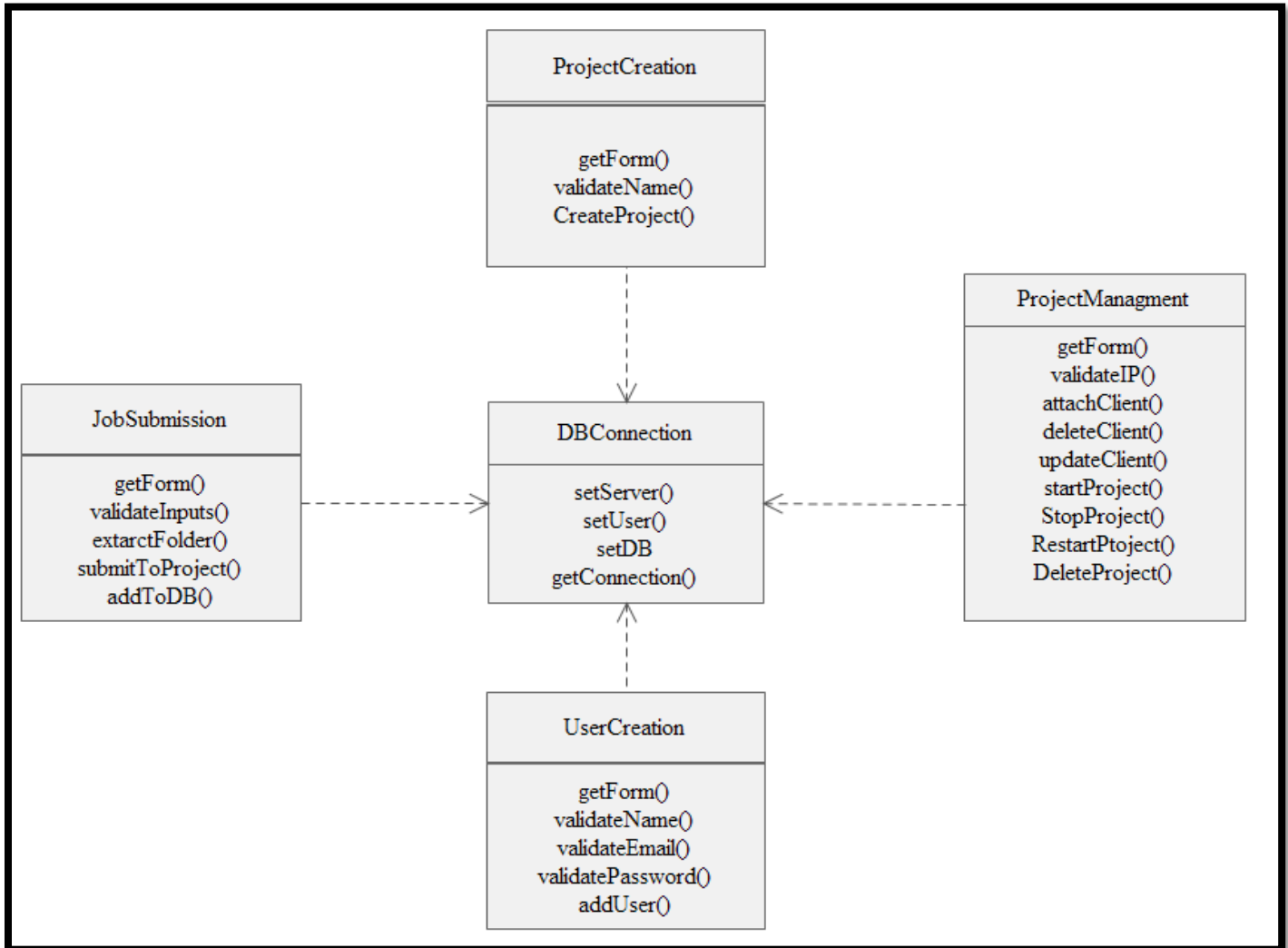


Figure 4.4: Class Hierarchies and relationships diagrams.

More details about this section will be discussed in chapter five.

4.5 Summary

System requirements were introduced in this chapter by defining system actors, use-cases templates, use-cases diagrams, CRC modeling and the class hierarchy and relationship. System actors defined as four basic actors, those actors are: Administrator, User, System and Grid Client (computing resource). All scenarios that may occur introduced in the use case templates. Each actor can initiate some of use-cases which clarified in use-case diagrams. At CRC modeling we define five classes which are system, resources, projects, jobs generator and result handler. Finally, class hierarchy and relationship clarify the relations between the classes in the system.

Chapter 5

Software design description

5.1 Overview

This chapter describes the overall system design. It describes the lower level of the system which is the BOINC middleware, then it goes in more details in describing the higher level of the system.

5.2 BOINC Middleware

Berkeley Open Interface for Network Computing (BOINC) is an open source middleware platform. BOINC has been developed by a team based at the Space Sciences Laboratory (SSL) at the University of California, Berkeley led by David Anderson, who also leads SETI@Home. SETI@HOME tries to find extraterrestrial life by analyzing radio signals [29].

BOINC is designed to be a free structure for anyone wishing to start a volunteer computing project. It is a set of software modules that enable the use of idle CPU cycles on a personal computer to do scientific computing.

BOINC is the most popular desktop grid systems today with the aggregated computational power of more than 2,576,332 participants is about 8,361.840 Teraflops, thus providing one of the most powerful —supercomputers in the world [30, 31].

5.2.1 BOINC Architecture

The structure of BOINC is simple. BOINC follows the client-server architecture; the server generates work units (WUs), distributes them to clients and collects their

results. Each PC, acting as a client, communicates with the server to get WUs which include executables and input files and return results of computation, BOINC is not peer-to-peer; the clients do not communicate with each other [32]. These components are shown in Fig. (5.1) and are explained below.

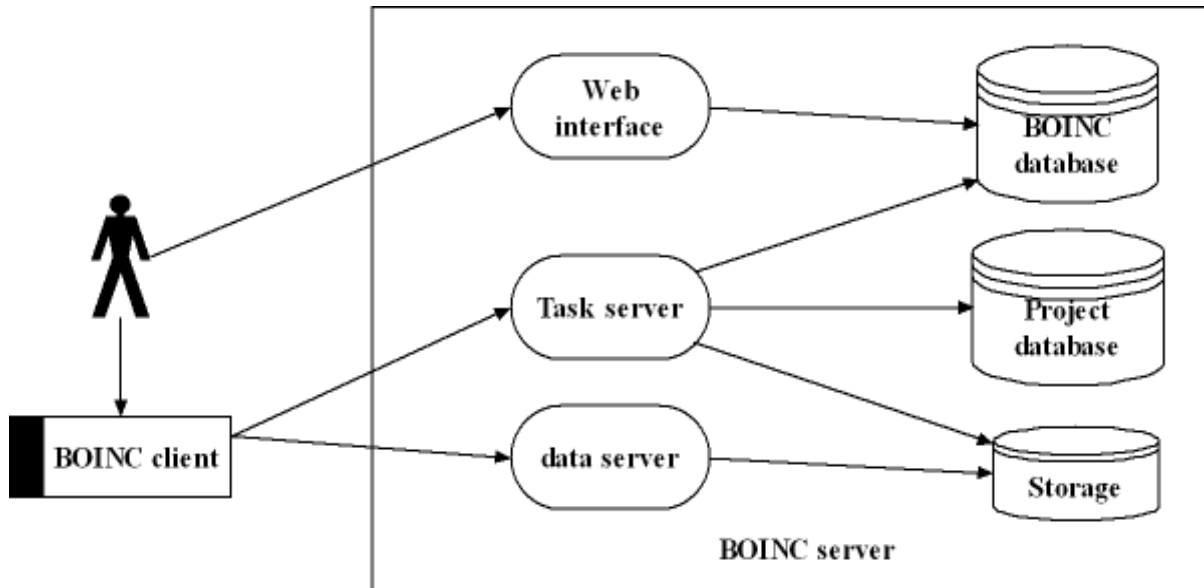


Figure 5.1: BOINC architecture [3].

5.2.2 BOINC Server

BOINC server runs on Linux and uses Apache, PHP, and MySQL as a basis for its web and database systems, which easily scales to projects of any size. BOINC server consists of the following components:

- **Web interfaces:** uses Apache web server and PHP, for user account and team management, message boards, current server status, and other features.
- **The task server:** creates tasks, posts them to clients, and processes returned tasks.
- **The data server:** downloads input files and executables, and uploads output file, uses HTTP protocol.

5.2.3 BOINC Client

Four components construct the BOINC client: core client, manager, screensaver, and command line tool. These components are shown in Fig. 5.2 below.

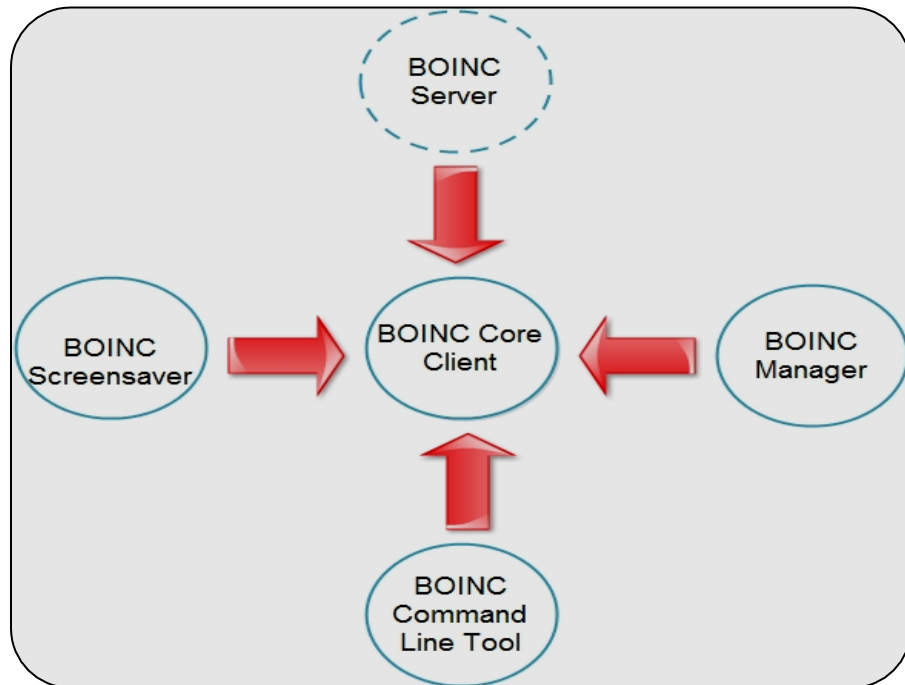


Figure 5.2: BOINC Client.

- 1- The core client takes care of scheduling among jobs from different projects, possibly preempting jobs, downloading and uploading results to the different projects to which it is attached [1].
- 2- BOINC manager enables control over the core client via user preferences. Some of these preferences are project specific and some are not. As a project specific preference the user can set a minimum and a maximum amount of work the client should keep on the computer for the given project [1].
- 3- The screensaver displays application specific graphics as a screensaver just to attract and entertain volunteer users in the first place [1].

- 4- BOINC command tools provide non-GUI version of BOINC manager [1]. The client is available for Windows and Linux on Intel x86 architectures, for Mac OS X on PowerPC, and Solaris on Sparc architectures, but since it is open source it should not be too difficult to get it running on other platforms as well.

5.2.4 BOINC Database

A MySQL database stores all information relevant to the BOINC project. This includes information about registered users and their associated hosts, about applications and application versions, about BOINC core clients and the versions involved and of course about WUs and their associated results. Basically the entire state of the server is stored in this database and queried by among others the above mentioned daemons [1].

5.3 System configuration design

This section includes a description for installing BOINC server software, creating and running a BOINC Project and deploying BOINC clients.

5.3.1 Installing BOINC Server Software

This involves setting up BOINC software, and other required open-source software, on a Linux computer. The software requirements installation process is shown in figure 5.3.

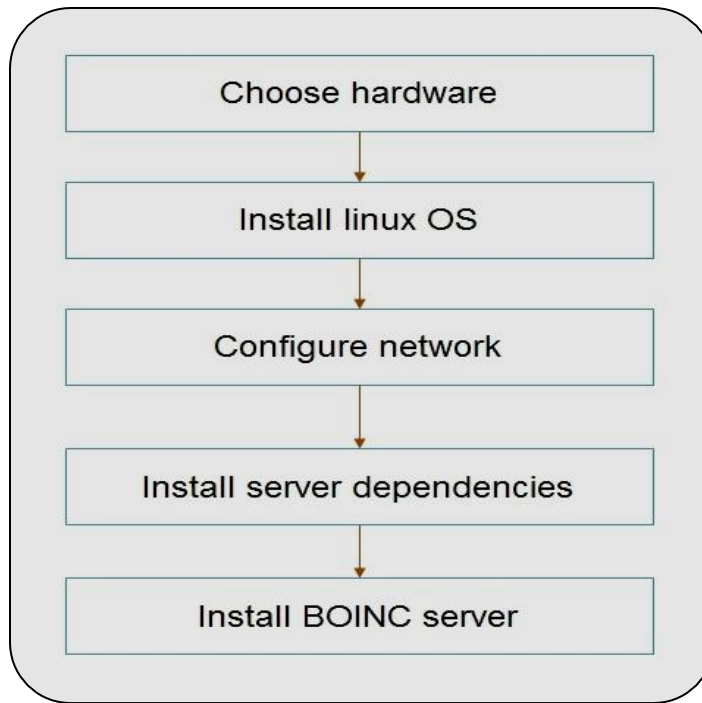


Figure 5.3: Installing BOINC Server Software.

1. Hardware

For experimentation and debugging, you can use almost any computer as a BOINC server. For better performance, availability, and security, the server needs the requirements [54]:

- The Internet connection should have adequate performance and reliability. The server must have a static IP address.
- The server should have good CPU speed (dual Xeon or Opteron), at least 2 GB of RAM, and at least 40 GB of free disk space. For a high-traffic project, use a machine with 8 GB of RAM or more and 64-bit processors.
- Other requirements to make it highly reliable (UPS power supply, RAID disk configuration, hot-swappable spares, temperature-controlled machine room, etc.).

2. Operating System installation:

BOINC server runs almost on any up-to-date UNIX or Linux variant machine so we can use for example Ubuntu Linux distribution.

3. Network configuration:

The server should be given a static IP out of the range of IPs available in the university.

4. BOINC Server Dependencies Installation:

BOINC server needs the following components and prerequisites to run:

- make 3.79+, m4 1.4+, libtool 1.5+
- autoconf 2.58+, automake 1.8+, GCC 3.0.4+ pkg-config 0.15+
- Python 2.2+ with MySQL DB module 0.9.2+
- MySQL 4.0.9 or higher (with mysql-dev(el), and mysql-client)
- SQLite 3.1 or higher (packages sqlite-dev(el) and SQLite)
- Apache web server with mod_ssl and PHP5+
- PHP5 with cli support and the GD and MySQL modules (packages php5-cli and php5-gd)
- OpenSSL version 0.9.8+

5.3.2 Creating And Running A BOINC Project

A BOINC project is the environment under which the grid application runs. Project creation block diagram is shown in figure 5.4.

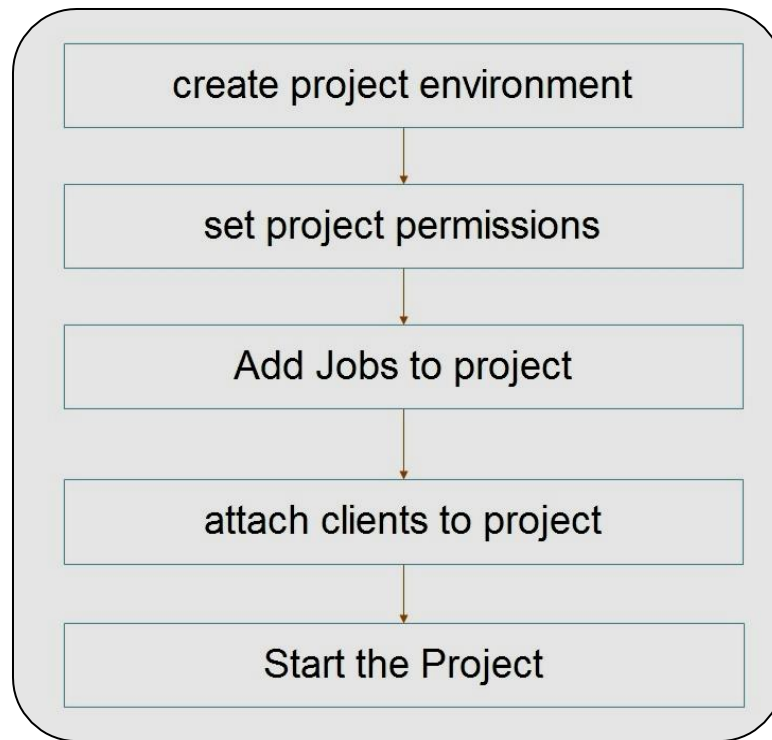


Figure 5.4: Creating And Running A BOINC Project.

- **Template Files for applications**

In order to run the application, its input and output files must be described via XML template files; commonly these are called `workunit.xml` and `result.xml`. The former describes the application and its input data while the latter describes the output data and the process which occurs once the application has executed. Template files must reside in the `projectName/templates/` directory.

1. A Typical Workunit (Input) Template

```
<input_ template>
```

```
  <file_ info>
```

```
    <number>0</number>
```

```

</file_info>

<file_info>
  <number>1</number>
</file_info>

<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>in_data1.txt</open_name>
  </file_ref>

  <file_ref>
    <file_number>1</file_number>
    <open_name>in_data2 . txt</open_name>
  </file_ref>
</workunit>

</ input_template>

```

This input template specifies that there are two input files called in_data1.txt and in_data2.txt. If an input file has been specified in the input template, this must be manually placed in the download directory before starting the project. The XML components above have the following meanings:

file_info: specifies the number of input files.

workunit: defines a unit of work. In this case, the executable only has two input files, hence the executable and the two input files can then be passed to the client and run.

file_ref: provides a reference to the file entered between the **open_name** tags.

Others XML descriptors can be entered dependent on the application resource requirements. These can be found at references [33, 34].

2. A Typical Result (Output) Template

```
<output_template>

  <file_info>
    <name><OUTFILE_0/></name>
    <generated_locally />
    <upload_when_present />
    <max_nbytes>5000</max_nbytes>
    <url><UPLOAD_URL/></url>
  </file_info>

  <result>
    <file_ref>
      <file_name><OUTFILE_0/></file_name >
      <open_name>out . txt</open_name>
    </file_ref>
  </result>

</output_template>
```

This output template describes the result of the application, specifies the unique name of the output file out.txt and that it is generated on the client and does not have to be downloaded. When present, it is automatically uploaded to the BOINC server and must not exceed the 5000 byte limit else it will fail to upload. The XML components above have the following meanings:

file_info: provides information of the resulting output file.

name/OUTFILE_0: gives the output file the unique name of workunitName_N.

generated_locally: indicates that the file will be generated on the client, rather than downloaded.

upload_when_present: indicates that the file should be uploaded when the application finishes.

url/UPLOAD_URL: this is replaced with the upload URL.

max_nbytes: if the output file is larger than this, an error will occur and it will not be uploaded.

file_ref: references the file and gives it a unique name related to the project and workunit identifier.

More information about the output file and the complete list of XML descriptors can be found at references [33, 34].

5.3.3 Deploying BOINC Clients

The last part in the installation process after setting up BOINC server and preparing the project to handle jobs, is installing BOINC client software in computer labs. Figure 5.5 shows a block diagram for the deployment process:

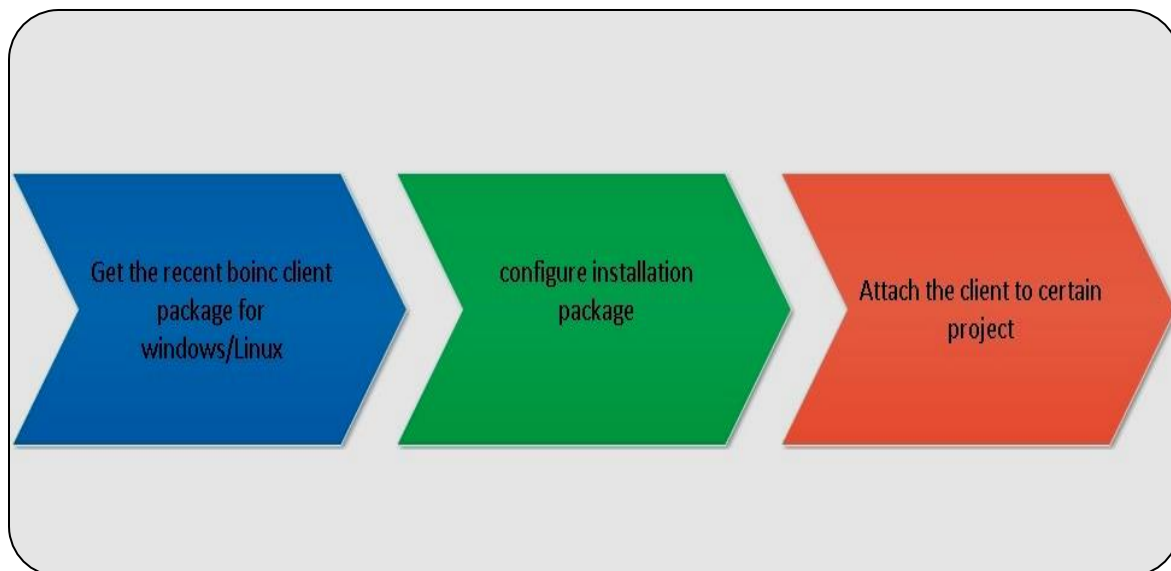


Fig. 5.5: Deploying BOINC Clients.

For the PPU campus environment, the BOINC client must be deployed to meet the following requirements:

- The BOINC Client must run all the time (even when no one is logged in).
- Project attachment must be automatic.
- BOINC manager must be disabled: no user intervention is required in client configuration (the student cannot attach to or disconnect from a project, cannot create a new user, or start/stop/pause/exit the BOINC client).
- Screen saver must be disabled: no screensaver is needed because it looks strange for students, they may think the PC is occupied/busy and choose not to use it.
- To insure security, no internet connection is required for the BOINC client since it runs within the university's LAN.
- For a large number of PCs, deployment must be performed automatically, not manually.

After deploying server and client software and attaching the clients to the created project, the project is now ready to send work units to clients and receive results back from them.

5.4 Object relational model and object design

The system has five basic software objects that are:

- **DBConnection:** manages the connection with the system DB's. all the other classes have the "access to" relation as shown in the class hierarchy in chapter 4.
- **ProjectCreation:** provides the functionality for project creation.
- **ProjectManagmant:** provides the functionality for project management.
- **UserCreation:** provides the functionality for adding new users to the grid system.
- **JobSubmission:** enable users submitting jobs.

The following tables show more details about the functionality of each class.

Table 5.1: DBConnection class

DBConnection
<pre>setDBServer(){ set the IP address of the DB server; } setDBUser(){ set the DB account username; } setDBUserPassword(){ set the password of the user account; } setDB(){ set the DB; } getDBConnection(){ return a connection to the DB specified; }</pre>

Table 5.2 ProjectCreation class.

ProjectCreation
<pre>getForm(){ return html code, represent a form to enable project creation } validateName(){ return boolean value based on the name entered for the project to be created.// project name may be a combination of letters,digits and //underscore or one of them. generate error messages for invalid inputs. } createProject(){ add the project to grid_system DB; execute the appropriate shell script that perform the project creation process; }</pre>

Table 5.3: Project Management class.

Project Management
<pre> getForm(){ return html code, represent a form to enable project management functions} attachClient(){ call the validateIP function to check the validity of the entered IP address; add the client who has the entered IP address to the computing resources that serve the project; generate error messages in the case of invalid IP address format; } deattachClient(){ call the validateIP function to check the validity of the entered IP address; delete the client who has the entered IP address from the computing resources that serve the project; generate error messages in the case of invalid IP address format; } stopProject(){ execute the shell script stopProject.sh;// stop all project diamonds modify the project status to not running status; } startProject(){ execute the shell script startProject.sh;// start all project diamonds modify the project status to running status; } restartProject(){// needed at the case when some diamonds are stopped working. execute the shell script restartProject.sh;// restart all project diamonds modify the project status to running status; } deleteProject(){ execute the shell script deleteProject.sh;// delete the project directory and //DB delete the project from grid_system DB; delete the project name from users following to this project; } validateIP(){ return Boolean value based on the entered IP address;//entered IP must match with the //correct IP address format generate error messages for invalid inputs. } </pre>

Table 5.4: User Creation class.

UserCreation
<pre>getForm(){ return html code, represent a form to enable adding new users; } validateName(){ return boolean value based on the name entered for the new user;// project name may be a combination of letters,digits and underscore or one of them. generate error messages for invalid inputs. } validateEmail(){ return Boolean value based on the email entered for the new user;//email //must match with the correct email format. generate error messages for invalid email format or in the case that the entered email is used by a user in the grid_system; } addUser(){ //in the case of valid inputs: add a new user to the grid_system DB having the information entered; create a directory for the new user;// to enable uploading the user jobs inside that //directory }</pre>

Table 5.5: JobSubmission class.

JobSubmission
<pre>getForm(){ return html code, represent a form to enable submitting jobs; } validateInputs(){ check that all inputs are set; check the type of selected file to upload;// it must be one of compression types return a Boolean value based on these parameters; } uploadFolder(){ upload the selected file to a specific folder; } extractFolder(){ extract the uploaded folder; } submitJob(){ submit the job to a specific project; add the job to DB; }</pre>

5.5 System functionalities

This section describes the system functionalities and how they are processed. System functionalities are classified as admin side functionalities, user side functionalities, and system functionalities. Admin side functionalities include project creation, project management, adding users to the system, and users' management. User side functionalities include submitting new jobs, monitoring jobs execution status, downloading results, deleting results, and aborting jobs execution. System functionalities handle managing the system at the lower level.

5.5.1 Admin side functionalities

Admin can create or delete projects, manage projects, add or delete users, and manage users.

Project creation:

Creating a BOINC project is done as shown in the flowchart below.

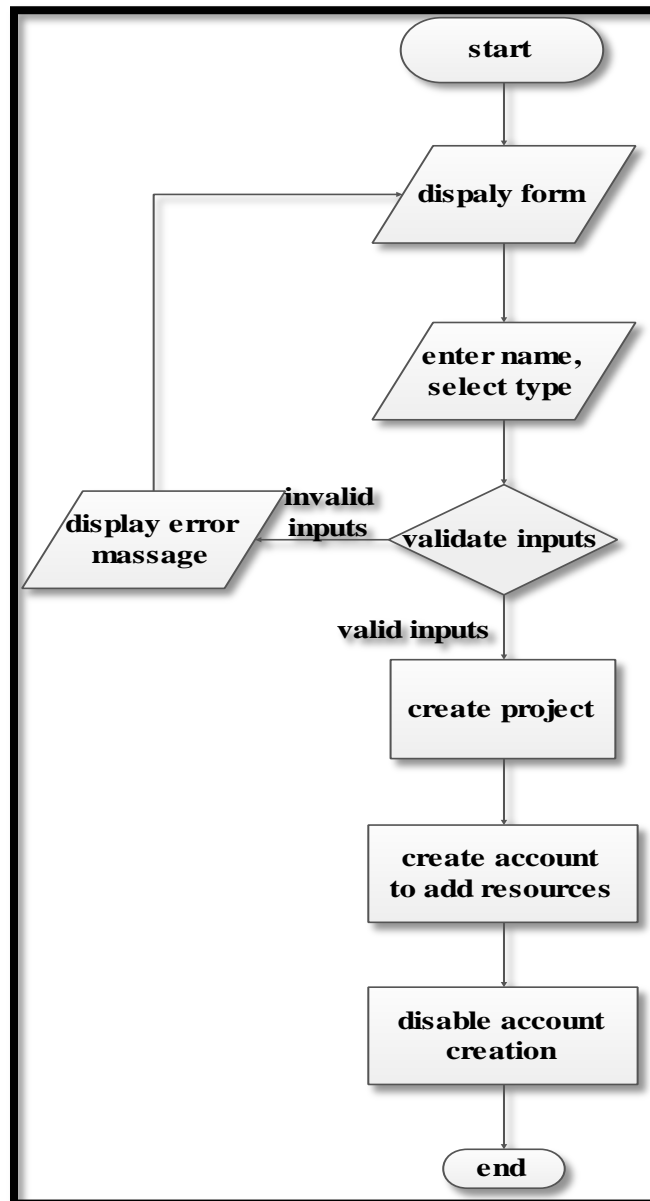


Figure 5.6: Project creation.

Project management:

Project management includes attach resources, delete resources, start, stop or restart project, disable or enable account creation, and project deletion. The following flowcharts describe these functionalities.

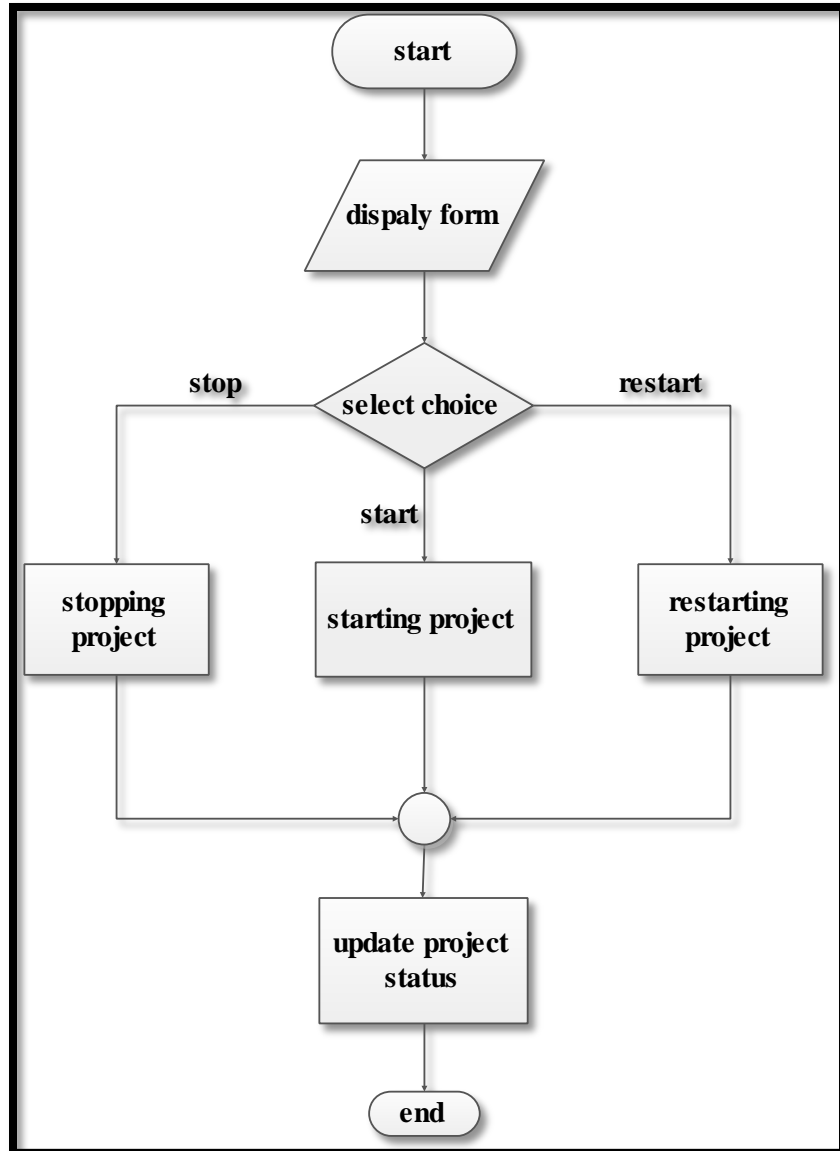


Figure 5.7: stopping/starting/restarting project.

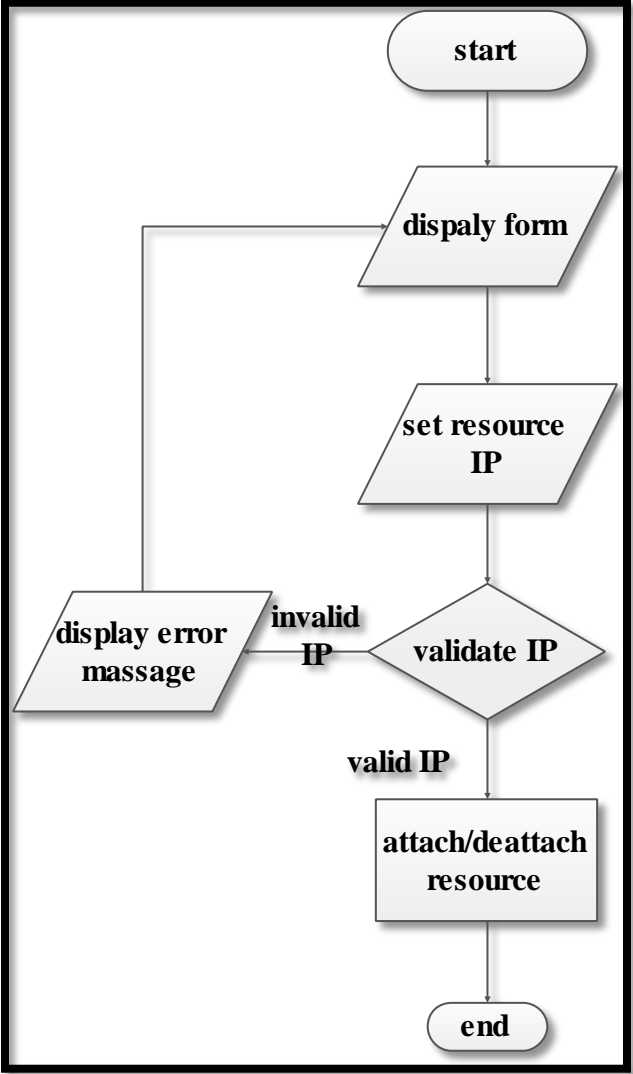


Figure 5.8: attaching/detaching client to a BOINC project.

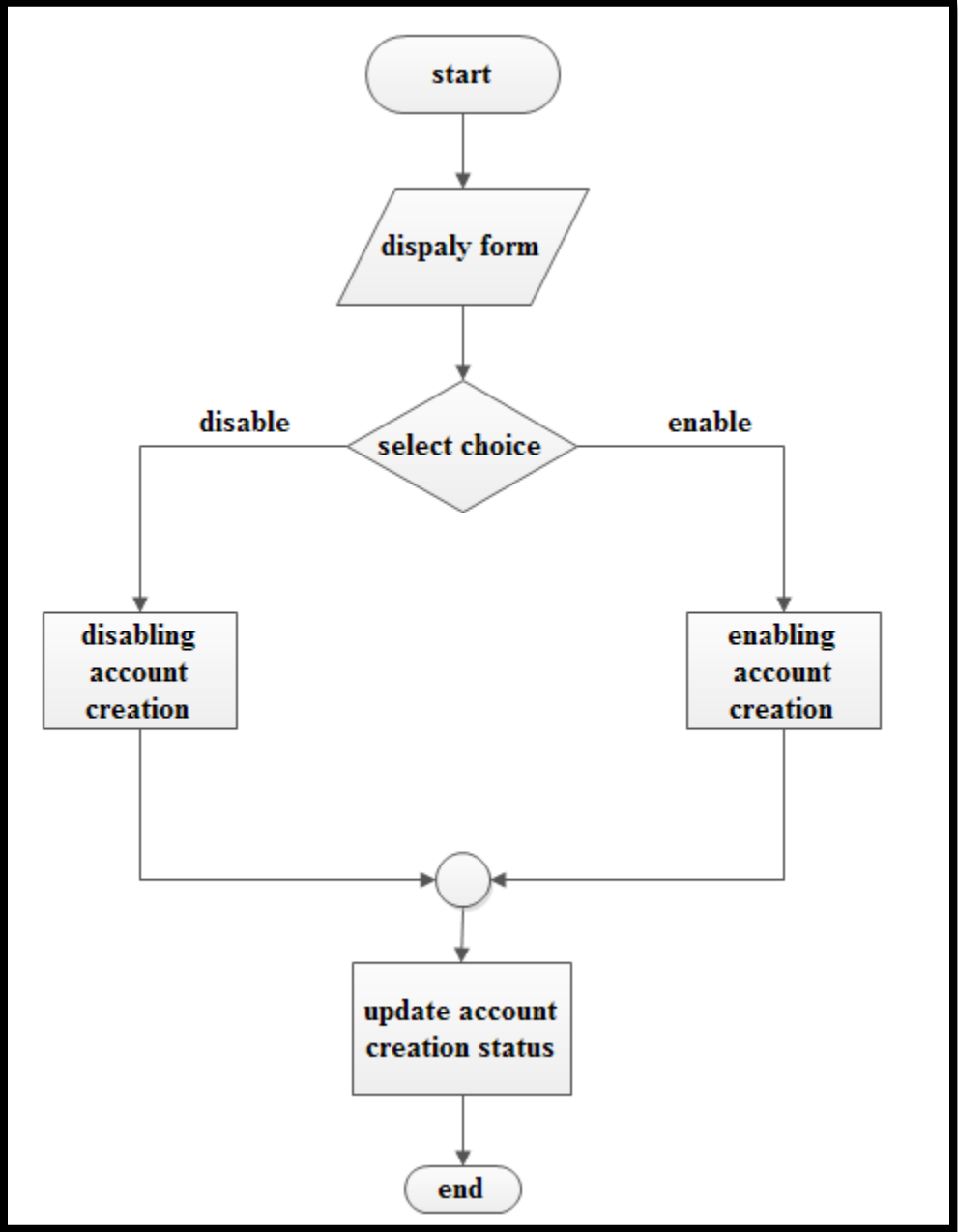


Figure 5.9: disable/enable account creation.

User creation and management:

Adding new user to the system is done as described in the figure below.

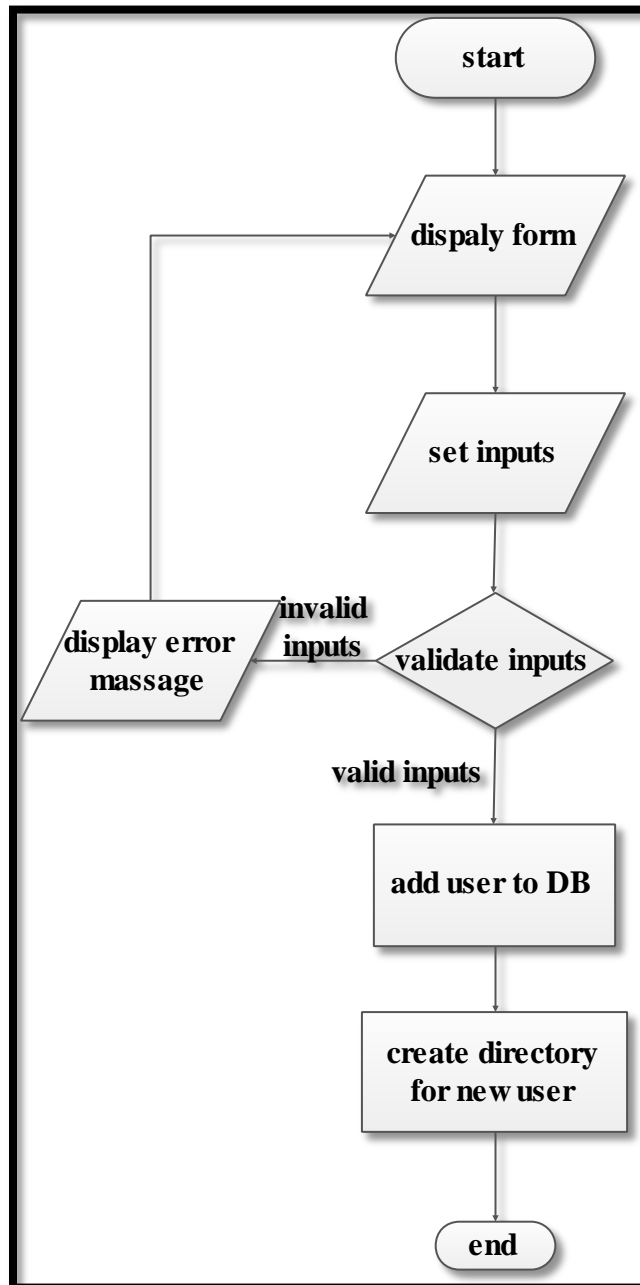


Figure 5.10: adding new user to a BOINC project.

Managing users is done by enabling the modification of some important information of the user, like moving the user from one project to another, or even by deleting the user.

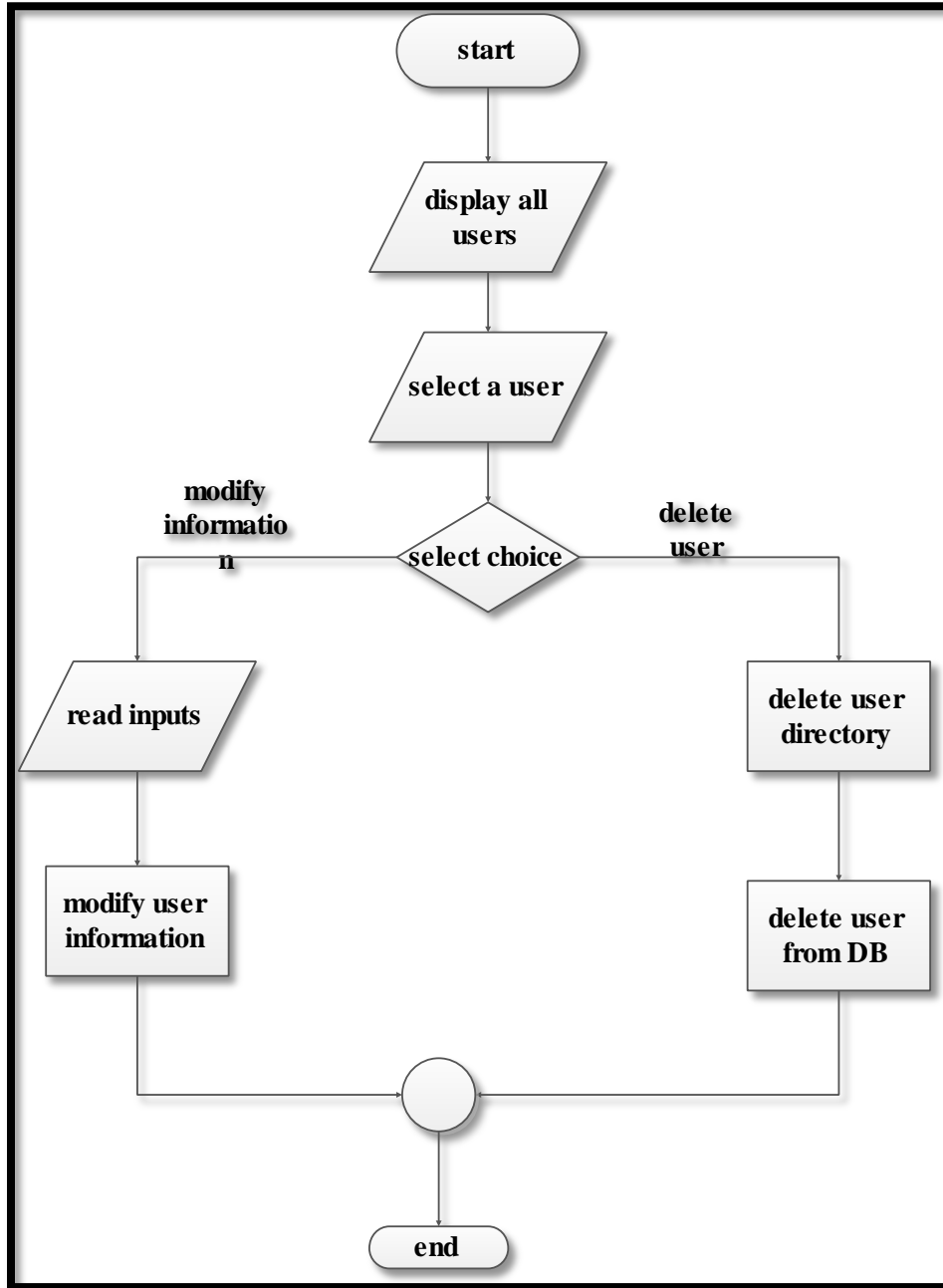


Figure 5.11: Users management.

5.5.2 User side functionalities

Users can submit jobs, monitor execution status, abort jobs, download results, delete results and modify their account information.

Job submission:

Job submission is done as in figure 5.11.

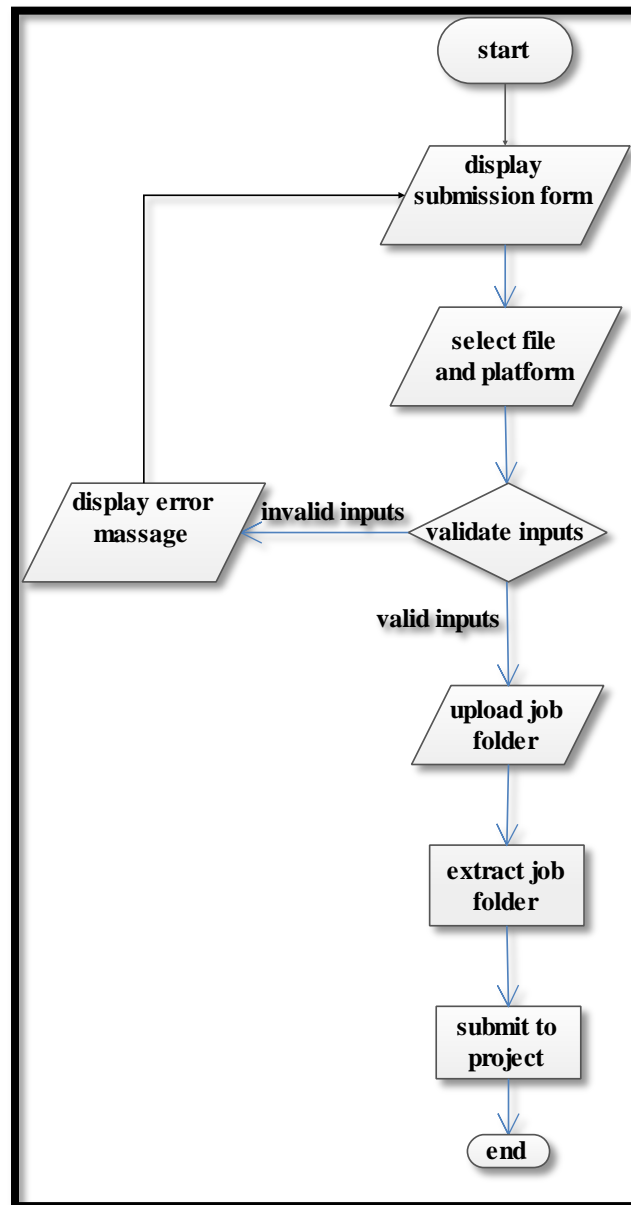


Figure 5.12: job submission.

Jobs management:

Jobs management includes monitoring execution status, aborting jobs while execution, downloading results of completed jobs, and deleting the results.

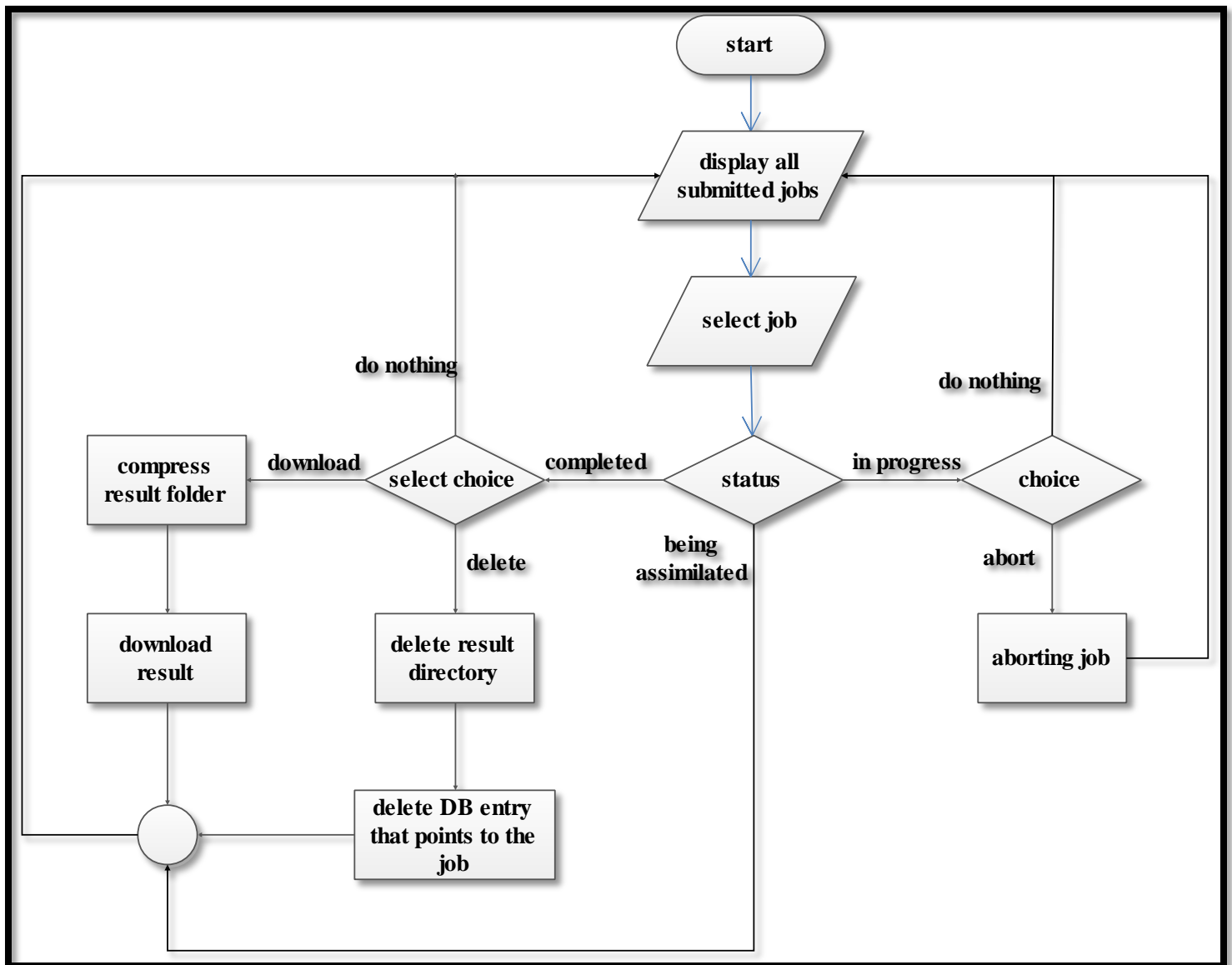


Figure 5.13: jobs management.

5.5.3 System functionalities

System needs to perform some functions automatically as a response for some functions from the higher level. Functions that are purely initiated by the system exist at the job execution stage. This stage described below.

Job execution:

As stated in chapter four; job execution is initiated by the system, while the overall process of the job execution is done by the cooperation between the server and the client. Figure below clarifies the process of job execution.

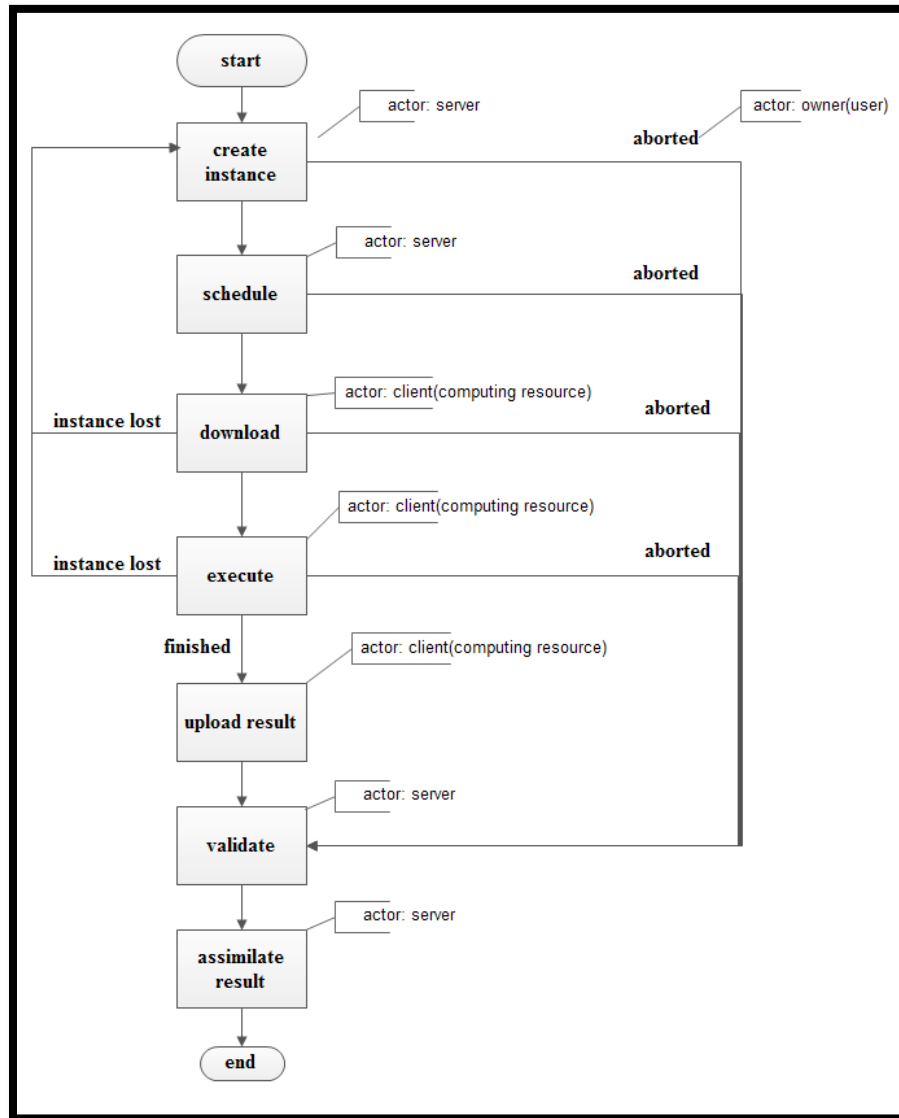


Figure 5.14: Job Execution.

5.6 Software interface design

This section describes user interfaces that will be implemented to enable the interaction between the system and grid users.

The About page interface:



Figure 5.15: The About Page.

Contact page interface:



Figure 5.16: Contact page.

Login page interface:

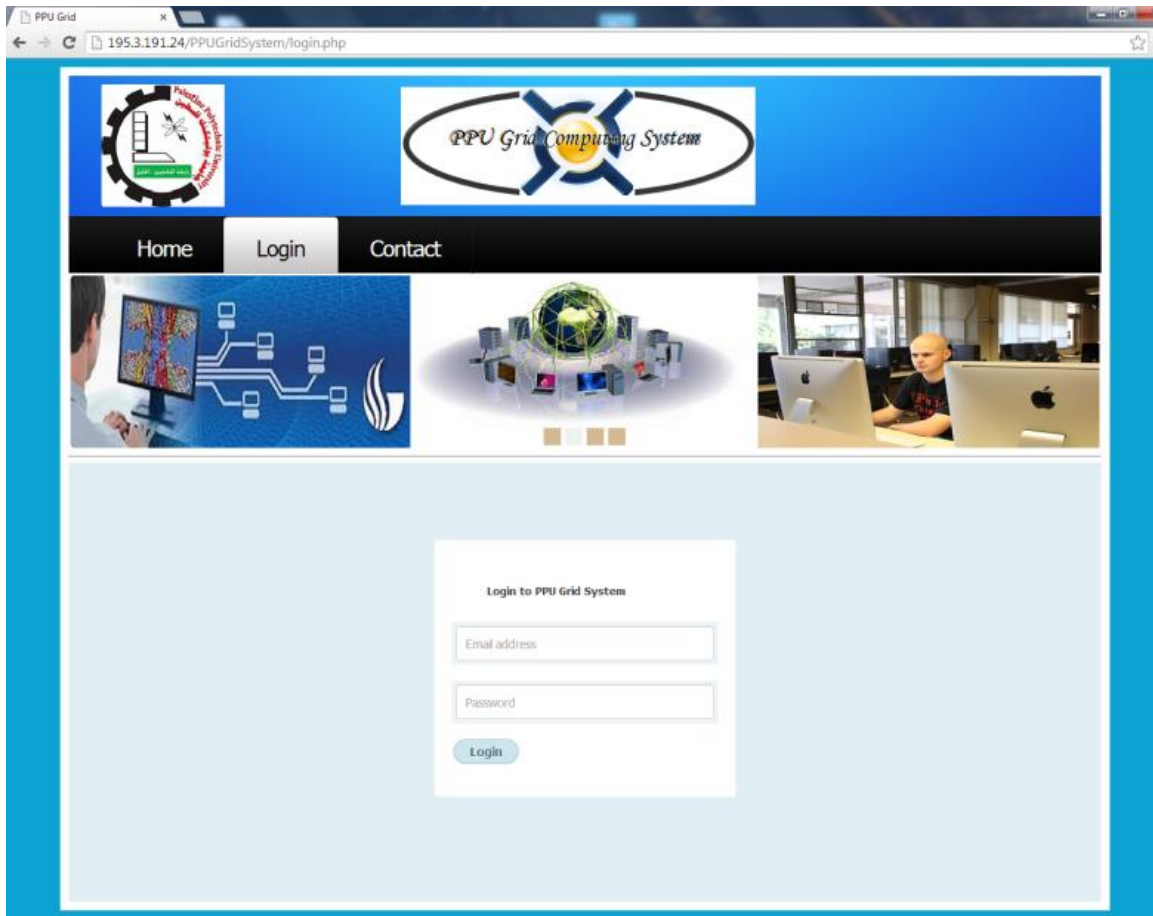


Figure 5.17: Login Page Interface.

Admin home page interface:

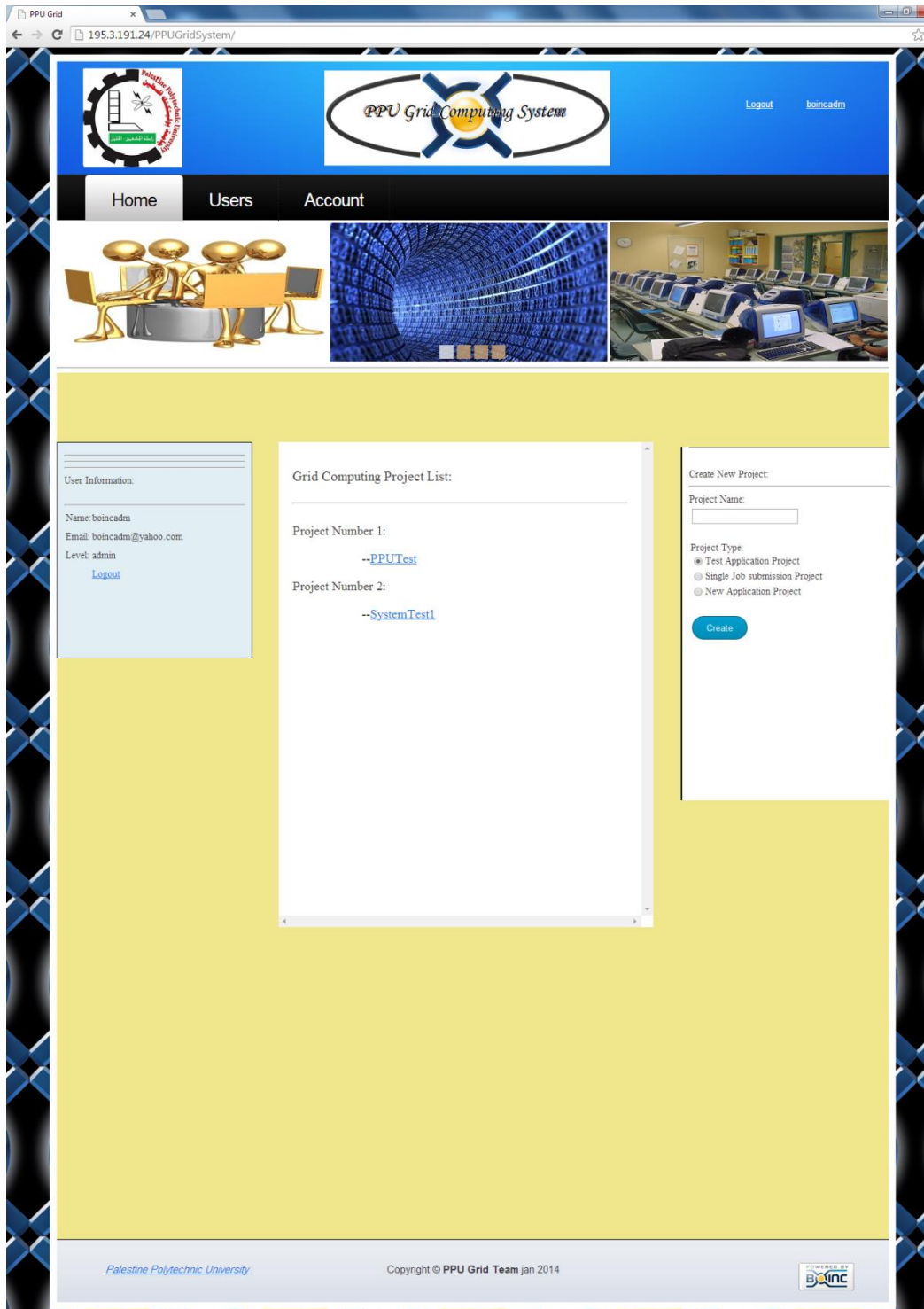


Figure 5.18: Admin home page interface.

Users page interface:

PPU Grid

195.3.191.24/PPUGridSystem/Users.php

Logout boincadm

Home Users Account

User Information:

Name: boincadm
Email: boincadm@yahoo.com
Level: admin
[Logout](#)

All Authenticated System Users:

Name	Email	Project	select
boincadm	boincadm@yahoo.com		<input type="checkbox"/>
Muhammad	mohammad.thwaib@yahoo.com	System Test1	<input type="checkbox"/>
Ibrahim	qusaymusa@yahoo.com	System Test1	<input type="checkbox"/>
Mohammed	mohammed@ppu.edu	System Test1	<input type="checkbox"/>
superuser	superuser@ppu.edu	System Test1	<input type="checkbox"/>
saleem	saleem@ppu.edu		<input type="checkbox"/>

[Delete users](#)

Add a New user:

Name:
Email:
Password:
ProjectID: SystemTest1
Level: user
[Add user](#)

Palestine Polytechnic University

Copyright © PPU Grid Team jan 2014

POWERED BY BOINC

Figure 5.19: Users page interface.

Project management page interface:

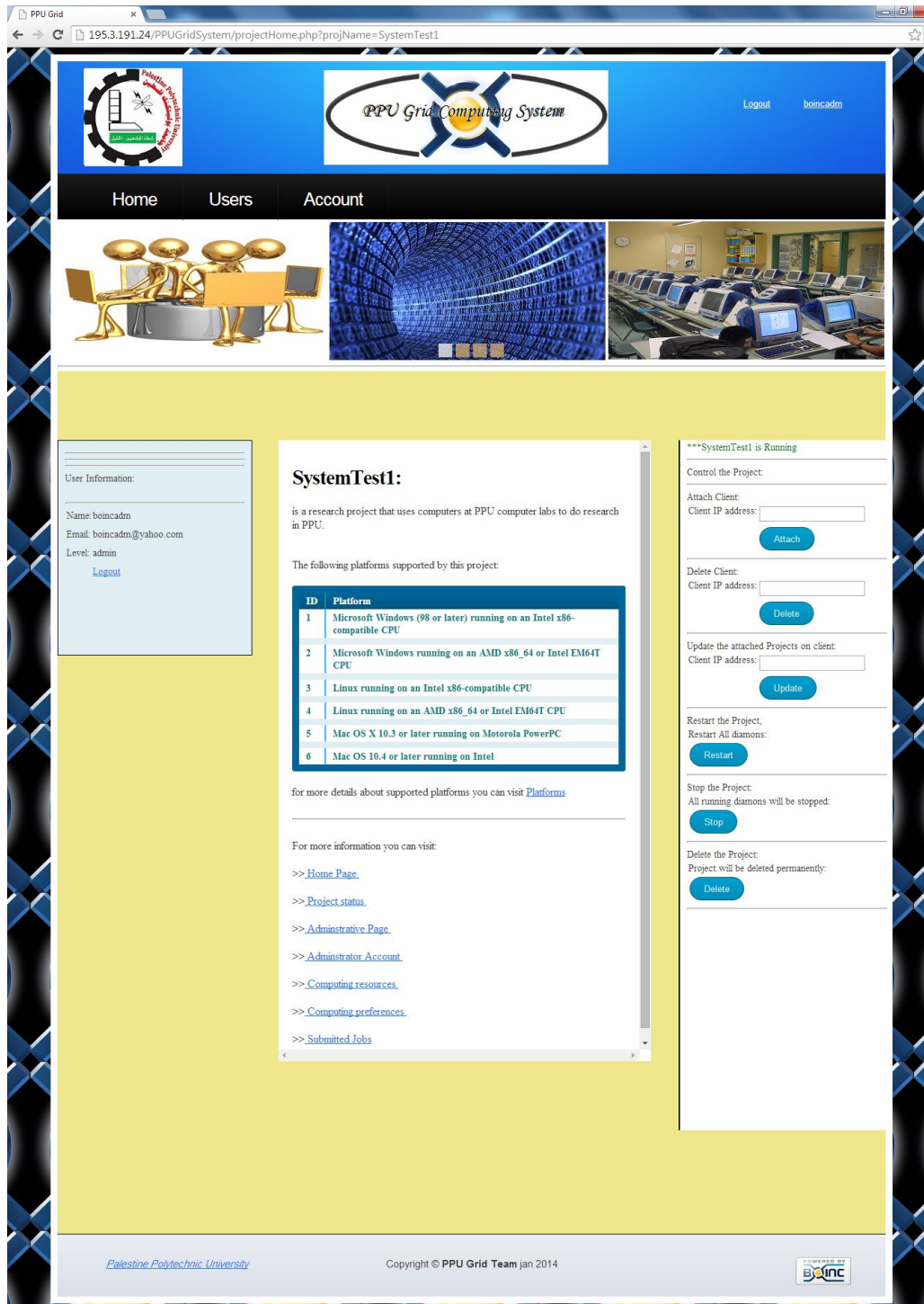


Figure 5.20: Project management page interface.

Modify user information page interface:

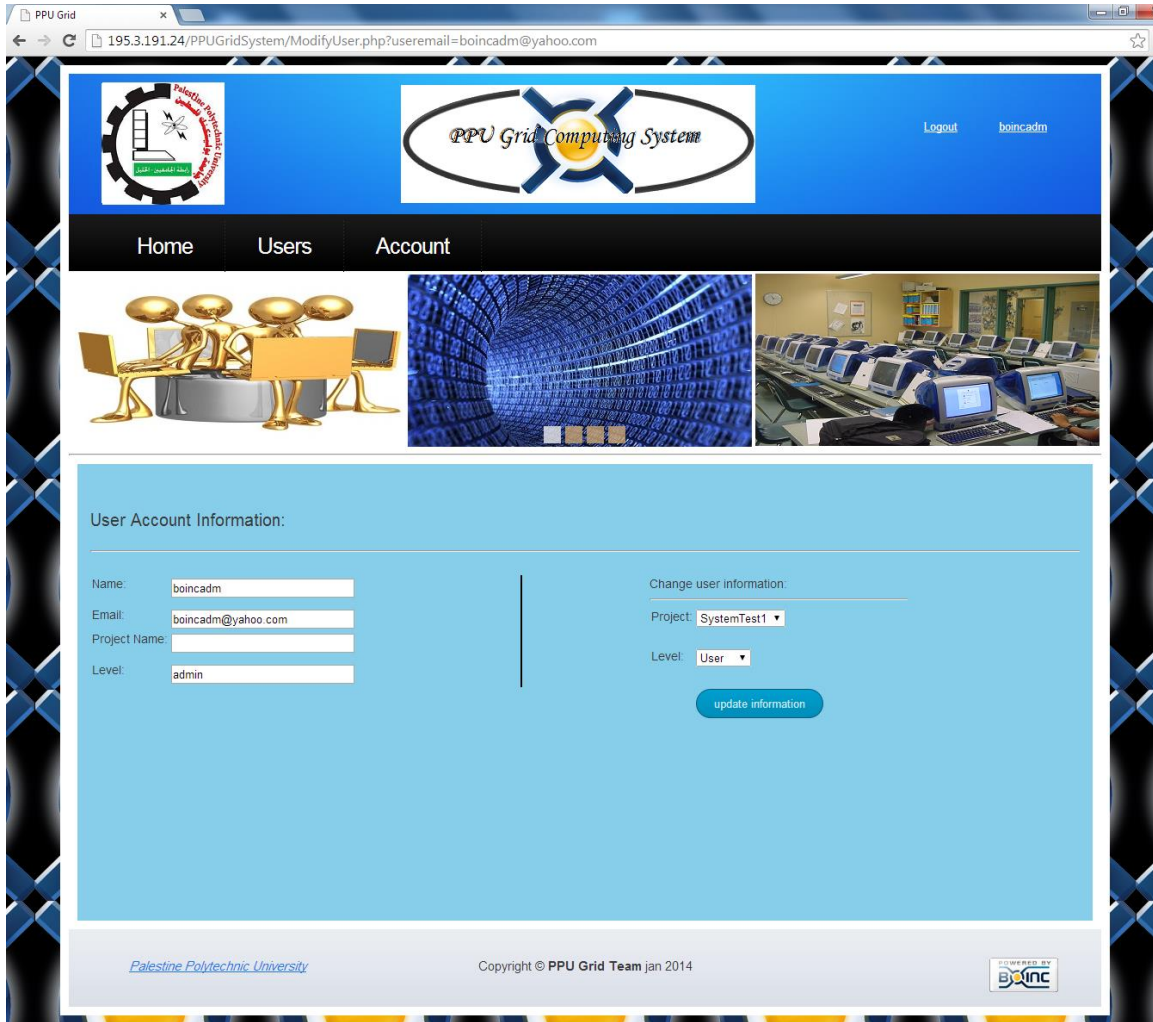


Figure 5.21: Modify user page interface.

Job submission page interface:

The screenshot shows a web browser window titled "PPU Grid" with the URL "195.3.191.24/PPUGridSystem/JobSubmission.php". The page features a header with three images: a person at a computer, a server rack, and a person at a computer. The main content area is yellow and contains several sections:

- User Information:** Name: Muhammad, Email: mohammad.thwaib@yahoo.com, Level: user, and a [Logout](#) link.
- Project Information:** Name: SystemTest1, Status: Running, and a [Go to Project home page](#) link.
- JOB Submission:** A "File to upload:" field with a "Choose File" button and "No file chosen" text.
- Platform Selection:** A table with the following rows:

Platform	Select
Jobs for Microsoft Windows (98 or later) running on an Intel x86-compatible CPU	<input type="radio"/>
Jobs for Microsoft Windows running on an AMD x86_64 or Intel EM64T CPU	<input type="radio"/>
Jobs for Linux running on an Intel x86-compatible CPU	<input type="radio"/>
Jobs for Linux running on an AMD x86_64 or Intel EM64T CPU	<input type="radio"/>
Jobs for Mac OS 10.4 or later running on Intel	<input type="radio"/>
Jobs for Intel 64-bit Mac OS 10.5 or later	<input type="radio"/>
- Parameters (optional):** An empty text input field and a blue "Submit" button.
- Instructions:** A text box containing instructions for job execution, including requirements for the compressed folder and specific file names for standard input and output.

The footer includes the text "Palestine Polytechnic University", "Copyright © PPU Grid Team jan 2014", and a logo for "BOINC".

Figure 5.22: Job submission page interface.

Modify account information page:

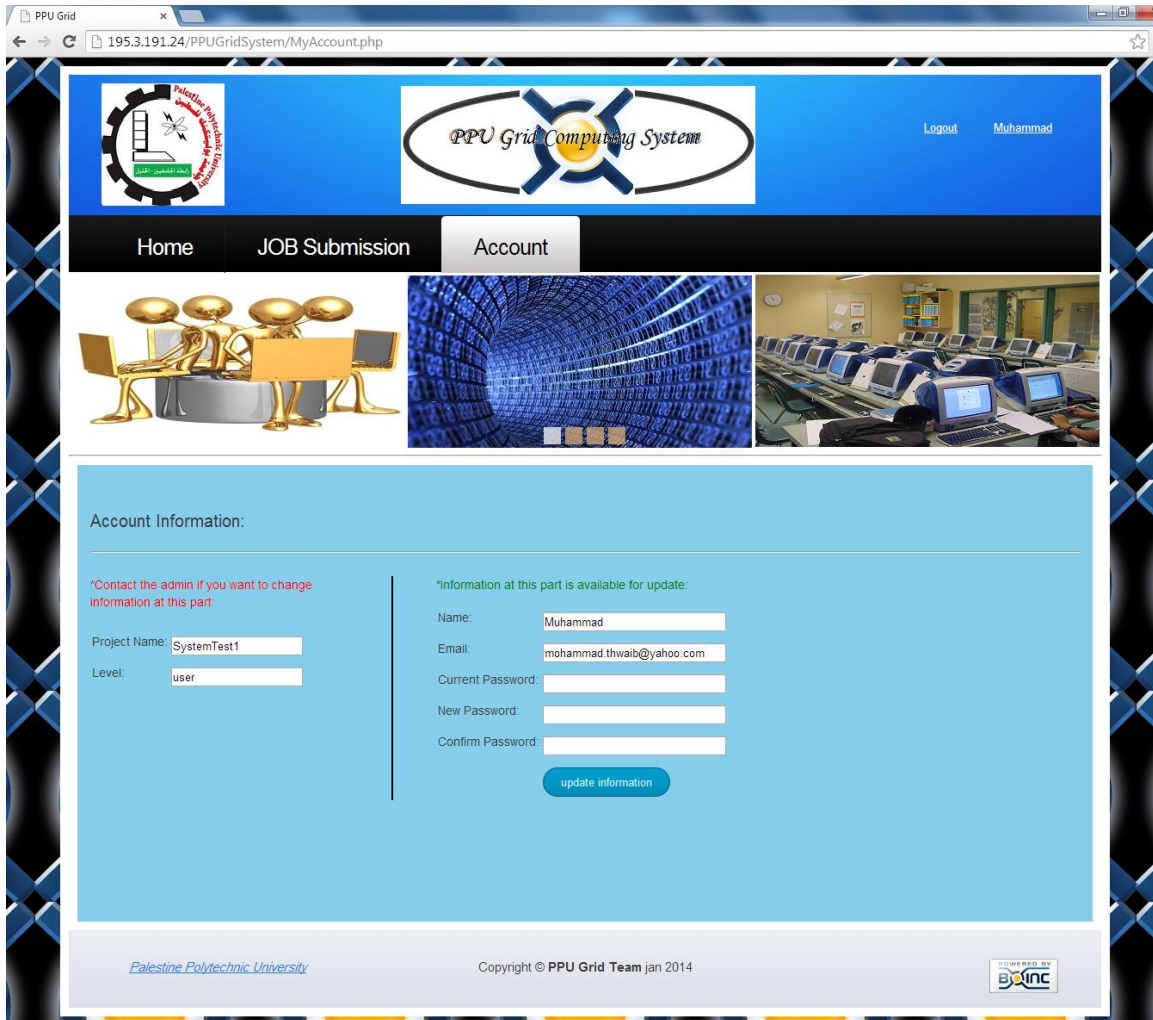


Figure 5.23: modify account page interface.

5.7 Data Base Design

The controlling DB named **grid_system** and has the ERD as in figure 5.22. This DB manages the overall system since it is project independent; it connects all projects together, stores all users of different projects, and manages the users' jobs.

There is another DB which is project dependent; each BOINC project has its own DB. So, in the case of many projects exists on the same server, there will be many DBs in the system. More information about the BOINC project DB can be found in appendix B.

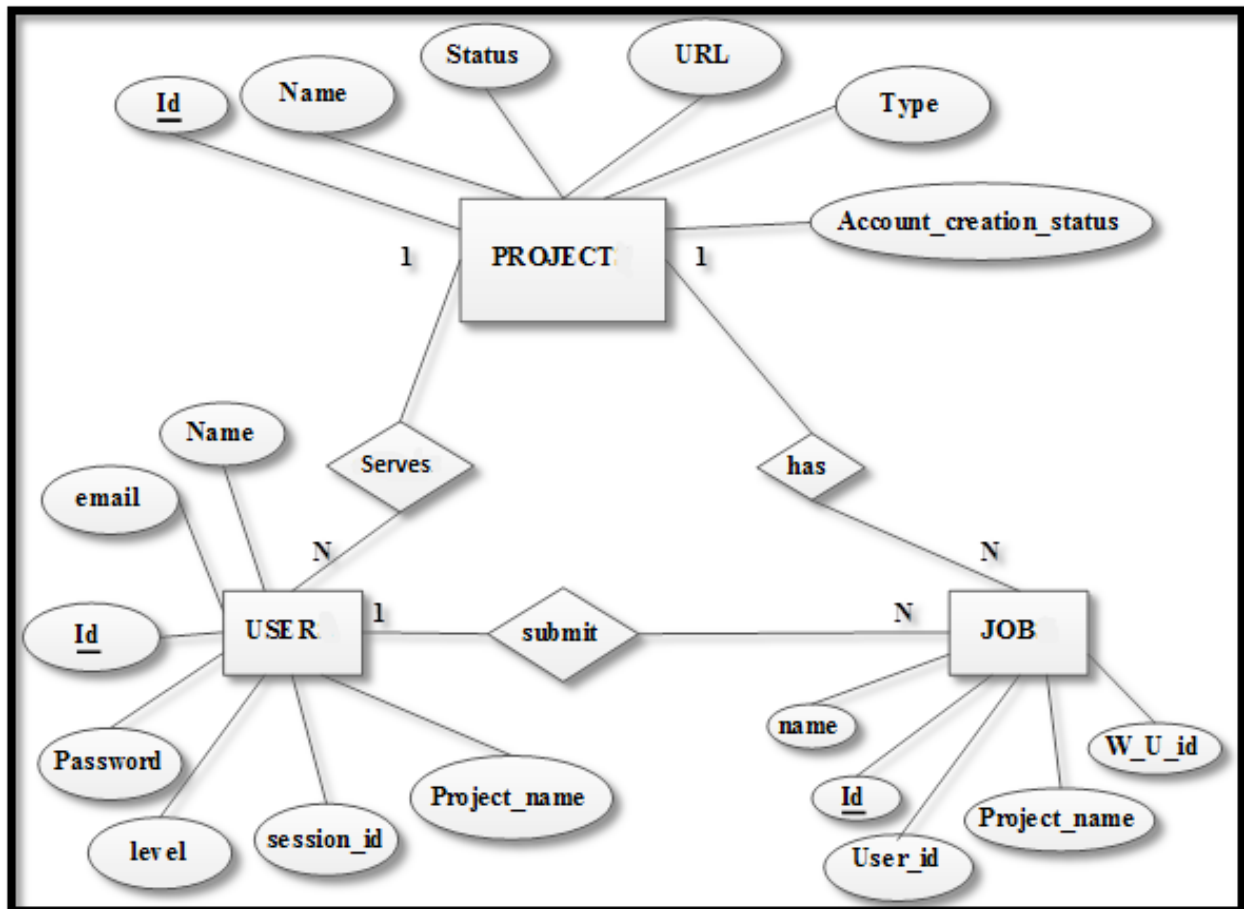


Figure 5.24: Grid System DB ERD.

5.8 Hardware interface Design

Basic hardware components in the system are:

- 1- Server: System coordinator.
- 2- Routers, switches.
- 3- Grid clients: the computing resources that perform the computations assigned to the grid system.

PPU LAN represents the system infrastructure.

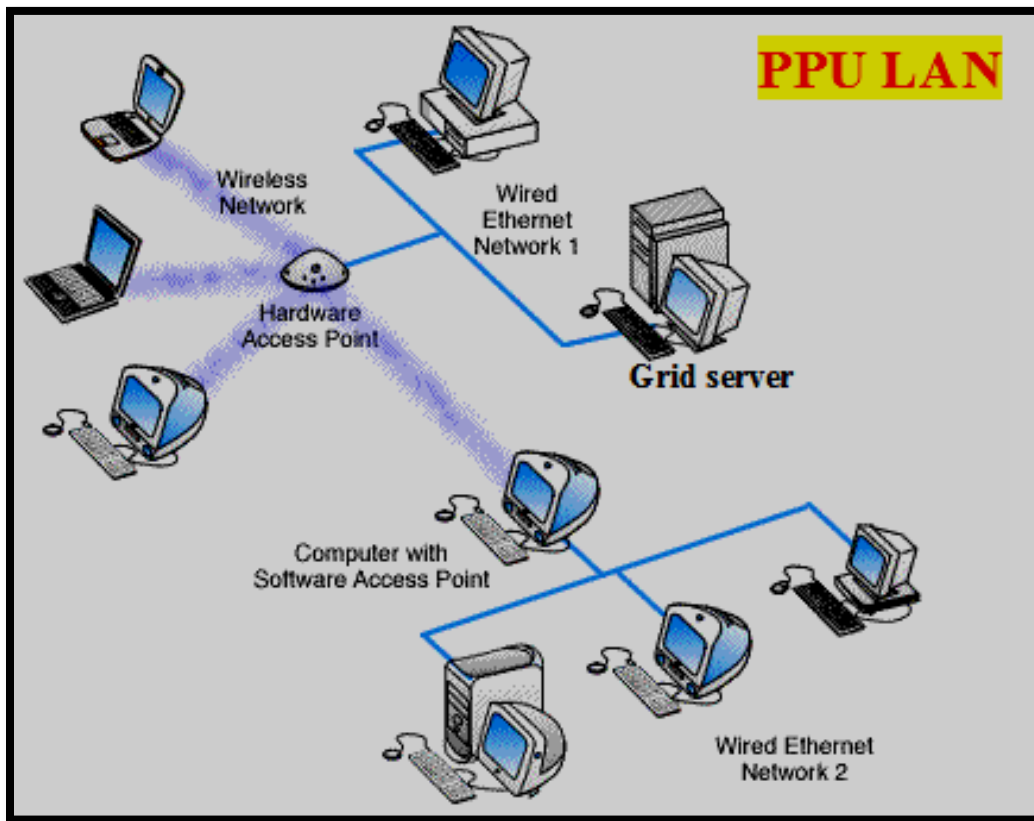


Figure 5.25: Hardware interface design.

5.9 Overall Work Summary

The work during the last semester was mainly concentrated on trying to better understand the project, determine its components, costs and time schedule. In addition, a lot of time was paid for analyzing and designing the project. This was done through five chapters, in each chapter we discussed a particular issue associated with this project. In the following paragraphs we summarize the work which was done in each chapter.

In chapter one, we talked about the project idea. Then we presented some of the project motivations. In addition, we have discussed the project scope which addressed these topics: system input and output, project requirements, components, deliverables, assumptions, boundaries constraints, and the Initial project organization.

In chapter two, we presented an overview of grid computing technology and its related issues. In addition, we discussed some of previous grid-based projects and studies and analyzed their results.

In chapter three, we defined all the project task sets and sketched the project Gantt chart. Then we stated the available options to be used in the project implementation. Also some of project risks were stated and analyzed. Finally we define all the components that are needed for the project evaluation in addition to the costs estimations.

In chapter four, we defined the system actors, use-cases templates, use-cases diagrams, CRC modeling and the class hierarchy and relationship. All scenarios that may occur were introduced in the use case templates. Each actor can initiate some of use-cases which clarified in use-case diagrams.

In chapter five, we stated the basic features of the system design. These features include object relational model, state behavioral model and System configuration design. It includes also, software interface, hardware interface and database design.

5.10 What Is Next?

The work of this project during the next four months of project life will be divided into three parts. First, we will measure the CPU utilization in selected computer lab PCs in the university. In order to get a realistic results, the study should be performed during the university work days (from Sunday to Thursday), and during work hours only (from 08:00 to 16:00).

In the second part, we start the system configuration process to build a local desktop grid using BOINC middleware. This grid consists of the server device, which can be a real server or one of our available high performance computers, and three or more grid clients(PCs), each one can be any computer available in our labs.

Finally, in the third part we will start testing the grid and trying to build the grid interface. This interface enables grid users like researchers to make use of the grid which was built in the second part by providing them the capability to submit jobs to grid's clients and get the results back.

5.11 Summary

This chapter stated the basic features of the system design. These features include each of the following:

- System configuration design: It has a large part of the final project design. That includes server and clients' configuration, software installation, project creation and deployment and other configurations.
- Object relational model: as shown previously there are five basic software objects that need to be implemented for this system. They are: ProjectCreation, ProjectManagement, UserCreation, and JobSubmission.
- System functionalities: describes all system functions using flowcharts.
- Software interface design: shows the main portal pages interfaces that will be implemented.
- Data base design: this section defines in context the basic structure of the controlling data base. This data base will contain basically three tables – project, user, and job.
- Hardware interface Design: it shows the system hardware environment.

In the last two sections we talked about what was accomplished until now during the first four months of life cycle of this project. Then we presented the major parts and the set of tasks which will be done in the next semester during the remaining time for this project.

Chapter Six

Implementation and Testing

6.1 Overview

In this chapter, we explain the implementation procedure at two levels. At the first level (the low level), we describe the system configuration steps to prepare the grid environment in addition to grid system core functions implemented using Linux bash scripts. At the second level (the higher level), we describe the portal implementation and how the core functions were enabled remotely. Also this chapter includes the testing applied on all system functionalities at both higher and lower levels.

6.2 System Configuration and Core Functions

In this section we describe the implantation of system configuration part of this project. This part represents the low level layer of the project. We show the process of setting up the grid environment.

Shell scripts using bash interpreter were used at this level to communicate directly with the operating system, the BOINC core client software and the BOINC middle-ware as a whole. The PPU Grid System Portal is built above this layer.

6.2.1 BOINC Server Deployment

The first step of building our grid system is to configure and prepare BOINC server to work. Our BOINC server computer has the following properties:

- Processor: core i5 3.2GHz.
- RAM: 4GB
- Disk storage capacity: 320GB.
- Network: 100Mbps.
- Global IP address: 195.3.191.24
- Local IP address: 10.10.16.12

BOINC server runs on UNIX operating systems. In our project we used Ubuntu12.04 LTS 64-bit Linux distribution. In addition, we installed the BOINC server software prerequisites and dependencies. Furthermore, we solved some problems that appeared during the deploying of BOINC server. The detailed explanation the BOINC server setting up process is described in appendix A.

6.2.2 BOINC Client Deployment

At the client side, BOINC client software was installed on each computer participated in our grid system. We installed BOINC client as a service by checking the **Service Install** checkbox. In addition, we disabled screen saver option and prevented other users (usually students) from controlling BOINC client software.

To make the communication between the BOINC core client and a remote computer secure, two files are added to **BOINC data directory** (where BOINC's data files will be stored). These files are:

1. **gui_rpc_auth.cfg file**

This file contains the BOINC client password. Any remote computer wants to communicate with BOINC core client must provide this password in its communication commands.

2. **remote_hosts.cfg file**

This file contains the IPs or DNS names of remote hosts that are allowed to communicate with BOINC core client if they provide the correct password stored in **gui_rpc_auth.cfg file**. Any other host will be prevented from communicating with BOINC core client.

The technical details of preparing grid clients (PCs) to participate in the grid system are described in appendix A.

6.2.3 Project Creation

A **project** is an entity that does distributed computing using BOINC. Projects are independent; each one has its own applications, database, web site, and servers, and is not

affected by the status of other projects. Each project is identified by a master URL, the URL of its web site. Multiple projects can coexist on a single server computer [37].

At the implementation level, a project consists of [37]

- a directory tree, containing files related to the project, and
- a MySQL database.

We created BOINC projects for different purposes, two types of these projects are:

1. **New/Empty Project**

The process of creating a BOINC project requires you to follow a certain steps and instructions given by BOINC documentation. At the end of these steps you may need to solve some problems regarding server Linux distribution. The process of creating **New/Empty** project on Ubuntu 12.04 LTS is described in appendix B.

The Empty project is a BOINC project that doesn't contain any application. Creating an Empty project requires well Linux administrative skills and a long list of steps. In addition, it needs to solve some problems regarding server Linux distribution (Ubuntu 12.04). Furthermore, one will find himself forced to repeat all these steps when he decides to create another project.

To facilitate project creation process, we need to automate this process. For this purpose, we create a bash script called **createProject.sh**. This script executes all required steps to create a project and solve all problems.

- **createProject.sh bash script**

- **Usage: bash createProject.sh projectName [installroot]**

Where:

projectName: the name of the project

installroot: the path of the folder containing the project. It is optional argument, if not determined then the default (/home/boincadm /projects) is used.

Usage Example: bash createProject.sh emptyProject

2. Test/Example Project

Test/Example Project is an Empty Project that has an example application for test purposes. The example application is a single-thread native BOINC application [38]. This application has application versions to run on different well known platforms. The application reads an input file, converts the file to upper case and writes it to an output file.

To create a BOINC project running the test application example you can follow the same scenario with creating empty BOINC project with some changes. This process is described in appendix B. Figure 6.1 describes the main steps of creating Test/Example application.

- **createTestProject.sh bash script**

- **Usage: bash createTestProject.sh projectName [installroot]**

Where:

projectName: the name of the project

installroot: the path of the folder containing the project. It is optional argument, if not determined then the default (/home/boincadm /projects) is used.

Usage Example: bash createTestProject.sh testProject

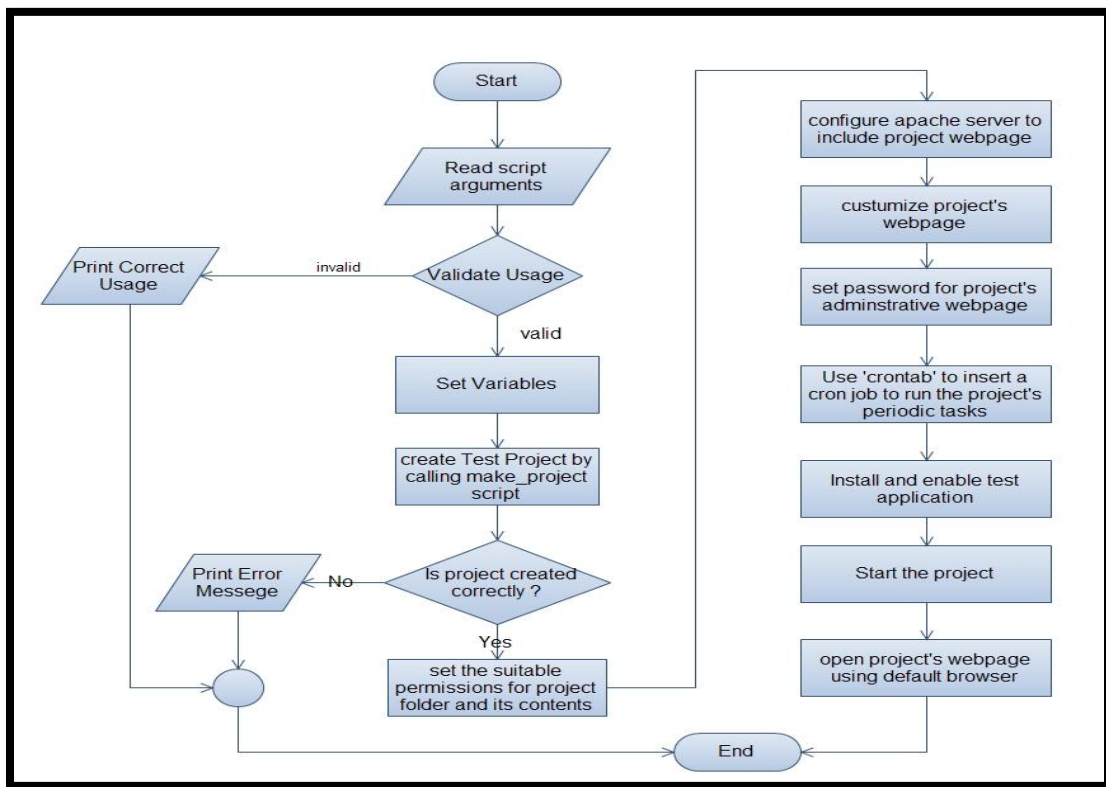


Figure 6.1: Create Test Project Algorithm

6.2.4 Create Admin Account

For each project you can create one or more accounts, any computing resource must follow a certain account. In our case, we only need one account since that all computing resources will be under the control of campus grid system Administrator (Admin account) for each project.

Admin account creation can be done using BOINC manager interface, but in our system we assumed the BOINC manager is not used. To solve this problem we created a bash script called (**createAccount.sh**) which uses **boinccmd tool** commands to create the admin account for a certain project.

The **createAccount.sh** bash script implements the steps of creation admin account that are depicted in Figure 6.2 show below:

- **createAccount.sh** bash script
 - **Usage: createAccount.sh projectName**
Where: **projectName:** is the name of the project

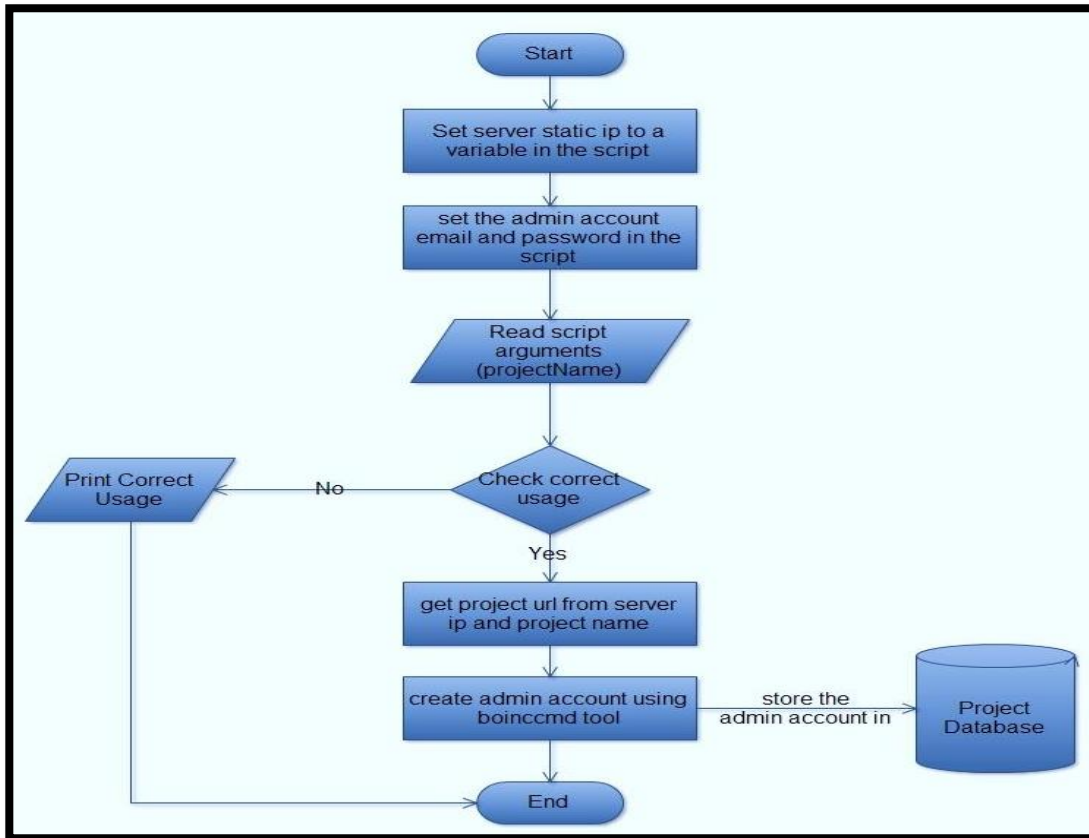


Figure 6.2: Create Account Algorithm

6.2.5 Disable/Enable Account Creation

Since we need only one account (admin account) that is responsible for all computing resources in our grid system, we should disable account creation after creating this account.

Disabling account creation for a certain project can be done by adding the following line `<disable_account_creation/>` between `<config>...</config>` tags inside `config.xml` file which exists in the project directory. To enable account creation again, we need to remove the previously added line from `config.xml` file.

To automate the process of enabling/disabling account creation for a particular project, we created two bash scripts which automatically add or remove the needed line. These scripts are: **disable_account_creation.sh** and **enable_account_creation.sh**.

- **disable_account_creation.sh bash script**

- **Usage: bash disable_account_creation.sh projectName [installroot]**

Where:

projectName: the name of the project.

installroot: the path of the folder containing the project. It is optional argument, if not determined then the default (/home/ boincadm /projects) is used.

Usage Example: bash disable_account_creation.sh test

- **enable_account_creation.sh bash script**

- **Usage: bash enable_account_creation.sh projectName [installroot]**

Where:

projectName: the name of the project

installroot: the path of the folder containing the project. It is optional argument, if not determined then the default (/home/ boincadm /projects) is used.

Usage Example: bash enable_account_creation.sh test

6.2.6 Client to Project Attachment/Detachment

After creating a project and admin account, clients (computing resources) can be attached to a certain project under the control of admin account of that project. Once the client (computer) becomes attached to a project, it becomes ready to download jobs from that project, execute them and return the results back to the project server.

The client attachment to a certain project can be done using BOINC manager interface. However, we want to attach the computing resources silently without any user interaction. For this purpose, we created a bash script called **attach.sh** that does the attachment process automatically.

The **attach.sh** bash script can attach multiple clients to the same project at the same time given their IP addresses and using boincmd tool to communicate with BOINC core client.

In contrast, if we used the BOINC manager interface approach, we would have to repeat the attachment process on each client (computer) separately which is effort and time consuming especially if the number of the clients to be attached is large. The explanation below shows the implementation and the usage of **attach.sh** bash script.

- **Attachment script**

- **Usage:** `bash attach.sh projectName client_ip`

- **OR :** `bash attach.sh projectName 'client_ip [other client_ips]'`

- Where: **ProjectName:** the name of the project

- **client_ip:** the ip of the client machine

- **Note:** If you have more than one ip, put them between single quotations.

- **Example 1:** `attach.sh ProjectX 192.168.1.5`

- As result of **example 1** script calling, the client (computer) with IP addresses 192.168.1.5 will be attached to ProjectX.

- **Example 2:** `attach.sh ProjectX '192.168.1.5 192.168.1.6'`

- As result of example 2 script calling, the two clients (computers) with IP addresses 192.168.1.5 and 192.168.1.6 will be attached to ProjectX.

On the other hand, sometimes we want to detach a certain client from a particular project. In other words, we want to stop a certain computing resource from serving a particular project. This process is called client detachment and it is done using another bash script called **deattach.sh**.

- **Detachment script**

- **Usage:** `bash deattach.sh projectName client_ip`

- **OR :** `bash deattach.sh projectName 'client_ip [other client_ips]'`

- Where: **ProjectName:** the name of the project

- **client_ip:** the ip of the client machine

- **Note:** If you have more than one ip, put them between single quotations.

- **Example 1:** `deattach.sh ProjectX 192.168.1.5`

As result of **example 1** script calling, the client (computer) with IP addresses 192.168.1.5 will be detached from ProjectX.

Example 2: deattach.sh ProjectX '192.168.1.5 192.168.1.6'

As result of example 2 script calling, the two clients (computers) with IP addresses 192.168.1.5 and 192.168.1.6 will be detached from ProjectX.

The attachment/detachment scripts implements the algorithm shown in figure 6.3.

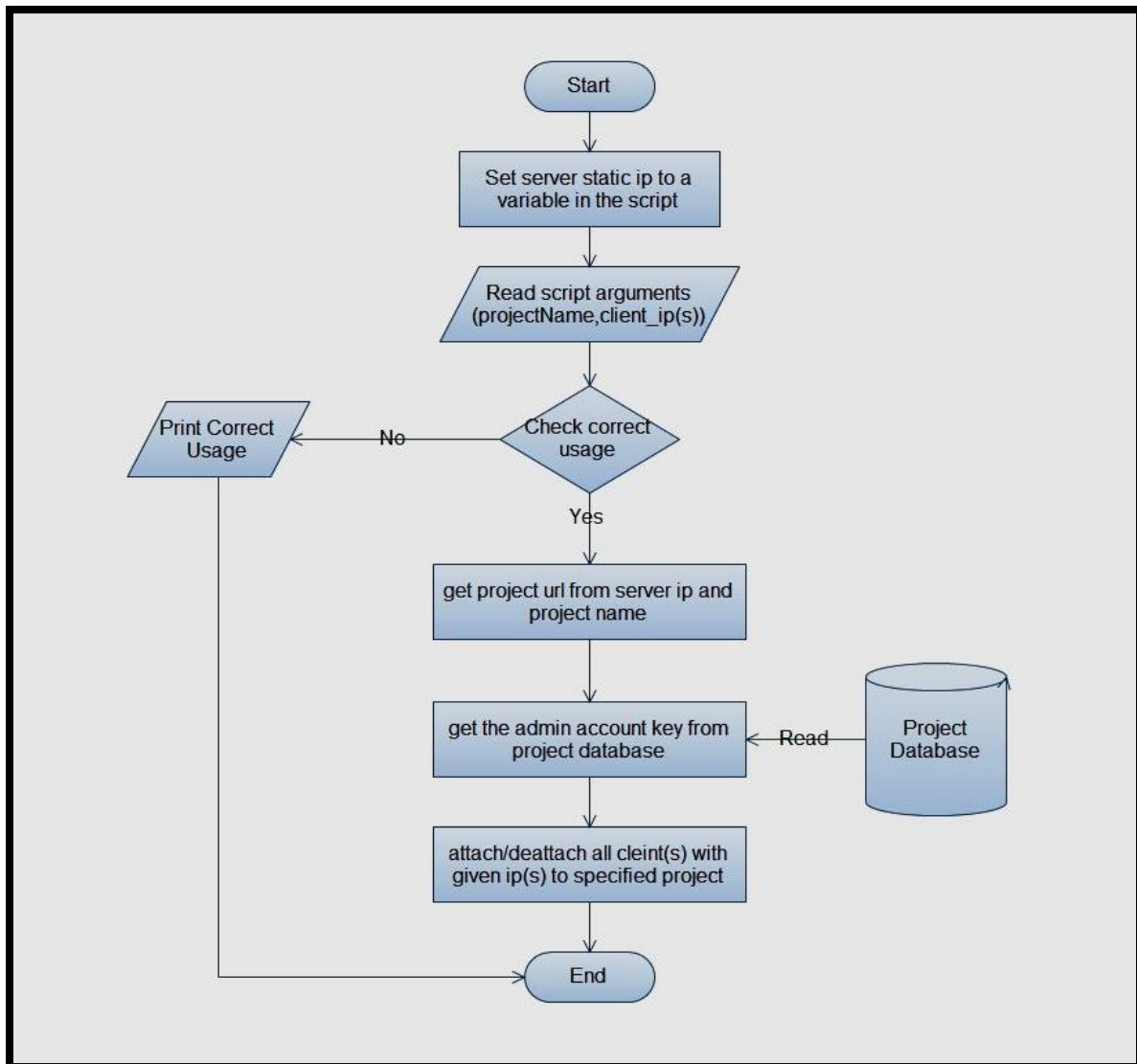


Figure 6.3: Attachment/Detachment Algorithm

6.2.7 Project Control

During the project life time we need to control BOINC project in different ways. BOINC has the following Python scripts control a project [67]:

- **bin/start**
Start the project: start all daemons, and remove the stop_sched and stop_daemon files (see below).
- **bin/stop**
Stop the project (create the stop_sched and stop_daemon files)
- **bin/start --cron**
If the project is started, perform all periodic tasks that are past due, and start any daemons that aren't running. Otherwise do nothing.
- **bin/status**
Show whether the project is stopped. Show the status of all daemons. Show the status of all periodic tasks (e.g., when they were last executed).

We created four bash scripts to control a certain project using the previous Python scripts. These bash scripts are:

- **Start project script**
 - **Usage:** `bash startProject.sh projectName [installroot]`
Where:
projectName: the name of the project
installroot: the path of the folder containing the project. It is optional argument, if not determined then the default (/home/ boincadm /projects) is used.
Usage Example 1: `bash startProject.sh test`
Usage Example 2: `bash startProject.sh test /home/boincadm /myprojects`
- **Stop project script**
 - **Usage:** `bash stop.sh projectName [installroot]`
 - **Implementation:** the same as the startProject.sh with the last line replaced by `./bin/stop`
- **Restart project script**
 - **Usage:** `bash stopProject.sh projectName [installroot]`
 - **Implementation:** the same as the startProject.sh with the last line replaced by:
`./bin/stop`

./bin/start

- **Project status project script**
 - **Usage:** `bash projectStatus.sh projectName [installroot]`
 - **Implementation:** the same as the `startProject.sh` with the last line replaced by `./bin/status`

6.2.8 Update Attached Projects

At the client side the core client which attached to one or more projects the **admin** needs sometimes to update the attached projects. Project update causes the BOINC core client to contact scheduling server immediately.

The bash script **updateAttachedProject.sh** is created in order to update all a client's attached projects. It can be called on a local or a remote client using client IP address. Multiple clients can be updated with same call.

- **Update script**
 - **Usage:** `bash updateAttachedProject.sh projectName client_ip`
OR: `bash updateAttachedProject.sh projectName 'client_ip [other client_ips]'`

Where: **ProjectName:** the name of the project

client_ip: the ip of the client machine

Note: If you have more than one ip, put them between single quotations.

Example 1: `updateAttachedProject.sh ProjectX 192.168.1.5`

As result of **example 1** script calling, all projects to which the client (computer) with IP addresses 192.168.1.5 will be updated.

Example 2: `updateAttachedProject.sh ProjectX '192.168.1.5 192.168.1.6'`

As result of example 2 script calling, all projects to which the two clients (computers) with IP addresses: 192.168.1.5 and 192.168.1.6 will be updated.

The `updateAttachedProject.sh` script implements the algorithm shown in figure 6.4.

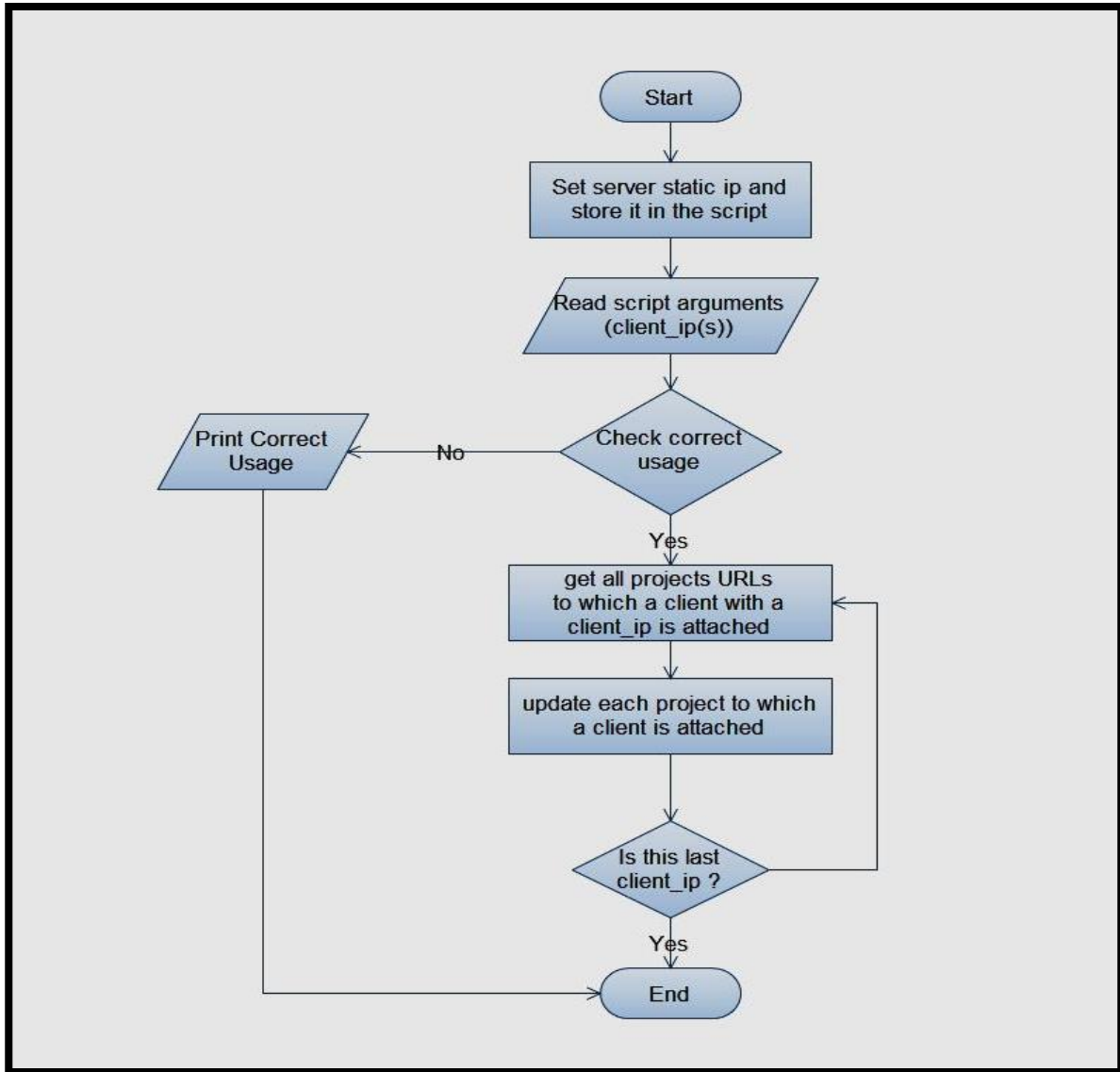


Figure 6.4: Update Attached Projects Algorithm

6.2.9 Project Deletion

To delete a particular project, we have to do two things:

- Delete the project directory tree, containing files related to the project, and
- Delete the project MySQL database.

We implemented the project deletion using **deleteProject.sh** bash script. Figure 6.5 describes the project deletion steps.

- **deleteProject.sh** bash script

- **Usage:** `bash deleteProject.sh projectName [installroot]`

Where:

projectName: the name of the project

installroot: the path of the folder containing the project. It is optional argument, if not determined then the default (/home/boincadm /projects) is used.

Usage Example: `bash delete.sh test`

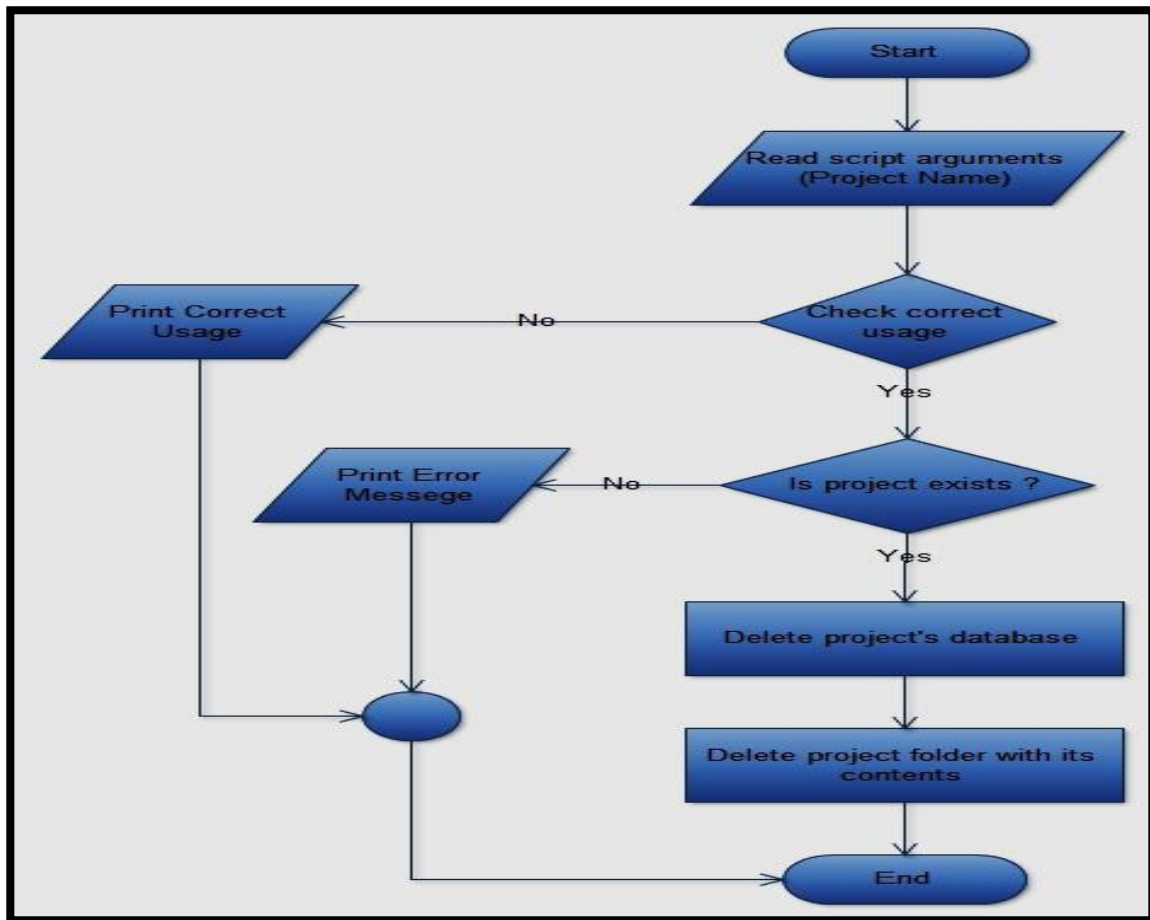


Figure 6.5 Project Deletion Algorithm.

6.2.10 Single Job Project

This is the third type of projects that is supported by our grid system. It is designed to support the single job submission mechanism.

6.2.10.1 Single job submission mechanism

BOINC is designed to handle streams of millions of jobs. It takes some work to set up a stream: you need to create applications and application versions, workunit (WU) and result templates, validators, assimilators, etc [39].

BOINC's single job submission mechanism lets you run a job without any of these hassles. In order to do this, one should configure BOINC project to handle single jobs. The configuration steps are described in [39]. Jobs submitted by this way will have some estimates and limit parameters. These parameters are [39]:

Job estimates and limits

- processing estimate: 1 GFLOPS-hour
- processing bound: 1 GFLOPS-day
- memory bound: 500MB
- disk bound: 1GB
- delay bound: 1 week

You can change these parameters by editing the `boinc_submit` script (see below).

6.2.10.2 Single Job Project Creation

We used single job mechanism supported by BOINC to build a single job project. To explain the process of configuring single job project, we assume the following:

- The directory of the BOINC Server source code is: `/home/boincadm/boinc/`
- The root directory of the BOINC project that is to be configured to work with single job submission mechanism: `/home/boincadm/projects/SingleJobProject`
- The boincadm user home directory is: `$HOME = /home/boincadm/ = ~`

-Steps for the server platform

1. Create **Empty/New project** (see appendix B)

bash createProject.sh projectname

2. Initially the wrapper for the server platform in `/home/boincadm/boinc/samples/wrapper` is not compiled. To compile it, we need to do the following:

a) **cd /home/boincadm/boinc/samples/wrapper**

b) **make**

Note: you need to do this only one time to compile the wrapper. Once it is compiled, it can be used in the next step without any need to recompilation.

3. Configure BOINC Project for single jobs:

a. change directory (cd) to project folder(directory)

b. **html/ops/single_job_setup.php path-to-boinc-samples**

c. follow the resulting instructions of the previous command

e.g. **cd /projects/SingleJob**

html/ops/single_job_setup.php /home/boincadm/boinc/samples

4. Set the environment variable `BOINC_PROJECT_DIR` to the root directory of the project.

export BOINC_PROJECT_DIR=\$HOME/projects/SingleJobProject

5. go to the path that contains the application(executable) and its input and output files:

cd /path/to/application

6. After configuring single job project, we use the **boinc_submit** PHP script to submit a job (see the subsection 6.2.10.4).

```
~/boinc/tools/boinc_submit [boinc-options] program [program-options]
```

The boinc-options are:

--infile name

specifies an input file.

--stdin name

direct the given file to the program's stdin.

--outfile name

specifies an output file.

--stdout name

direct the program's stdout to the given file.

--platform

the platform on which the program is to be run (default: the server's platform; assumed to be Linux).

- Using other platforms

In order to use other platforms, we need to do further steps, let us take windows_intelx86 platform as an example. In this case you need to do the following:

1. Go to your project's apps directory
2. Create directories apps/single_job_windows_intel86/1.0/windows_intelx86.

Note: 1.0 is the application version

3. Download the BOINC wrapper executable for windows_intelx86, and put it in the windows_intelx86 directory.
4. Add single job application for **windows_intelx86** platform. This is done by editing the file **\$HOME/projects/SingleJobProject/project.xml**, adding the following lines before **</boinc>** tag:

```
<app>
```

```
<name>single_job_windows_intelx86</name>
```

**<user_friendly_name> Jobs for Microsoft Windows (98 or later)
running on an Intel x86-compatible CPU </user_friendly_name>
</app>**

5. Add **single_job_assimilator** and **sample_trivial_validator** daemons. This is done by editing the file **\$HOME/projects/SingleJobProject/config.xml**, adding the following lines before **</daemons>** tag:

```
<daemon>  
  <cmd>single_job_assimilator -app single_job_windows_intelx86</cmd>  
  <output>single_job_assimilator_windows_intelx86.out</output>  
  <pid>single_job_assimilator_windows_intelx86.pid</pid>  
</daemon>
```

```
<daemon>  
  <cmd>single_job_assimilator -app single_job_windows_intelx86</cmd>  
  <output>single_job_assimilator_{$friendly_name}.out</output>  
  <pid>single_job_assimilator_{$friendly_name}.pid</pid>  
</daemon>
```

6. Go to your project's root directory
7. Run `bin/xadd`
8. Run `bin/update_versions`. Answer yes to all questions.
9. Restart the project.
 - `bin/stop`
 - `bin/start`

You can then submit jobs to Windows/x86 hosts:

1. Create a directory with a Windows executable for your application, say `app.exe`
2. `cd /path/to/application`
3. Type a command of the form: `boinc_submit --platform windows_intelx86 app.exe`

Example: testBoinc.exe is C++ program compiled on windows_intelx86 computer, it reads from standard input (**cin**) and writes to standard output(**stdout**). A job can be submitted using the following command:

1. Go to application directory that contains the files: testBoinc.exe , **stdin** and **stdout**(optional: created if not exist).
cd /path/to/application
2. Run the command:
~/boinc/tools/boinc_submit --platform windows_intelx86 --stdin in --stdout out testBoinc.exe

BOINC developers have built wrappers for different platforms. Table 6.1 shows the BOINC supported platforms wrappers. To develop applications for hosts of these platforms, you should follow the same previous steps for windows_intelx86 platform with the following changes:

- Create directories
apps/**single_job_PlatformTechnicalName/1.0/PlatformTechnicalName**.
Note: 1.0 is the application version
- Add single job application for particular platform. This is done by editing the file **\$HOME/projects/SingleJobProject/project.xml**, adding the following lines before **</boinc>** tag:
<app>
 <name>single_job_PlatformTechnicalName</name>
 <user_friendly_name> PlatformUserFriendlyName
</user_friendly_name>
</app>
- Download the BOINC wrapper executable for a particular platform from BOINC website, and put it in the **PlatformTechnicalName** directory.

- Add **single_job_assimilator** and **sample_trivial_validator** daemons for application of particular platform. This is done by editing the file **\$HOME/projects/SingleJobProject/config.xml**, adding the following lines before **</daemons>** tag:

```
<daemon>

    <cmd>single_job_assimilator -app
single_job_PlatformTechnicalName</cmd>
    <output>single_job_assimilator_PlatformTechnicalName.out</output>
    <pid>single_job_assimilator_PlatformTechnicalName.pid</pid>
</daemon>
```

```
<daemon>
    <cmd>single_job_assimilator -app
single_job_PlatformTechnicalName</cmd>
    <output>single_job_assimilator_PlatformTechnicalName out</output>
    <pid>single_job_assimilator_PlatformTechnicalName.pid</pid>
</daemon>
```

- **For each particular platform:**
 - **PlatformTechnicalName:** is platform technical name that is adopted by BOINC and used in the code. These names are shown in table 6.1 under **Technical Names** title.
 - **PlatformUserFriendlyName:** is platform user-friendly name that appear to end users. These names are shown in table 6.1 under **User-friendly Names** title.

Table 6.1: Supported Platforms

User-friendly names	Technical names
Jobs for Microsoft Windows (98 or later) running on an Intel x86-compatible CPU	windows_intelx86
Jobs for Microsoft Windows running on an AMD x86_64 or Intel EM64T CPU	windows_x86_64
Jobs for Linux running on an Intel x86-compatible CPU	i686-pc-linux-gnu
Jobs for Linux running on an AMD x86_64 or Intel EM64T CPU	x86_64-pc-linux-gnu
Jobs for Mac OS 10.4 or later running on Intel	i686-apple-darwin
Jobs for Intel 64-bit Mac OS 10.5 or later	x86_64-apple-darwin

If you want to support all BOINC supported platforms, you have to use previous steps to add a single job application and wrapper for each platform. By doing this, you will have a single job project which can be used to submit jobs compiled on different platforms. According to that, you will be able to exploit more hosts(computers) that have different architectures (platforms).

6.2.10.3 Automatic SingleJob Project Configuration

As it is obvious from previous explanation, the process of creating and configuring single job project contains a lot of hassle and requires you to follow a long list of steps. Also, if you want to let the single job project supporting different platforms, then the difficulty will increase. In addition, if we want to create more than one single job project, we will have to repeat all previous steps again which will become a headache for developers.

The best solution of such problem is to automate this process. In Linux systems this can be done by writing a shell script that does the whole job. For this purpose we wrote a shell script using bash interpreter to configure an **Empty/New project** as a **single job project**. This script is called **singleJob.sh**.

However, **singleJob.sh** script assumes that two folders are stored in **\$HOME** directory. The first folder is called **wrappers** which contains all pre-compiled wrappers for different platforms that are downloaded from BOINC website and stored with certain names in this folder. The second folder is called **ppu_boinc** which contains two PHP files relating job submission and monitoring.

singleJob.sh script is called after calling **createEmpty.sh** script that creates an **Empty/New project** (see appendix B). The work flow of **singleJob.sh** is described in the algorithm shown in the figure 6.6.

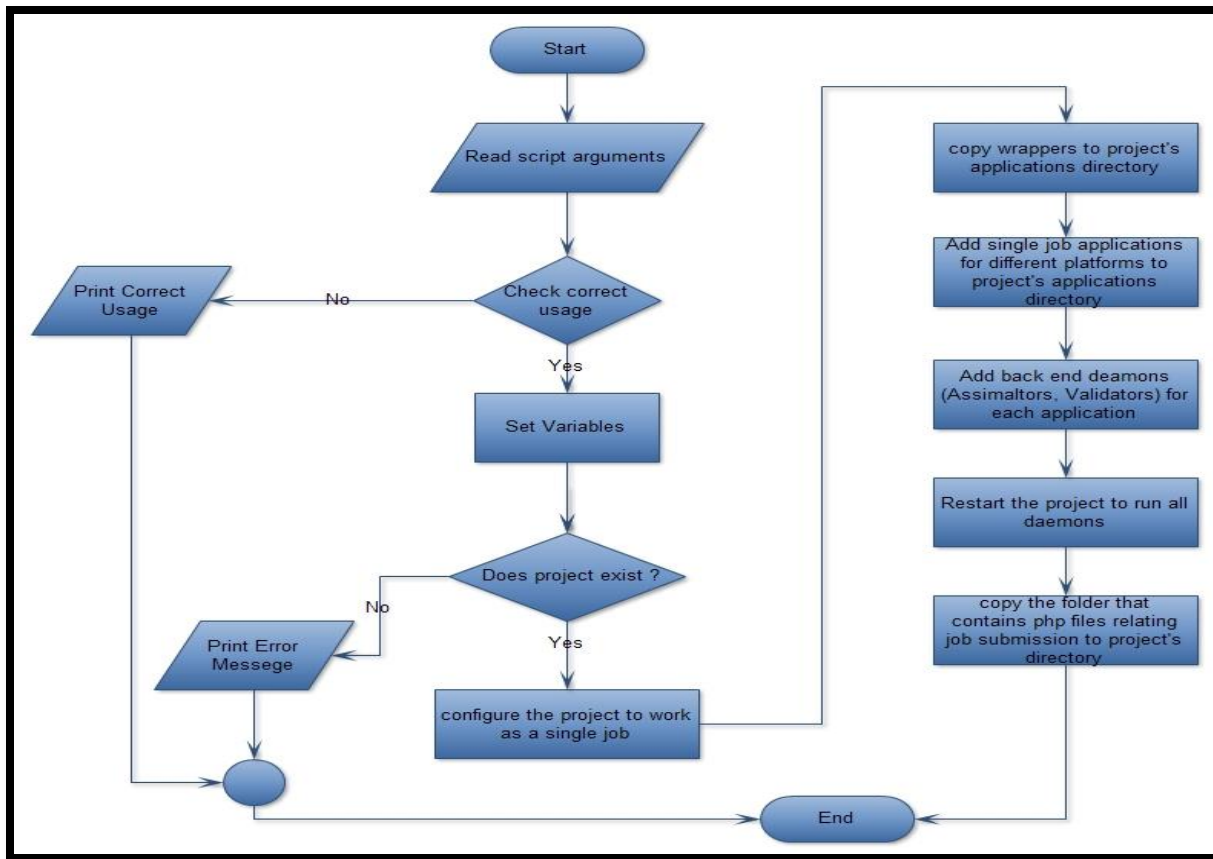


Figure 6.6: Single Job Configuration Algorithm

6.2.10.4 Job Submission

After configuring the project to work as a single job project, you can submit jobs for hosts that are running on any platform supported by your project. To do this, BOINC single job submission mechanism provides an executable PHP file that is called **boinc_submit**. This file can be used to submit a job as follows:

1. go to the path that contains the application(executable) and its input and output files:

```
cd /path/to/application
```

2. call **boinc_submit script (script path: ~/boinc/tools/boinc_submit)**

```
~/boinc/tools/boinc_submit [boinc-options] program [program-options]
```

The boinc-options are:

--infile name

specifies an input file.

--stdin name

direct the given file to the program's stdin.

--outfile name

specifies an output file.

--stdout name

direct the program's stdout to the given file.

--platform

the platform on which the program is to be run (default: the server's platform; assumed to be Linux).

Notes: [39]

- You can include as many **--infile** and **--outfile** options as you want, and at most one of others.
- The **program-options** will be passed as command-line arguments to the program when it runs on the remote machine.

- **--platform** option is followed with one of platform technical names shown in table 6.1.
- If the program requires any non-standard libraries, link these statically. Otherwise it will fail on machines that lack these libraries.
- You can run `boinc_submit` from any host that NSF-mounts your project directory and can access the MySQL database.
- When the job is completed successfully, the output files will appear in the job directory.

During the testing process, we found that **boinc_submit** script deals all command line arguments as one single argument. We explored the source code of this script carefully, and we were able to solve this problem.

In addition, we modified and created a new version of **boinc_submit**. The new modified version is called **ppu_boinc_submit**. This modification is done to facilitate the integration of job submission mechanism in our Grid System. The following points summarize the performed modifications:

1. The new version prints the wuid(job Id) as a return value. The job Id can be stored and used to check job status
2. This version solves a problem in the origin `boinc_submit` that has a bug which deal all commmand line arguments as one argument. To solve this problem, the line:
`$cmdline_args .= ".$argv[$i];` is replaced with `$cmdline_args .= ' '.$argv[$i];`
3. A general modification is done to make the file suitable to submit a job in PPU Grid System (e.g. remove calling some functions)

After a job is submitted we need a mechanism to monitor the job status. For this purpose we create an executable PHP script called **ppu_boinc_job** making

use of the functions provided in boinc_submit script. This script is used as follows:

- **ppu_boinc_job PHP script**

- **Usage: ppu_boinc_job project_dir job-options**

project_dir: is the path to the single job project

job-options:

--show_job

Show job id and whether the job is in progress, being assimilated or completed

--show_status jobID

Show job's status (show more information than '--show_job jobID')

--show_status2 jobID

the same as --show_status jobID but without showing jobID and current date.

--job_status jobID

shows job status without showing job id ('in progress','being assimilated' and 'completed')

--show_host

shows the instance(s) of Job location (on which host, the instance of job is being handled)

--wait jobID

Wait for the completion of an existing job

--abort jobID

Abort an existing job

--jobs

Show pending jobs

--help

Print this (print the usage)

The **ppu_boinc_submit** and **ppu_boinc_job PHP scripts** are usually used by single job projects so they are stored in ppu_boinc folder to facilitate the automation of single job configuration which is done by **singleJob.sh** bash script. The folder **ppu_boinc** is stored inside home directory (**\$HOME/ppu_boinc**).

When a singleJob.sh bash script is called to configure the project as a **single job project** the **ppu_boinc** folder is copied to project directory for later use. By doing this, each **single job project** will contain its own version of ppu_boinc folder that contains the two PHP scripts to submit a job and monitor its status.

6.2.10.5 Job Submission Automation

Calling ppu_boinc_submit.sh script differs according to the job to be submitted. Here are some examples:

- Job1: contains one input file (myInput), one output file (myOutput) and the executable file (program1). The job is submitted to the default platform (server platform).
ppu_boinc_submit --infile myInput --outfile myOutput program1
- Job2: contains two input file (in1 and in2), one output file (out1 and out2), and the executable file (program2).The job is submitted to the **windows_intelx86 platforms**.

```
ppu_boinc_submit --infile in1 --infile in2 --outfile out1 --outfile out2 --platform  
windows_intelx86 program2
```

- Job3: contains standard input file (input), standard output file (out1), one output file (out) and the executable file (program3). The job is submitted to computers with platform of type **x86_64-pc-linux-gnu**.

```
ppu_boinc_submit --stdin input --stdout out1 --outfile out2 --platform  
windows_intelx86 program3
```

As we can see from previous examples, there are different versions of calling the submission script (calling with different arguments) depending on the application to be submitted. For each job we must determine the following:

- **Input files:** each input file used by job executable must be stated after **--infile** option
- **Output files:** each output file used by job executable must be stated after **--outfile** option
- **Standard input file:** Any job that reads from standard input (keyboard) which is equivalent to “**cin**” statements in c/c++ must be redirected to a file since there is no user input in the grid system. The name of this file must be stated after **--stdin** option.
- **Standard output file:** Any job that reads from standard output (screen) which is equivalent to “**cout**” statements in c/c++ must be redirected to a file. The name of this file must be stated after **--stdout** option.
- **Platform name:** the name of the platform type that job executable is compiled to run on .The name of this platform comes after **--platform option**. In other words, this option determines the computers that can execute this job.
- **Executable file:** this is the executable file which is compiled to run on a particular platform type.

To automate the process of submitting a certain job, we put a set of guidelines (restrictions) on the structure of the job to be submitted. The submitted Job folder should follow the following conventions (guides) accurately:

- Contains one executable file for specific platform supported by the project.
- All other files are optional but if any of them exists, it must be compatible with the following guides:
 - Name of standard input file must be stdin.
 - Name of standard output file must be stdout.
 - All other input files must be located in a folder named inputs.
 - All other output files must be located in a folder named outputs.

After that we created **submit.sh** bash script which parses the job folder that is assumed to follow previous guides and generates the correct call (using correct arguments) of ppu_boinc_submit script. The work flow of **submit.sh** is described in the algorithm shown in the figure 6.7.

- **submit.sh bash script**

- **Usage: bash submit.sh projectName jobPath platform**

Where:

projectName: the name of the single job project under which the job is submitted.

jobPath: the path of the folder containing the job.

platform: is the platform name where the program can be executed. This argument is put after **--platform** option in ppu_boinc_submit.sh call.

Note: projects are assumed to be stores in /home/boincadm/projects so the project path is /home/boincadm/projects/**projectName**

- **Usage Example:** bash submit.sh SingleJobProject \$HOME/Jobs/job1 windows_intelx86

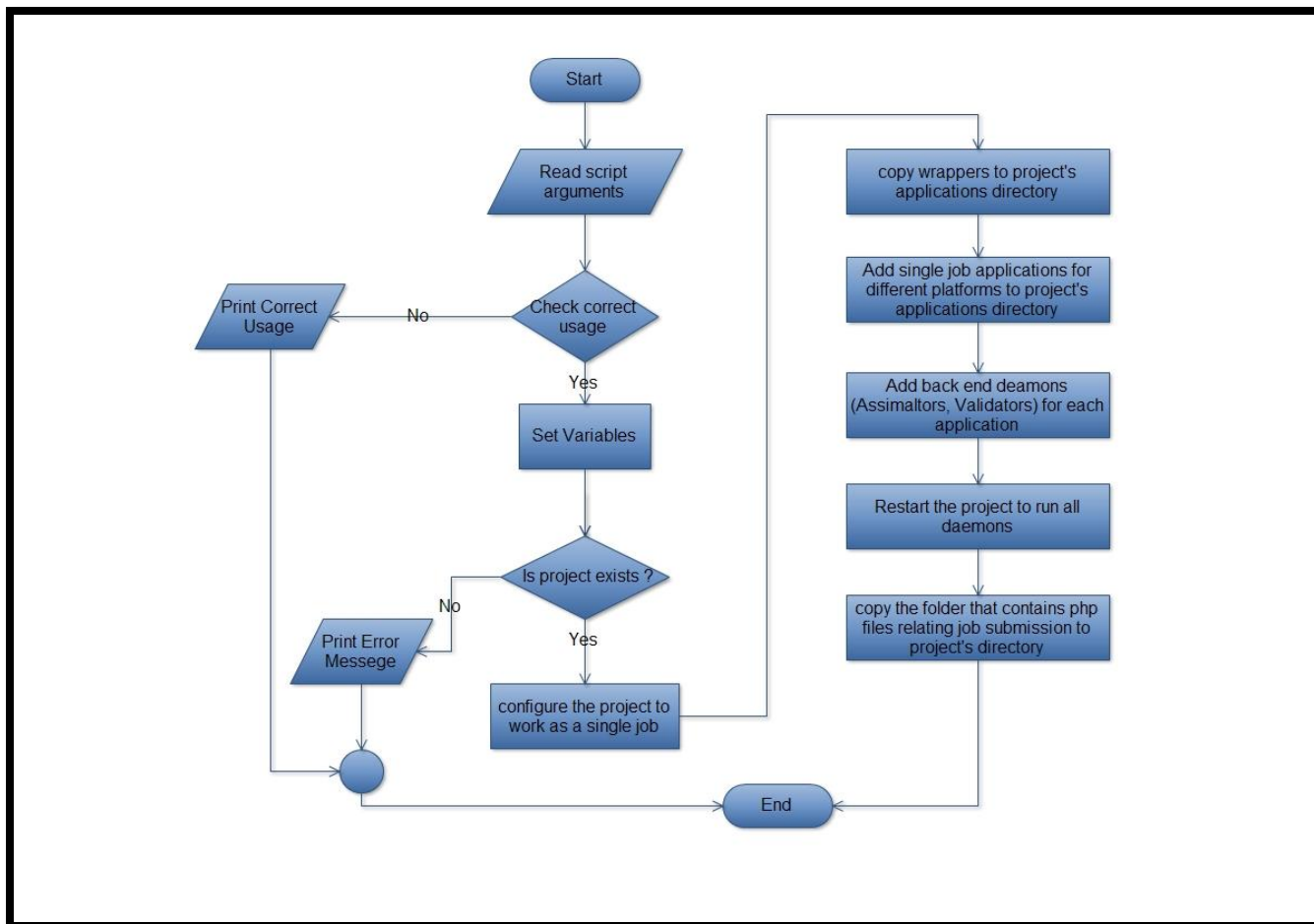


Figure 6.7: Submit Job Algorithm

Furthermore, we created another bash script called **output_handler.sh** to handle the result of the submitted job when it is completed and returned back to the server. This script put the output files, **job_summary** file, stdin and stdout files if any in a compressed result folder. The **output_handler.sh** implements the algorithm shown in the figure 6.8.

The **job_summary** file contains the final state of the job. If the job is executed successfully, then the job_summary file contains on which host (computer) the job is executed and the CPU time needed for executing that job. Otherwise, if the job is not executed successfully, then the job_summary file will contain an error message.

- **output_handler.sh bash script**

- **Usage:** `bash output_handler.sh projectName jobPath jobID`

Where:

projectName: the name of the single job project under which the job is submitted.

jobPath: the path of the folder containing the job.

jobID: the Job ID. The Job Id is returned when `ppu_boinc_submit.sh` is called.

Note: projects are assumed to be stores in `/home/boincadm/projects` so the project path is `/home/boincadm/projects/projectName`

- **Usage Example:** `bash output_handler.sh SingleJobProject $HOME/job1 1`

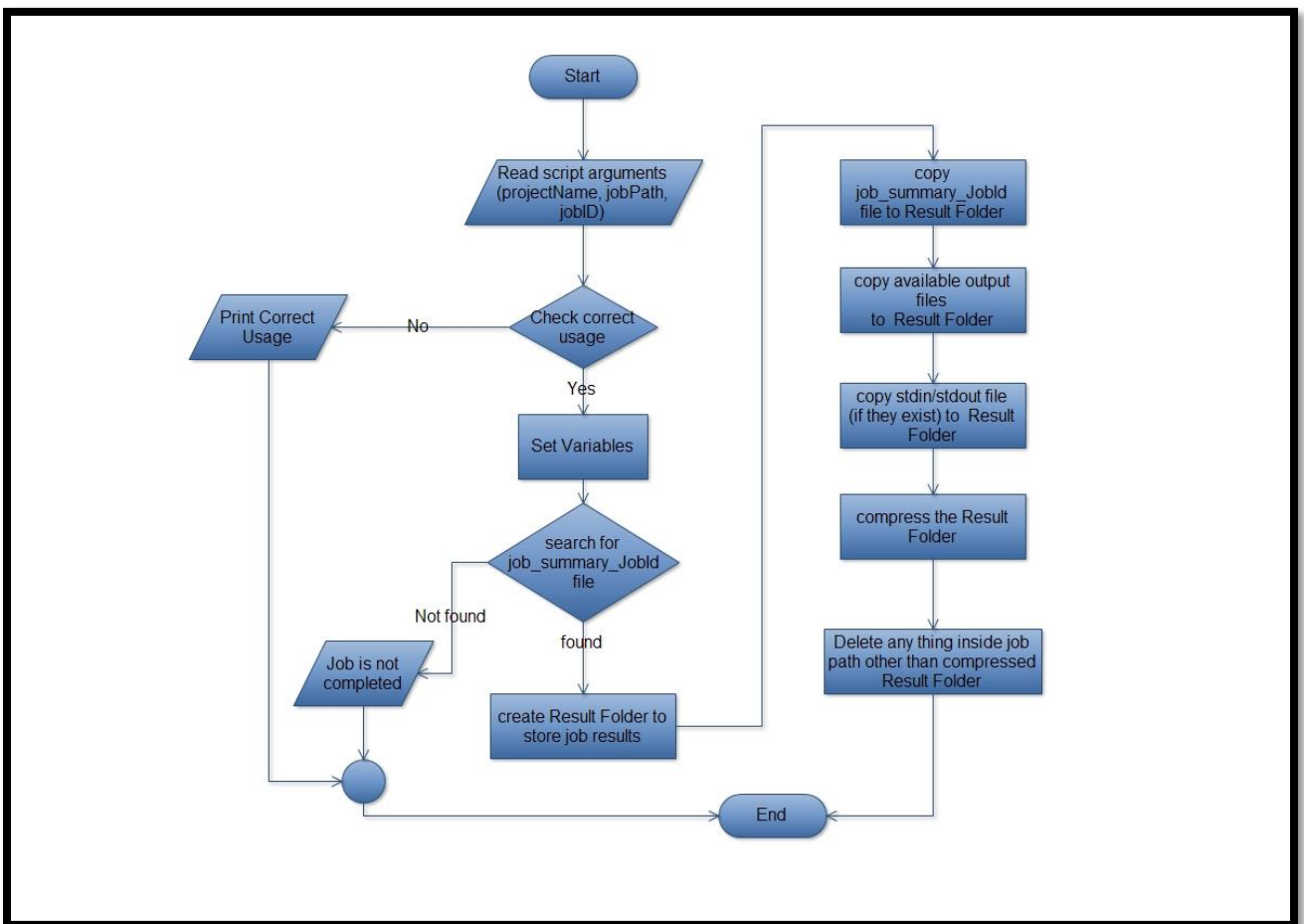


Figure 6.8: Output Handler Algorithm

6.3 Portal implementation

Here we clarify the system higher level implementation which is the portal implementation. Firstly, we talk about the software development tools that were used in implementing the portal. Secondly, we show the invisible pages and the sub-pages that support the main pages functionalities. Finally, we state the main pages of the portal and describe their functionalities.

6.3.1 Software development tools and programming languages

This subsection includes programming languages used to implement the portal and the IDE used. It also states the software servers needed to support the portal functionalities.

Programming languages used in the portal implementation:

- PHP.
- HTML.
- JavaScript.
- CSS.

The IDE used in the portal development:

- NetBeans7.4 IDE.

Software servers need to be installed on the portal server to enable the system functionalities:

- BOINC server.
- Apache server.
- MYSQL server.

6.3.2 Portal subpages:

Portal sub-pages are the pages that called by the main pages to perform a specific functionalities. These pages are:

1. **PageLeftPart:** this page required by all portal main pages. It displays information about the logged user. See figures below.

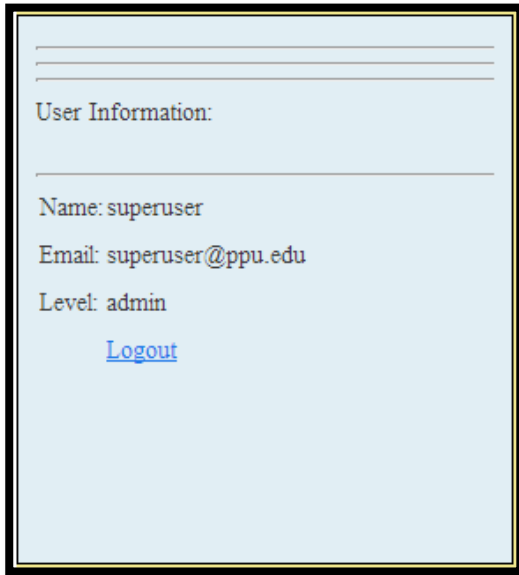


Figure 6.9: Admin page left part.

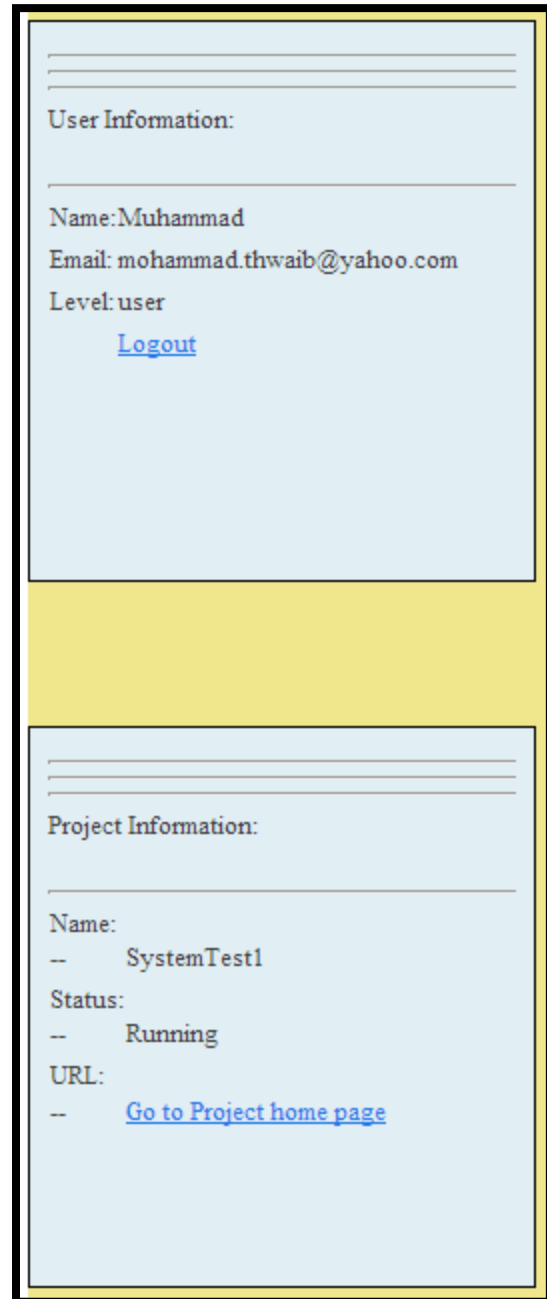


Figure 6.10: User page left part.

2. **PageRightPart:** this page supports the administrator main pages functionalities. It manages the content of the right side of the admin pages. **PageRightPart** declares objects from the classes **ProjectCreation**, **ProjectManagement**, and **UserCreation**- discussed in chapter five.

PageRightPart is required from three different pages:

2.1 Admin Home page: Requiring page Right Part from this page lunches calling **getForm()** - the function of the **projectCreation** object. The generated form is shown in figure 6.11.

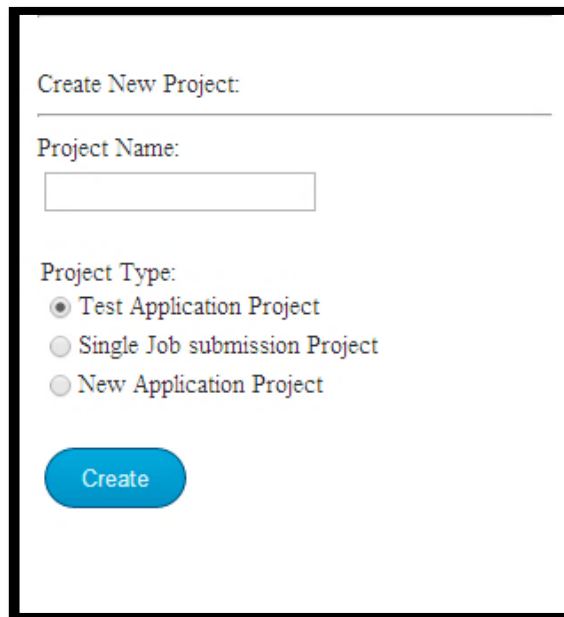


Figure 6.11: Project creation form.

Project creation process:

- ★ Admin requests creating a BOINC project by doing the following steps:
 - Enter a project name.
 - Select the project type.
 - Click on create button.
- ★ As a response for the admin request the following steps done:

- Calling the function **createProject(\$projectName,\$ProjectType,\$con)** – function of the ProjectCreation object.

Where:

- ❖ **\$projectName**: is a variable contains the name of the project.
- ❖ **\$projectType**: is a variable contains the type of the project.
- ❖ **\$con**: is the DB connection.

The function **\$createProject(..)** do the following:

- Validate the name of the project with a specific format- a valid project name must be only letters, digits, underscores.
- Generate error messages in the case of invalid project name. See figure 6.12.
- If the entered name is valid, the process goes into the steps below:
 - Add the project to the list of projects in the **grid_system** DB.
 - Execute the appropriate shell script; based on the selected project type. Execute the script **createTestProject.sh** for the first choice. Execute the script **createProject.sh** for the second and the last choices. Execute additional script for the second choice which is **singleJob.sh**, and create a directory named **users** in the project directory.
 - Create admin account following to this project by executing the script **createAccount.sh**.
 - Disable account creation by executing the script **disable_account_creation.sh**.

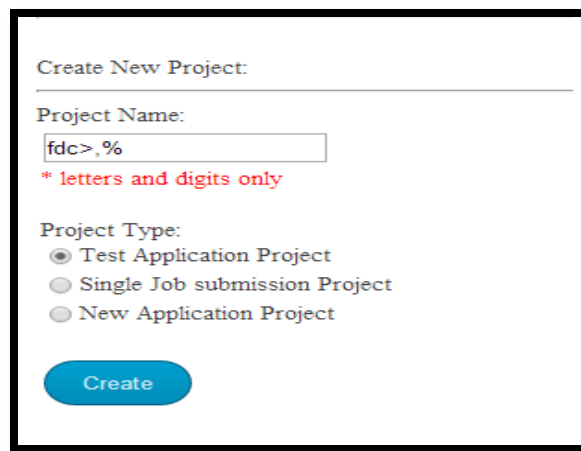
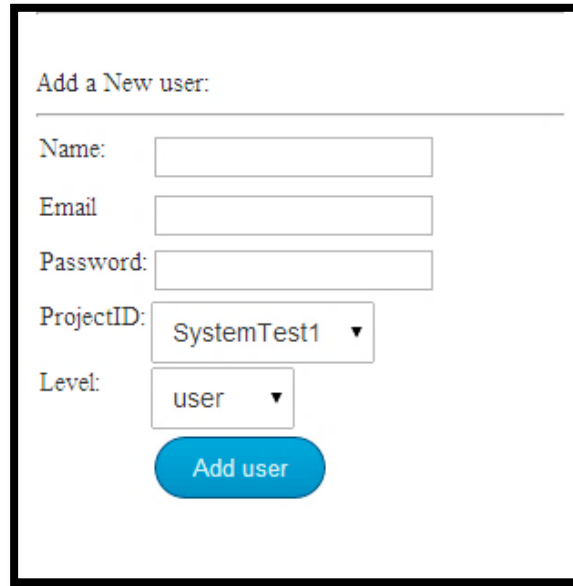


Figure 6.12: Project creation form.

2.2 Users page: Requiring page Right Part from Users page lunches calling `getForm()` - the function of the `userCreation` object. The generated form is shown in figure 6.13.



The image shows a web form titled "Add a New user:". It contains the following fields and controls:

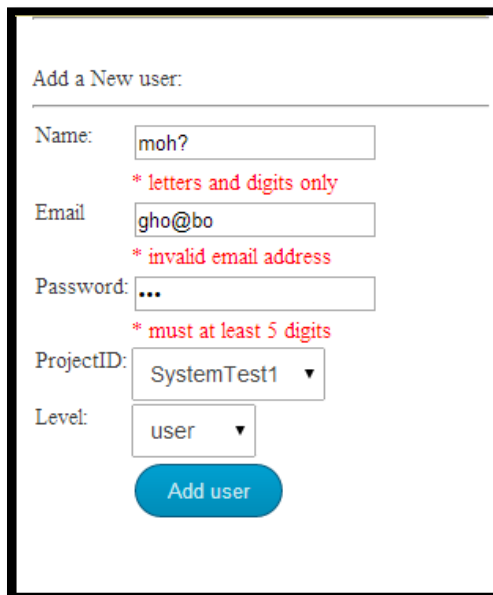
- Name:** A text input field.
- Email:** A text input field.
- Password:** A text input field.
- ProjectID:** A dropdown menu with "SystemTest1" selected.
- Level:** A dropdown menu with "user" selected.
- Add user:** A blue button.

6.13: Add new user form.

Adding a new user:

- Admin requests adding new user by doing the steps:
 - Enter the name of new user.
 - Enter the email.
 - Enter a password.
 - Select a Project to allow submitting jobs to that project from this new user.
 - Select the level of the user to be admin or user.
 - Click on add user button.
- System response to admin request by doing the steps:
 - Calling the function `createUser($con,$email,$userName,Password,$level)`- function of the `UserCreation` object.
- The function `createUser(...)` in tern calls the following functions:

- ❖ **validateName(\$userName)**: is a function of the **UserCreation** object. It checks the validity of the user name to a specific format (letters, digits, and underscores only). It also generates error messages in the case of mismatch the correct format.
 - ❖ **validateEmail(\$email)**: is a function of the **UserCreation** object. It checks the validity of the email and checks if this email exists in the DB or not. Generate error message in both cases: invalid email format or the email exist in the DB previously.
 - ❖ **validatePassword(\$password)**: is a function of the **UserCreation** object. It checks the validity of the password – valid password must be at least five digits.
- After calling these three functions, **createUser(..)** check if neither of the errors was generated, then perform the steps:
- ❖ Add the new user to the DB.
 - ❖ Create a directory for this user inside the users directory that is exist in the directory of the selected project.
 - In the case of errors exist the form will be re-generated with errors messages.



The screenshot shows a web form titled "Add a New user:". It contains several input fields with associated error messages in red text:

- Name:** Input field contains "moh?". Error message: "* letters and digits only".
- Email:** Input field contains "gho@bo". Error message: "* invalid email address".
- Password:** Input field contains "...". Error message: "* must at least 5 digits".
- ProjectID:** Dropdown menu with "SystemTest1" selected.
- Level:** Dropdown menu with "user" selected.

At the bottom of the form is a blue button labeled "Add user".

6.14: Add new user form.

2.3 Project management page: Requiring page Right Part from Project Management page lunches calling **getForm()** - the function of the **projectManagement** object. The generated form is shown in figure 6.15.

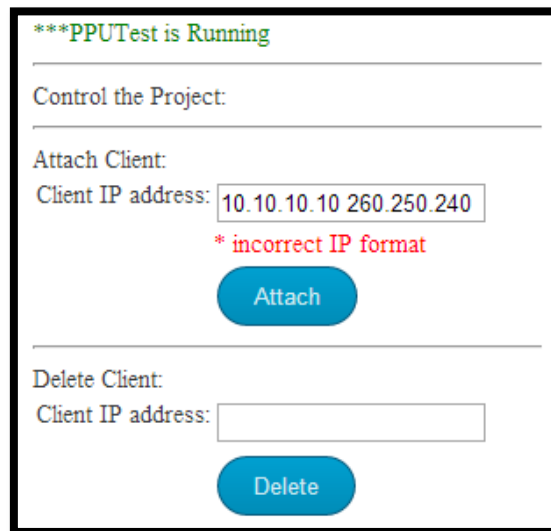
The screenshot shows a web form titled "Control the Project:" with several sections separated by horizontal lines. At the top, it says "***PPUTest is Running". The sections are: "Attach Client:" with a "Client IP address:" input field and an "Attach" button; "Delete Client:" with a "Client IP address:" input field and a "Delete" button; "Update the attached Projects on client:" with a "Client IP address:" input field and an "Update" button; "Restart the Project, Restart All diamons:" with a "Restart" button; "Enable account creation:" with an "Enable" button; "Stop the Project: All running diamons will be stopped:" with a "Stop" button; and "Delete the Project: Project will be deleted permanently:" with a "Delete" button.

6.15: Project management form.

Through the project management form, admin can attach or de-attach computing resources to the project, update the projects running on a specific resource, start the project if it is stopped and stop the project if it is running, disable account creation if it is enabled and enable account creation if it is disabled, and he also can delete the project.

Attaching new resource/s:

- Admin enter the IP address/es of the computing resource/s to be attached then click on attach button.
- System calls a function **attachClient(\$ipList)** - a function of projectManagement object.
- **attachClient(\$ipList)** calls the function **validateIPs(\$ipList)**.
- **validateIPs(\$ipList)** expands the \$ipList in to array of IP's and calls the function **validateIP(\$ip)** for each array element.
- **validateIP(\$ip)** checks the IP format and generate error message in the case of invalid IP.
- When error appears no resources will be attached and error messages will be displayed like the figure 6.16.
- At the case of no errors the function will attach the resource/s by calling the script **attach.sh**.



***PPUTest is Running

Control the Project:

Attach Client:

Client IP address:

* incorrect IP format

Attach

Delete Client:

Client IP address:

Delete

Figure 6.16: Client attachment.

Detaching a computing resource:

- ✱ Admin enter the IP address/es of the computing resource/s to be deleted from the computing resources list then click on delete button.
- ✱ System calls a function **deattachClient(\$ipList)** - a function of **projectManagement** object.
- ✱ This function calls the same function as in the attachment process.
- ✱ When error appears no resources will be de-attached and error messages will be displayed; same as in the attachment process.
- ✱ At the case of no errors the function will delete the attached resource/s by calling the script **deattach.sh**.

Updating projects at a client:

- ✱ Admin enter the IP address of the computing resource to update the attached projects then click on update button.
- ✱ System calls a function **updateProject (\$ipAddress)** - a function of **projectManagement** object.
- ✱ **updateProject (\$ipAddress)** calls the function **validateIP(\$ipAddress)** – stated above.
- ✱ When error appears error messages will be displayed; same as in the attachment process.
- ✱ At the case of no errors the function will update the attached projects by calling the script **updateAttachedProjects.sh**.

Stopping project:

- ✱ Admin clicks on stop button.
- ✱ System stops the project by executing the script **stopProject.sh** and modifies the project status to be 'not running' in the DB.

Starting project:

- ✱ Admin clicks on start button.

- ✿ System starts the project by executing the script **startProject.sh** and modifies the project status to be 'running' in the DB.

Restarting project:

- ✿ Admin clicks on restart button.
- ✿ System restarts the project by executing the script **resartProject.sh** and modifies the project status to be 'running' in the DB.

Disabling account creation:

- ✿ Admin clicks on disable button.
- ✿ System disables account creation by executing the script **disable_account_creation.sh** and modifies the account_creation_status to be 'disabled' in the DB.

Enabling account creation:

- ✿ Admin clicks on enable button.
- ✿ System enables account creation by executing the script **enable_account_creation.sh** and modifies the account_creation_status to be 'enabled' in the DB.

Deleting project:

- ✿ Admin clicks on delete button.
- ✿ System deletes the project directory and the project DB by executing the script **deleteProject.sh**.
- ✿ System deletes the project from the projects table.
- ✿ System deletes all jobs following to this project.
- ✿ System sets the project_name to all users following to this project to 'NULL'.

3. **Header:** manages pages headers, menu, and the slide bar. Figure below show the header of the home page.



Figure 6.17: Header subpage.

4. **Footer:** displays the footer in all pages as in the figure 6.18.



Figure 6.18: Footer subpage.

5. **Logout:** destroys the existent session and logs the user out.
6. **AjaxResponse:** used to asynchronously abort the selected job – abort the job while processing other functionality in the main page without stopping. Aborting jobs is done by executing the PHP script called **abort_job**.
7. **Download:** enable downloading results.

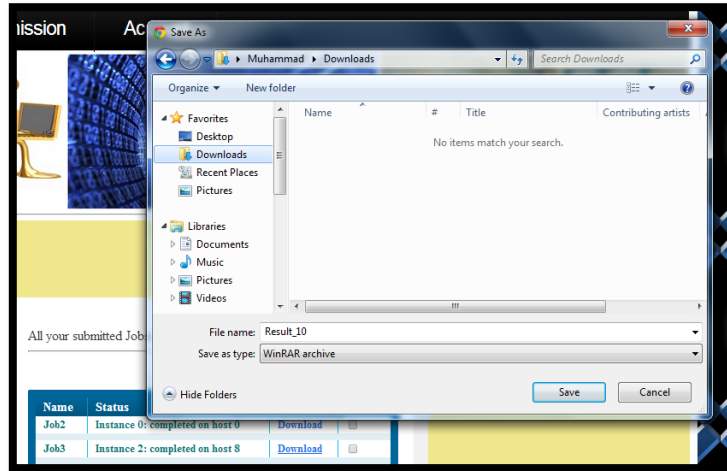


Figure 6.19: Download result.

- 8. Results:** supports the functionality of the user Home page by doing the functions:
- * Displays all submitted jobs and their execution status. The job execution status is retrieved by executing the PHP script **show_status**.
 - * Enable aborting jobs during the execution using **AjaxResponse** page.
 - * Enable downloading results using **Download** page.
 - * Enable deleting results.

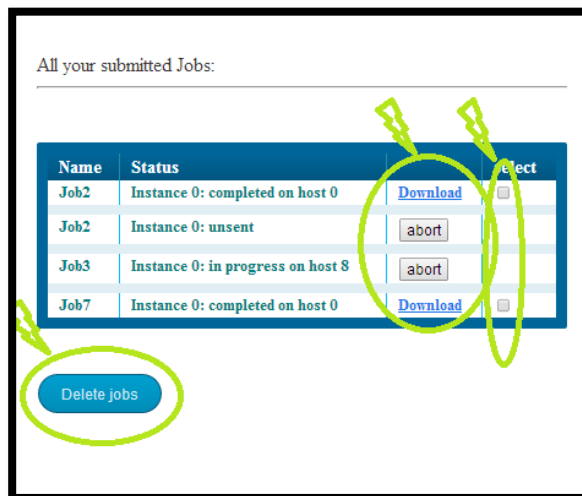


Figure 6.20: Results subpage.

9. **Config:** declares the global variables needed in all pages.

6.3.3 Portal main pages

The portal main pages are classified into three types:

- 1- Common pages.
- 2- Admin pages.
- 3- User pages.

1. Common pages:

The common pages are displayed for all visitors and the system users before login. These pages are:

- **About:** provides basic definitions of the grid computing, the importance of grid computing, and the project future vision.
- **Contact:** states more dedicated information about the project team, project organization, and the middleware used to build the grid system.
- **Login:** authenticates the access to the system for authorized users.

2. Admin Pages:

Admin Pages is the pages that can be accessed by the admin users only. Those pages are:

- **Home:** requires the **PageRightPart** and **PageLeftPart**. Also it displays all existing projects with links for projects management. See figure 6.21.

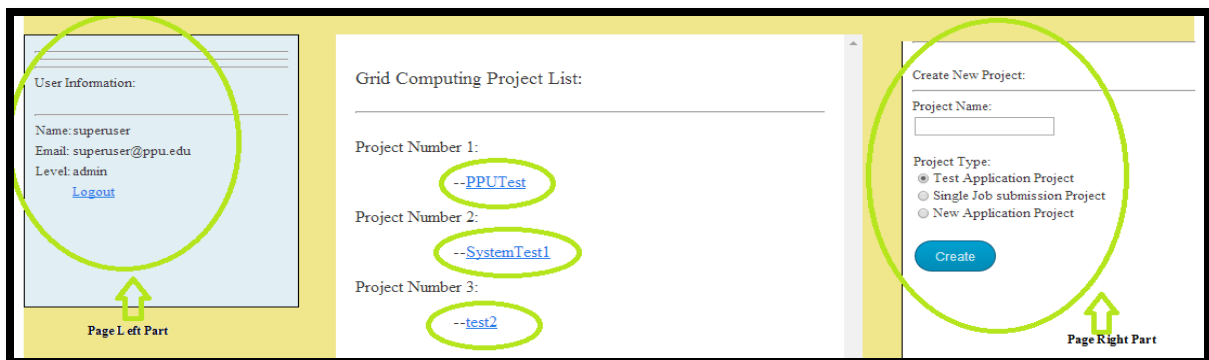


Figure 6.21: Admin home page.

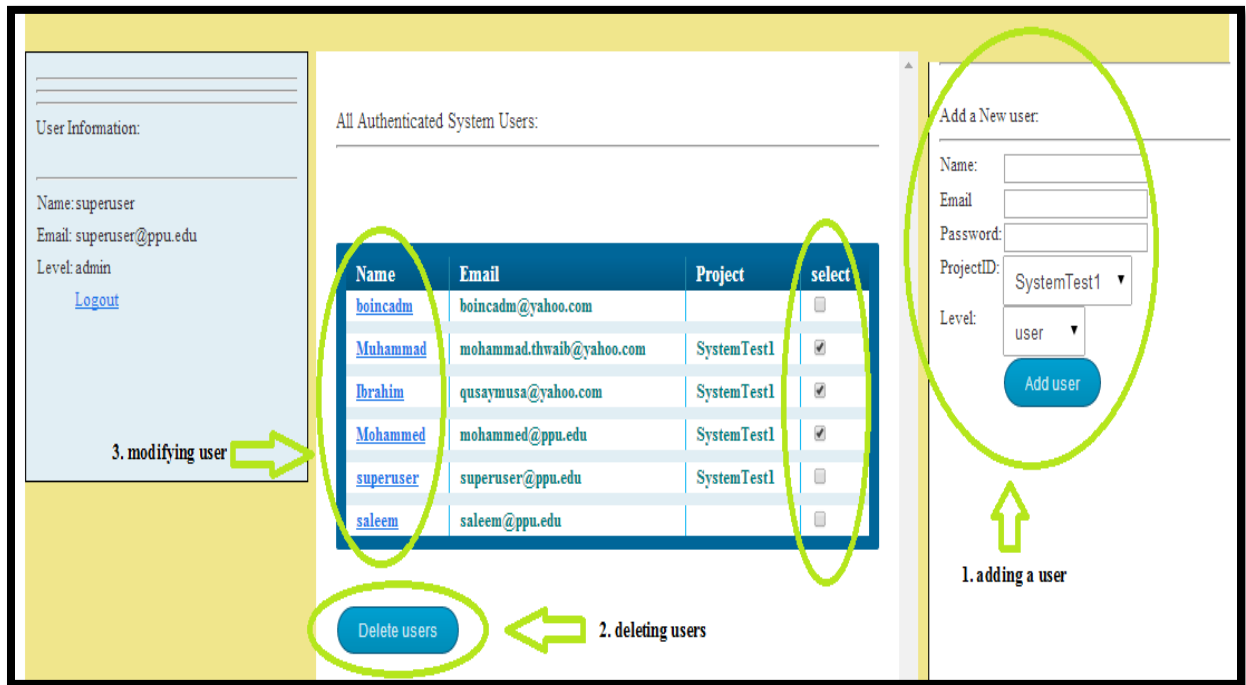


Figure 6.22: Users page.

- **Users:** requires the **PageRightPart** and **PageLeftPart**. Also it displays all system users with links for modify their information. See figure 6.22.
- **Account:** displays the account information and enable the admin to modify his them.
- **Project management:** requires **PageRightPart**, **PageLeftPart**, and states basic information about the project with important links to the main pages of the project.
- **Modify user information:** enable the admin to modify the account information for any user.

3. User pages:

- **Home:** requires **PageRightPart**, **PageLeftPart**, and **Results** pages.
- **Job submission:**
 - * Requires **PageLeftPart**.
 - * Display the submission form.
 - * Validate the inputs.

- * Generate error messages.
 - * Upload the job folder.
 - * Extract the job folder and generate a unique name.
 - * Submit job for execution and add a new entry to the jobs table.
- **Account:** displays the account information and enable the user to modify his them.

6.4 Security Issues

In this section, we talk about the project's security issues. We explain the security mechanisms supported by BOINC. In addition, we show our security measures to make our grid system project secure.

6.4.1 Securing the Server and the Clients

To secure the grid server, BOINC advises to read and implement the UNIX Security Checklist 2.0 from AusCERT and CERT/CC. According to that we can give the proper permissions for users, groups and others to ensure the security.

In addition, we can put the server and all client computers behind the university firewall that lets through minimal traffic (e.g., HTTP and SSH where needed). Also, we read about MySQL general security guidelines, and we follow these guides to make MySQL server as secure as possible.

At the client side, we installed the BOINC core client as a service on admin account and we prevent other users (usually students) from controlling the boinc client software. Only the admin can control the BOINC client software.

In addition, BOINC was configured to use account-based sandboxing - that is, to run project applications under an unprivileged account [70]. This is the

default on Android, Mac OS X and on the installers provided by Linux distributions. Currently, it is not the default on Windows because GPU applications can't run under unprivileged accounts [70]. However, we enabled it on Windows by checking the “**Service Install**” checkbox in advanced option during installation.

On the other side, the developers of BOINC applications should make sure that the application does not become infected and secure their source-code repository. Also, they should read about Secure Programming for Linux and UNIX, especially if the application does network communication.

6.4.2 Client/ Server Authentication and Authorization

The **BOINC client** typically is controlled by the **BOINC Manager** or the **Boinc cmd tool** (see appendix D) running on the same machine. The two programs communicate over a local connection, using 'GUI RPC' (Graphical User Interface Remote Procedure Call). It is also possible to use the BOINC Manager to control a client on a different host.

The BOINC command tool (boinc cmd) provides a command-line interface to a running BOINC client (local or remote). This provides an alternative to the BOINC Manager, e.g. on systems with no graphics display.

In our grid system we want to run the core client software silently in the background without any user intervention; so we used **Boinc cmd tool** to communicate with the core client instead of **BOINC Manager**.

We continuously need to communicate with BOINC core clients remotely from the server. This communication is needed to control our computers like attaching them to our projects in addition to get their status and for other purposes (see appendix D for more information). This remote communication (remote RPC) must be secure and safe.

To make the system secure, all remote RPCs are authenticated using the GUI RPC password which is stored in **gui_rpc_auth.cfg** file inside BOINC data directory on

each computer participating in our grid. By default, remote RPCs are not accepted from any host. To specify a set of hosts from which RPCs are allowed, we created a file **remote_hosts.cfg** in BOINC data directory containing a list of allowed DNS host names or IP addresses (one per line). In our case, this file contains the server IP address (10.10.16.12). Therefore, only the server will be able to connect to core client software installed on our computers.

For example if a host A wants to communicate with a BOINC core client on a remote computer X, host A must satisfy two conditions. First, the IP of host A should be included in **remote_hosts.cfg** file on computer X. Second, host A should give GUI RPC password of the computer X in his RPC command (using boinccmd tool). If any of previous conditions is missed, then host A will not be able to connect to machine X.

6.4.3 Protecting Administrative web interface

Each BOINC project has an **administrative web interface** that lets you [40]:

- Browse the database
- Screen user profiles
- Administer "special users" (e.g., forum moderators)
- Create and edit applications and app versions
- Send mass email to users (note: a more flexible way of doing this is described here).
- Send emails to users with malfunctioning hosts.
- See a distribution of how many FLOPs results are using.
- Cancel work units
- View recent results, and analyze failures
 - Browse strip charts
 - Browse log files

If project's URL is for example <http://a.b.c.d/test>, then the URL of the admin web interface is http://a.b.c.d/test_ops. The directory containing the admin pages is `~/projects/test/html/ops/`.

Because the admin interface lets you do things like see user email addresses, it's extremely important that it be secure. There are two levels of protection: **protection by .htaccess** and **Project-defined protection policy**. These two techniques are clarified in appendix C.

6.4.4 Securing the Web Portal

Web portal provides interfaces for administrator and users. Administrator interfaces provide critical information and control for the overall system. Users interfaces display information about the user account and enable all the user functionalities described previously. Since of that, system portal must be secure enough to prevent any unauthorized access. Below we describe briefly the mechanisms implemented to make the system portal secure.

One of the expected attacks to the system through the portal is the SQL code injection. SQL code injection is a code injection technique, in which malicious SQL statements are inserted into an entry field for execution. These SQL statements may cause for example deleting the system DB. To secure the portal against this type of attack, we implement validation mechanisms that handle all the user inputs and insure that the inputs match the proper formats before making any execution on these inputs. It also returns error messages in the case of invalid formats of inputs.

It is important that any user needs to access to the system must firstly prove his authenticity using his email address and password in the login page. In addition the same user mustn't be able to login from different locations at the same time. The criteria implemented in this system that the last login to a user from a specific location is the only active login to that user; if a user login to the system from a location then the same user login from another location then the first login will be destroyed directly.

All passwords are encrypted before storing them in the DB using MD5 encryption mechanism. So, when a user enters his password for login this password encrypted firstly then compared with the password stored in the DB.

6.5 Testing

In this section, we explain the testing procedure at two levels. At the first level (the lower level), we test the functionality of each shell script independently. At the second level (the higher level), we test the integration of the portal functionalities with underlying level of the system.

6.5.1 Testing system functionalities (Testing lower level)

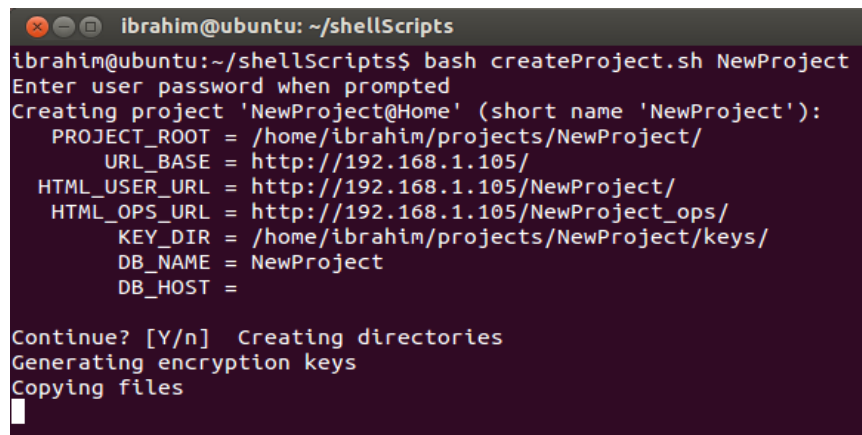
In this subsection we provide figures that were captured during the testing of the functionality of the shell scripts.

1. Testing the script `createProject.sh`

The command used in testing this script was:

```
bash creatProject.sh NewProject
```

The execution of this command will create a BOINC project named **NewProject**. If the project name is not set in the command the script will handle this error and display a help message. Also the script takes another optional parameter, that is, the root directory of the **NewProject** where the project directory tree will be stored. If the user enters an invalid directory, the script will handle this error and display a help message. Figures below show the output of the execution of this script in two cases: in the case of correct usage and in the case of missing the name of the project in the command.



```
ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash createProject.sh NewProject
Enter user password when prompted
Creating project 'NewProject@Home' (short name 'NewProject'):
  PROJECT_ROOT = /home/ibrahim/projects/NewProject/
  URL_BASE = http://192.168.1.105/
  HTML_USER_URL = http://192.168.1.105/NewProject/
  HTML_OPS_URL = http://192.168.1.105/NewProject_ops/
  KEY_DIR = /home/ibrahim/projects/NewProject/keys/
  DB_NAME = NewProject
  DB_HOST =

Continue? [Y/n] Creating directories
Generating encryption keys
Copying files
█
```

Figure 6.23: Create Project case 1.

After executing the script, the project home page looks like shown in the figure below.

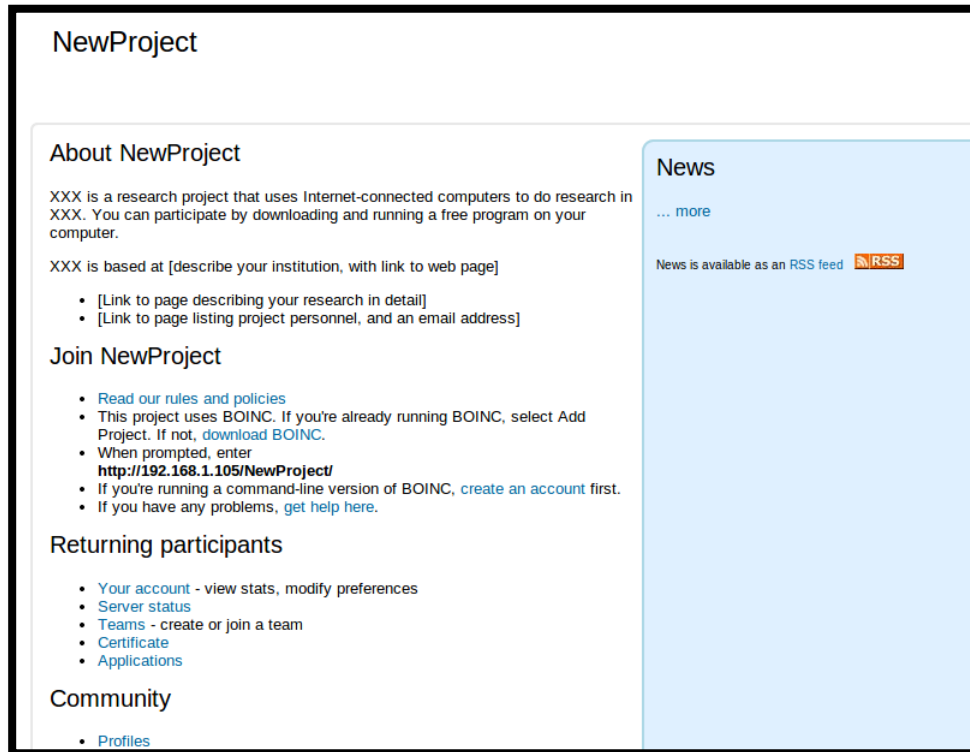


Figure 6.24: Project home page.

```
muhammad@ubuntu: ~/shellScripts
muhammad@ubuntu:~/shellScripts$ bash createProject.sh
Usage: createProject.sh projectName [installroot]
projectname: the name of the project
installroot: the path of the folder containing the project
installroot: is optional if not determined then the default is used
installroot: the default of installroot is /home/muhammad/projects
Usage Example1 : createProject.sh test
Usage Example2 : createProject.sh test /home/muhammad/projects
muhammad@ubuntu:~/shellScripts$
```

Figure 6.25: Project creation case 2(wrong usage).

```
boincadm@ubuntu: ~/shellScripts
boincadm@ubuntu:~/shellScripts$ bash createProject.sh PPUTest2 home/projects
Enter user password when prompted
mkdir: cannot create directory 'home/projects': No such file or directory
createProject.sh: line 106: cannot create temp file for here-document: No space
left on device
The directory 'home/projects/PPUTest2' is not existing
Error, do not continue.
boincadm@ubuntu:~/shellScripts$
```

Figure 6.26: Project creation case 3(wrong usage).

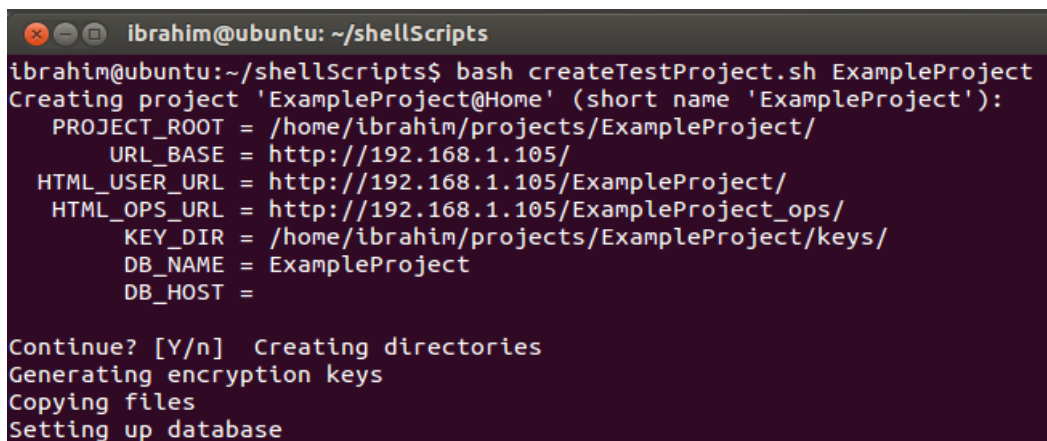
All of our scripts support error handling mechanisms. Since there are too many cases of the errors that can occur and need to be handled by the scripts; it will not be easy to show all of these cases within this subsection. All of the remaining tests show only the output of the scripts in the case of correct usage of their execution commands.

2. Testing the script **creatTestProject.sh**

The execution command:

```
bash creatTestProject.sh ExampleProject
```

The result of execution is creating the Test example project. Figure 6.27 shows the output of the script execution.

A terminal window screenshot showing the execution of the script creatTestProject.sh. The terminal title is 'ibrahim@ubuntu: ~/shellScripts'. The command entered is 'bash creatTestProject.sh ExampleProject'. The output shows the script creating a project named 'ExampleProject@Home' with a short name 'ExampleProject'. It lists several configuration variables: PROJECT_ROOT, URL_BASE, HTML_USER_URL, HTML_OPS_URL, KEY_DIR, DB_NAME, and DB_HOST. The script then prompts for confirmation to continue, and proceeds with creating directories, generating encryption keys, copying files, and setting up the database.

```
ibrahim@ubuntu:~/shellScripts$ bash creatTestProject.sh ExampleProject
Creating project 'ExampleProject@Home' (short name 'ExampleProject'):
  PROJECT_ROOT = /home/ibrahim/projects/ExampleProject/
  URL_BASE = http://192.168.1.105/
  HTML_USER_URL = http://192.168.1.105/ExampleProject/
  HTML_OPS_URL = http://192.168.1.105/ExampleProject_ops/
  KEY_DIR = /home/ibrahim/projects/ExampleProject/keys/
  DB_NAME = ExampleProject
  DB_HOST =

Continue? [Y/n] Creating directories
Generating encryption keys
Copying files
Setting up database
```

Figure 6.27: Create Test Project.

3. Testing the script **creatAccount.sh**

The execution command:

```
bash creatAccount.sh ExampleProject
```

Figure 6.28 shows the output of the script execution.

```
ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash createAccount.sh ExampleProject
status: Success
poll status: operation in progress
account key: 7999f42f57b06a2acb42a708deda30eb
ibrahim@ubuntu:~/shellScripts$
```

Figure 6.28: Create Account.

4. Testing the script **attach.sh**

The execution command:

```
bash attach.sh ExampleProject 192.168.1.100
```

Figure 6.29 show the output of the script execution.

```
ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash attach.sh ExampleProject 192.168.1.100
ibrahim@ubuntu:~/shellScripts$
```

Figure 6.29: Attach client.

Figure 6.30 shows the BOINC manager interface after attaching the client.

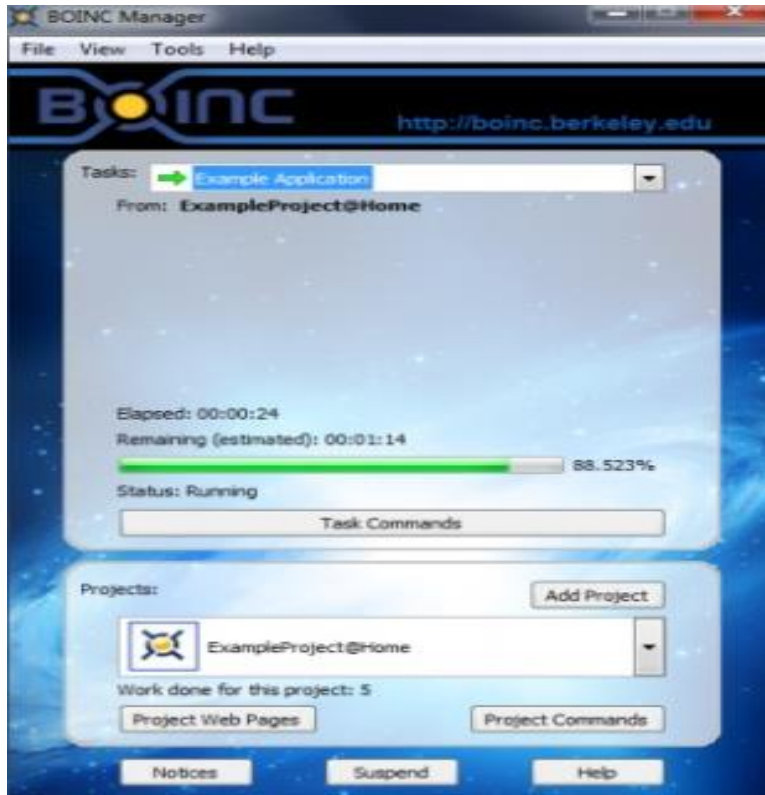


Figure 6.30: Attached client BOINC manager.

5. Testing the script **update.sh**

The execution command:

```
bash updateAttachedProject.sh 192.168.1.105
```

Figure 6.31 shows the script execution.

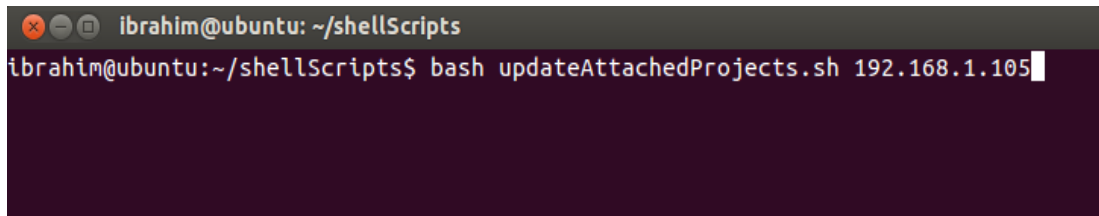


Figure 6.31: Update attached projects.

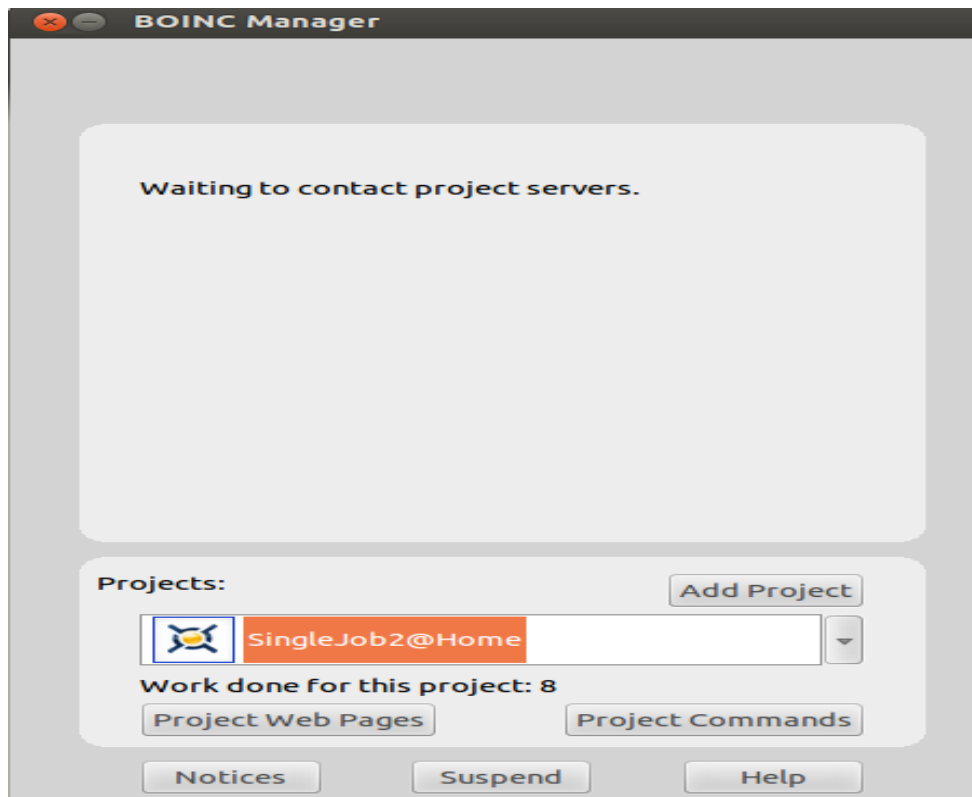


Figure 6.32: BOINC manager after executing updateAttachedProject.sh

6. Testing the script **deattach.sh**

The execution command:

```
bash deattach.sh ExampleProject 192.168.1.105
```

Figure 6.33 shows the output of the script execution.

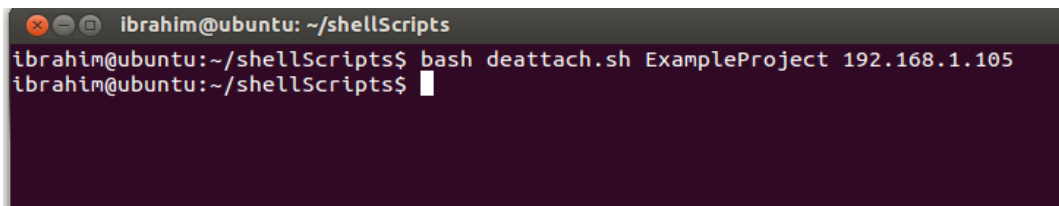


Figure 6.33: Detach a client.

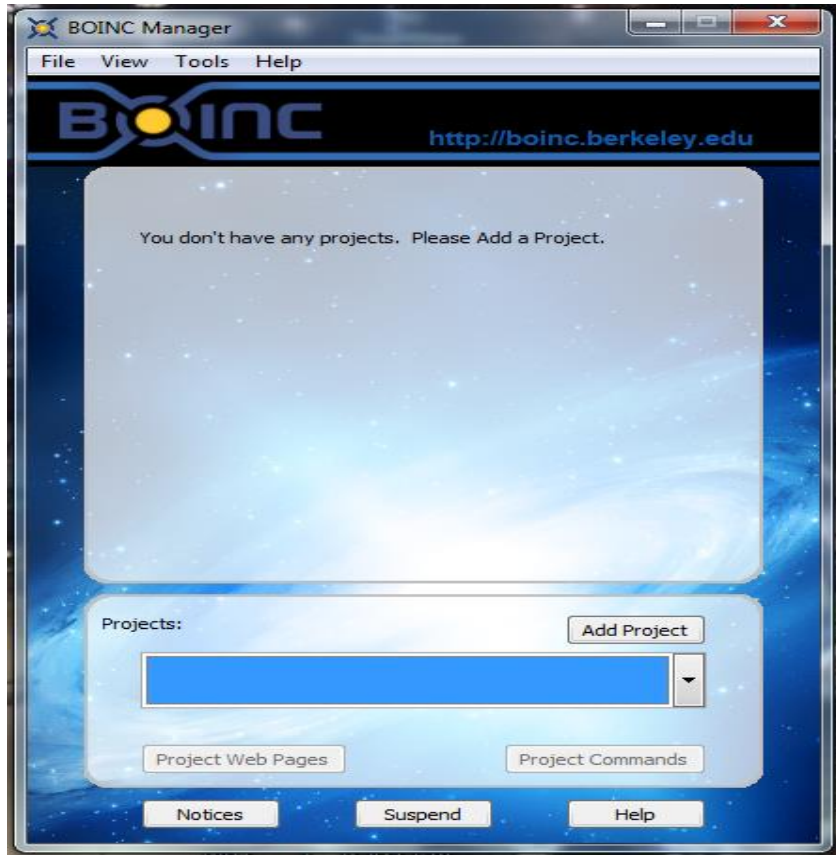


Figure 6.34: BOINC manager after executing deattach.sh.

7. Testing the script **stopProject.sh**

The execution command:

```
bash stopProject.sh ExampleProject
```

Figure 6.35 shows the output of the script execution.

```
ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash stopProject.sh ExampleProject
entering DISABLED mode
topping all daemons
Killed process 5870
Killed process 5872
Killed process 5885
Killed process 5876
Killed process 5882
Killed process 5879
Waiting for process 5870 to end: . ok
ibrahim@ubuntu:~/shellScripts$
```

Figure 6.35: Stop project.

8. Testing the script **startProject.sh**

The execution command:

```
bash startProject.sh ExampleProject
```

Figure 6.36 shows the output of the script execution.

```
ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash startProject.sh ExampleProject
Entering ENABLED mode
Starting daemons
Starting daemon: feeder -d 3
Starting daemon: transitioner -d 3
Starting daemon: file_deleter -d 3
Starting daemon: sample_work_generator -d 3
Starting daemon: sample_bitwise_validator -d 3 --app example_app
Starting daemon: sample_assimilator -d 3 --app example_app
ibrahim@ubuntu:~/shellScripts$
```

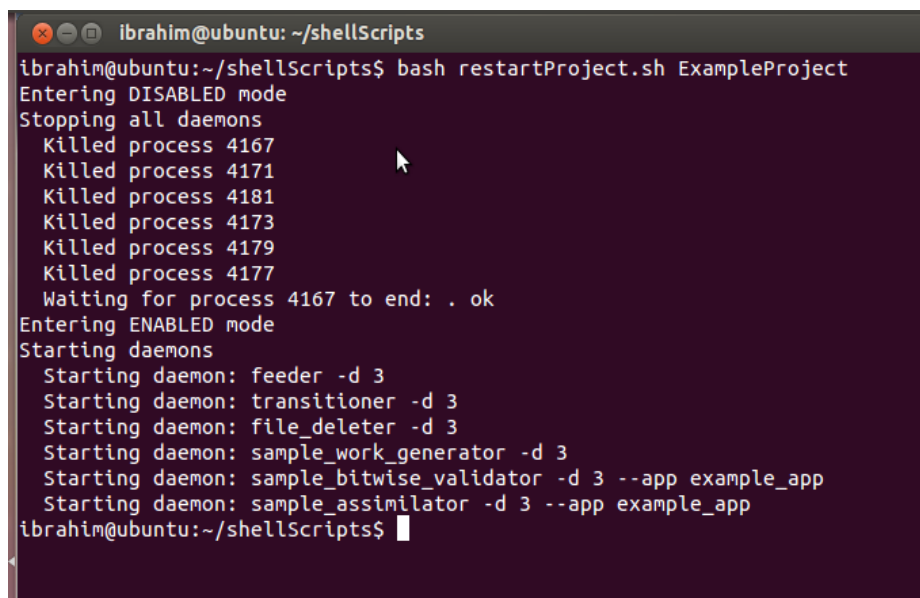
Figure 6.36: Start project.

9. Testing the script **restartProject.sh**

The execution command:

```
bash restartProject.sh ExampleProject
```

Figure 6.37 shows the output of the script execution.



```
ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash restartProject.sh ExampleProject
Entering DISABLED mode
Stopping all daemons
  Killed process 4167
  Killed process 4171
  Killed process 4181
  Killed process 4173
  Killed process 4179
  Killed process 4177
  Waiting for process 4167 to end: . ok
Entering ENABLED mode
Starting daemons
  Starting daemon: feeder -d 3
  Starting daemon: transitioner -d 3
  Starting daemon: file_deleter -d 3
  Starting daemon: sample_work_generator -d 3
  Starting daemon: sample_bitwise_validator -d 3 --app example_app
  Starting daemon: sample_assimilator -d 3 --app example_app
ibrahim@ubuntu:~/shellScripts$
```

Figure 6.37: Restart project.

10. Testing the script **singleJob.sh**

The execution command:

```
bash singleJob.sh singleJob
```

Figure 6.38 shows the output of the script execution.

```

ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ ./singleJob.sh singleJob
Installing current wrapper.
- type 'bin/update_versions', and answer 'y' to all questions.
- Add the following to the <daemons> section of config.xml:

<daemon>
  <cmd>single_job_assimilator -app single_job_x86_64-pc-linux-gnu</cmd>
  <output>single_job_assimilator_x86_64-pc-linux-gnu.out</output>
  <pid>single_job_assimilator_x86_64-pc-linux-gnu.pid</pid>
</daemon>
<daemon>
  <cmd>sample_trivial_validator -app single_job_x86_64-pc-linux-gnu</cmd>
  <output>sample_trivial_validator_x86_64-pc-linux-gnu.out</output>
  <pid>sample_trivial_validator_x86_64-pc-linux-gnu.pid</pid>
</daemon>
Then restart your project by typing
bin/stop
bin/start

Found application version: single_job_x86_64-pc-linux-gnu 1.0 x86_64-pc-linux-gn
U
platform not found: x86_64-pc-linux-gnu
Processing <Platform#None windows_intelx86> ...
Committed <Platform#1 windows_intelx86> ; values:

```

Figure 6.38: Customize project to single Job project.

Project status

21 May 2014, 9:20:40 UTC

Server status

Program	Host	Status
data-driven web pages	ubuntu	Running
upload/download server	ubuntu	Running
scheduler	ubuntu	Running
feeder	ubuntu	Running
transitioner	ubuntu	Running
file_deleter	ubuntu	Running
single_job_assimilator_x86_64-pc-linux-gnu	ubuntu	Running
sample_trivial_validator_x86_64-pc-linux-gnu	ubuntu	Running
single_job_assimilator_windows_intelx86	ubuntu	Running
sample_trivial_validator_windows_intelx86	ubuntu	Running
single_job_assimilator_windows_x86_64	ubuntu	Running
sample_trivial_validator_windows_x86_64	ubuntu	Running
single_job_assimilator_i686-pc-linux-gnu	ubuntu	Running
sample_trivial_validator_i686-pc-linux-gnu	ubuntu	Running
single_job_assimilator_i686-apple-darwin	ubuntu	Running
sample_trivial_validator_i686-apple-darwin	ubuntu	Running
single_job_assimilator_x86_64-apple-darwin	ubuntu	Running
sample_trivial_validator_x86_64-apple-darwin	ubuntu	Running

Computing status

Work	#	Users	#
Tasks ready to send	0	with recent credit	0
Tasks in progress	0	with credit	0
Workunits waiting for validation	0	registered in past 24 hours	0
Workunits waiting for assimilation	0	Computers	#
Workunits waiting for file deletion	0	with recent credit	0
Tasks waiting for file deletion	0	with credit	0
Transitioner backlog (hours)	0	registered in past 24 hours	0
		current GigaFLOPs	0

Tasks by application			
application	unsent	in progress	avg runtime of last 100 r
Jobs for x86_64-pc-linux-gnu	0	0	0.00 (0.00 - 0.00)
Jobs for windows_intelx86	0	0	0.00 (0.00 - 0.00)
Jobs for windows_x86_64	0	0	0.00 (0.00 - 0.00)
Jobs for i686-pc-linux-gnu	0	0	0.00 (0.00 - 0.00)
Jobs for i686-apple-darwin	0	0	0.00 (0.00 - 0.00)
Jobs for x86_64-apple-darwin	0	0	0.00 (0.00 - 0.00)

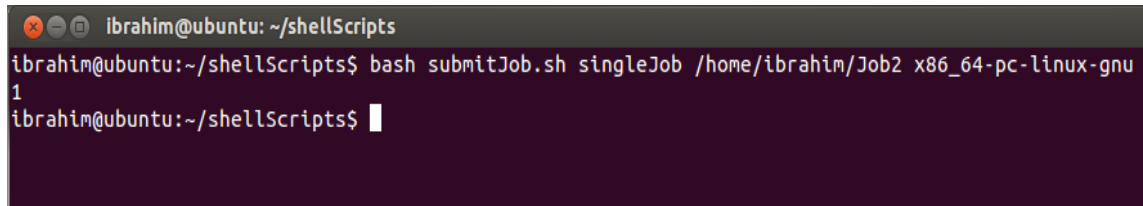
Figure 6.39: Project status executing singleJob.sh.

11. Testing the script **submitJob.sh**

The execution command:

```
bash submitJob.sh singleJob home/ibrahim/job2 x86_64-pc-linux-gnu
```

Figure 6.40 show sthe output of the script execution. It return the wuid(work unit id) assigned to the submitted job which is 1 in this case as shown in the figure below.

A terminal window with a dark purple background. The title bar shows 'ibrahim@ubuntu: ~/shellScripts'. The prompt is 'ibrahim@ubuntu:~/shellScripts\$'. The command entered is 'bash submitJob.sh singleJob /home/ibrahim/Job2 x86_64-pc-linux-gnu'. The output is '1'. The prompt returns to 'ibrahim@ubuntu:~/shellScripts\$' with a cursor.

```
ibrahim@ubuntu:~/shellScripts$ bash submitJob.sh singleJob /home/ibrahim/Job2 x86_64-pc-linux-gnu
1
ibrahim@ubuntu:~/shellScripts$
```

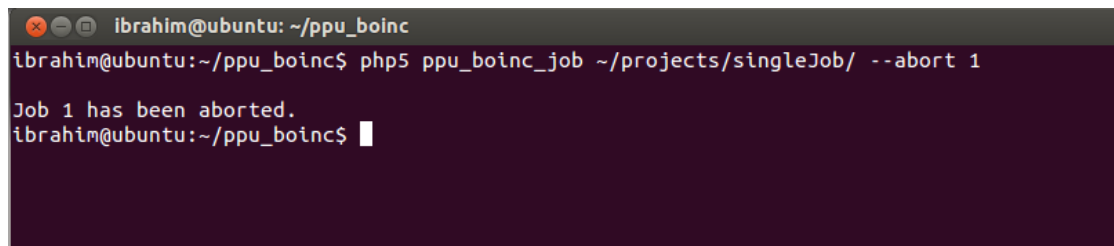
Figure 6.40: Submit job.

12. Testing the php script **abort**

The execution command:

```
php5 ppu_boinc_job ~/projects/singleJob/ --abort 1
```

Figure 6.41 shows the output of the script execution.

A terminal window with a dark purple background. The title bar shows 'ibrahim@ubuntu: ~/ppu_boinc'. The prompt is 'ibrahim@ubuntu:~/ppu_boinc\$'. The command entered is 'php5 ppu_boinc_job ~/projects/singleJob/ --abort 1'. The output is 'Job 1 has been aborted.'. The prompt returns to 'ibrahim@ubuntu:~/ppu_boinc\$' with a cursor.

```
ibrahim@ubuntu:~/ppu_boinc$ php5 ppu_boinc_job ~/projects/singleJob/ --abort 1
Job 1 has been aborted.
ibrahim@ubuntu:~/ppu_boinc$
```

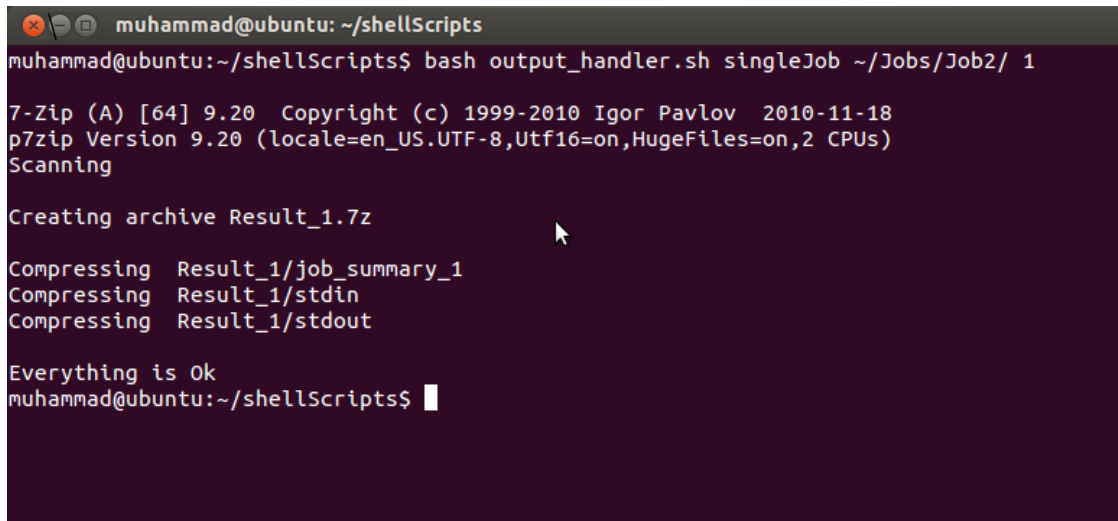
Figure 6.41: Abort job.

13. Testing the script **output_handler.sh**

The execution command:

```
bash submitJob.sh ExampleProject home/ibrahim/job2 1
```

Figure 6.42 shows the output of the script execution.



```
muhammad@ubuntu: ~/shellScripts
muhammad@ubuntu:~/shellScripts$ bash output_handler.sh singleJob ~/Jobs/Job2/ 1
7-Zip (A) [64] 9.20 Copyright (c) 1999-2010 Igor Pavlov 2010-11-18
p7zip Version 9.20 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,2 CPUs)
Scanning

Creating archive Result_1.7z

Compressing  Result_1/job_summary_1
Compressing  Result_1/stdin
Compressing  Result_1/stdout

Everything is Ok
muhammad@ubuntu:~/shellScripts$
```

Figure 6.42: Output handler.

14. Testing the script **enable_account_creation.sh**

The execution command:

```
bash enable_account_creation.sh ExampleProject
```

Figure 6.43 shows the output of the script execution.

```
ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash enable_account_creation.sh ExampleProject
ibrahim@ubuntu:~/shellScripts$
```

Figure 6.43: Enable account creation.

The project home page will show the option of creating account as shown in the figure below.

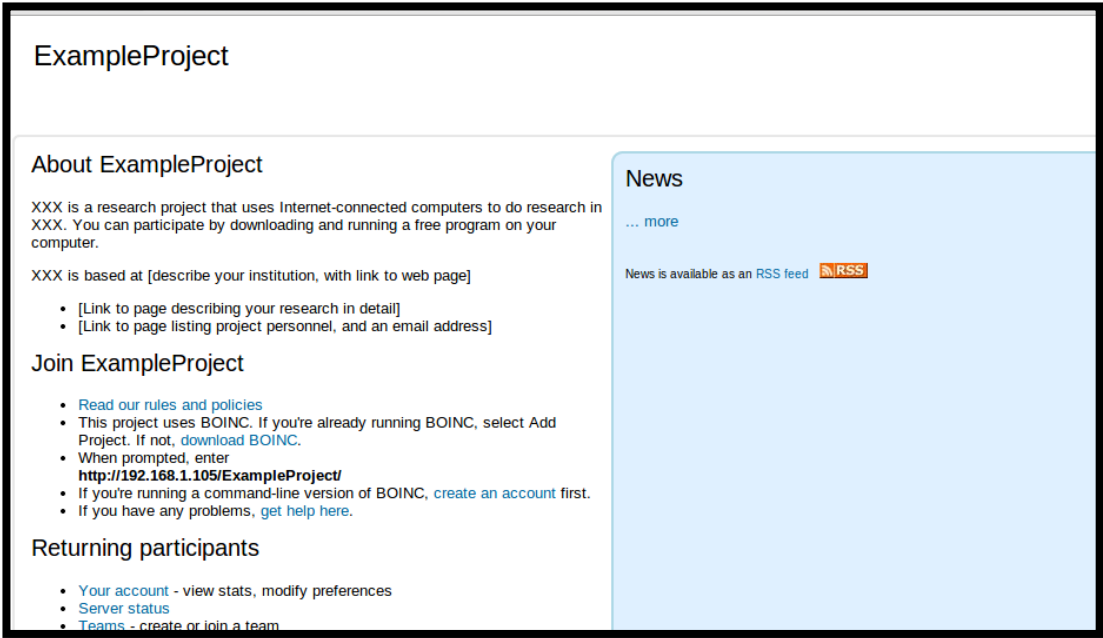


Figure 6.44: Project home page after executing enable_account_creation.sh.

15. Testing the script **disable_account_creation.sh**

The execution command:

```
bash disable_account_creation.sh ExampleProject
```

Figure 6.45 shows the output of the script execution.

```
ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash disable_account_creation.sh ExampleProject
ibrahim@ubuntu:~/shellScripts$
```

Figure 6.45: Disable account creation.

The project home page will remove the option of creating account as shown in the figure below.

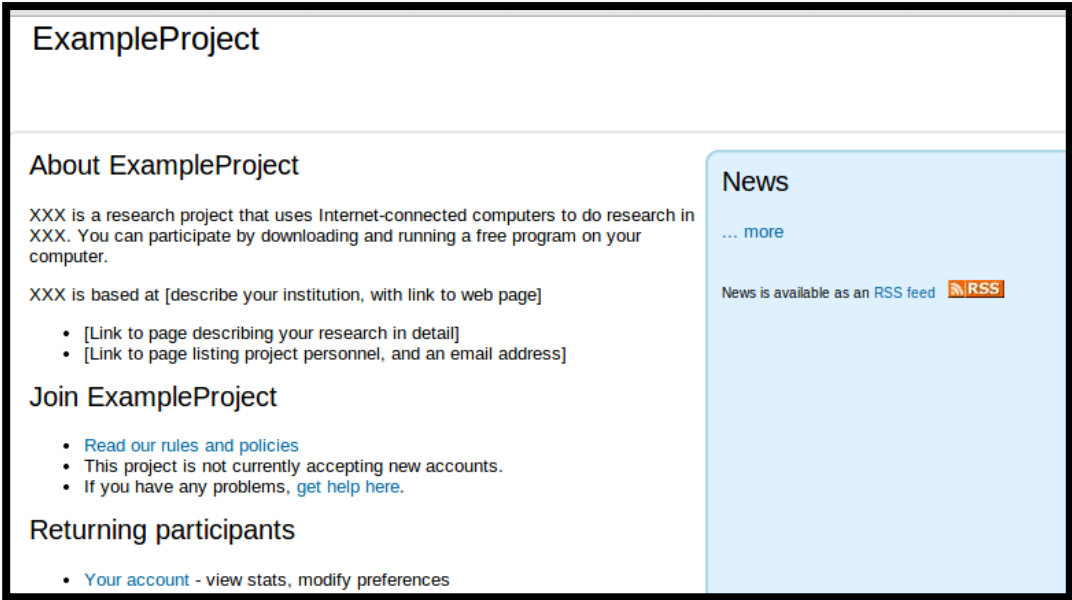


Figure 6.46: Project home page after executing disable_account_creation.sh.

16. Testing the script **projectStatus.sh**

The execution command:

```
bash projectStatus.sh ExampleProject
```

Figure 6.47 shows the output of the script execution.

```

ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash projectStatus.sh ExampleProject
BOINC is ENABLED

DAEMON  pid  status  lockfile  disabled  commandline
  1    8293  running  locked    no        feeder -d 3
  2    8296  running  locked    no        transitioner -d 3
  3    8299  running  locked    no        file_deleter -d 3
  4    8302  running  locked    no        sample_work_generator -d 3
  5    8305  running  locked    no        sample_bitwise_validator -d 3 --app
example_app
  6    8308  running  locked    no        sample_assimilator -d 3 --app exampl
e_app

TASK      last run      period      next run      lock file  disabled  co
commandline
  1      ?            24 hours    NOW           unlocked  no        ar
tigue_file_deleter -d 2
  2      ?            24 hours    NOW           unlocked  no        db
_dump -d 2 --dump_spec ../db_dump_spec.xml
  3      ?            1 days     NOW           unlocked  no        ru
n_in_ops ./update_uotd.php
  4  2014/05/20 16:50:01  1 hour     2014/05/20 17:50:01  unlocked  no        ru

```

Figure 6.47: Project status.

17. Testing the script **deleteProject.sh**

The execution command:

```
bash deleteProject.sh ExampleProject
```

This script deletes the project directory as well as the project DB. Figure 6.48 shows the script execution.

```

ibrahim@ubuntu: ~/shellScripts
ibrahim@ubuntu:~/shellScripts$ bash deleteProject.sh NewProject
ibrahim@ubuntu:~/shellScripts$ █

```

Figure 6.48: Project deletion.

6.5.2 Testing Portal functionalities (Testing higher level)

Through this section we test all the functionalities provided by the system portal. These tests discussed below.

1. Project creation

Admin can create BOINC projects remotely using the system portal. The created project is ready for adding computing resources and users to submit jobs. Through the project creation many scripts are executed; **createTestProject.sh** for Test example project, **createProject.sh** for empty and single job project, **singleJob.sh** for single job project, **createAccount.sh** and **disable_account_creation.sh**. Each of these scripts is described previously in the implementation section.

Creating project is done through the project creation form. Admin enters the project name and project type and click on the **create** button. The system must validate the name of the project before creating it, and displays error messages for invalid names. The test of this functionality was done by entering an invalid name (**TestSystem?**). The result is shown in the figure below.

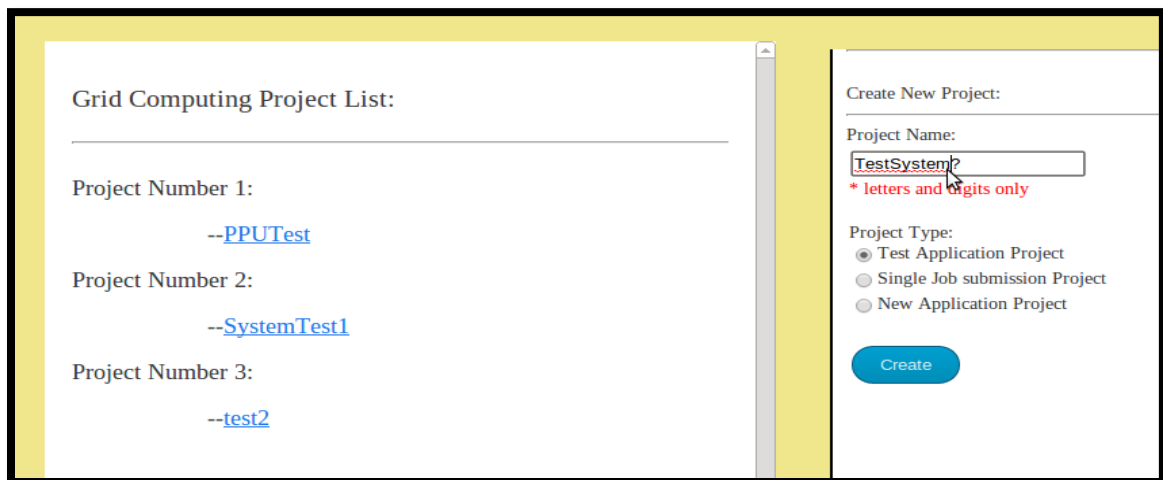


Figure 6.49: Project creation test 1.

The second test was done by entering the name “**TestSystem**”. Figure 6.50 shows the portal interface during the project creation process. A successful project creation is shown in figure 6.51.

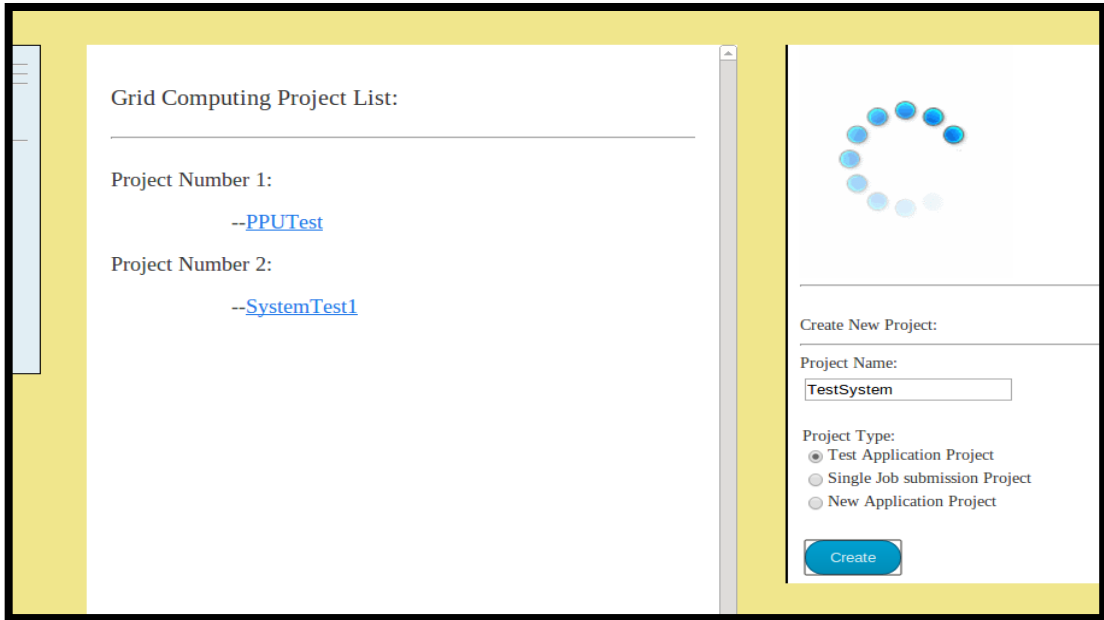


Figure 6.50: Project creation test 2.

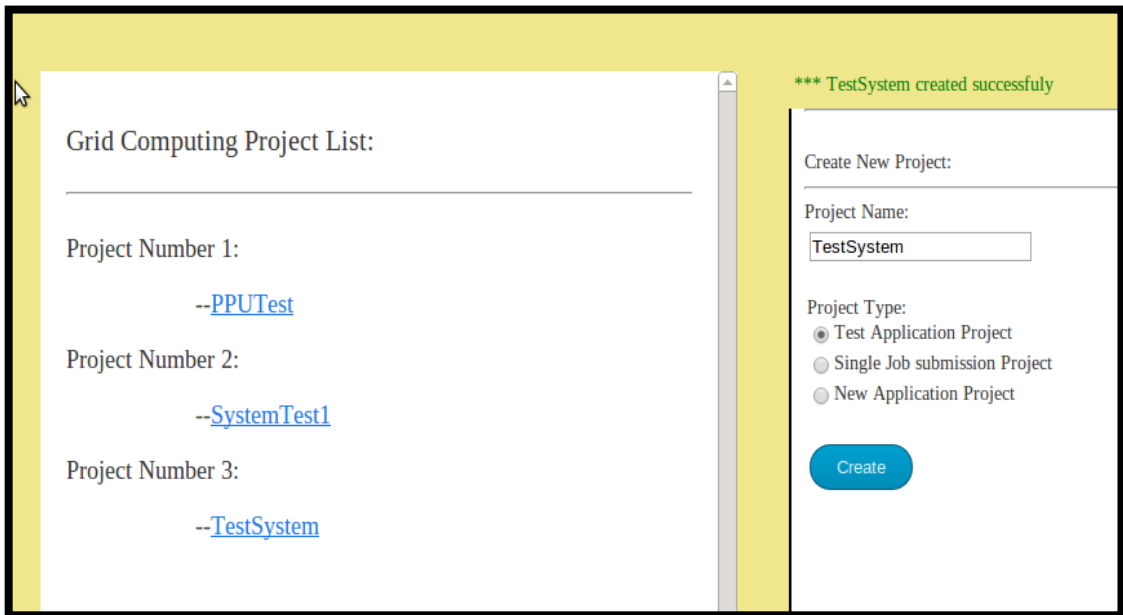


Figure 6.51: Successful Project Creation.

The same tests were applied to all project types and all of them followed the same scenario.

2. Adding new user

The first test is applied by entering invalid inputs; the result is shown in figure 6.52.

The screenshot displays a web application interface for user management. On the left, a section titled "All Authenticated System Users:" contains a table with the following data:

Name	Email	Project	select
boincadm	boincadm@yahoo.com		<input type="checkbox"/>
Muhammad	mohammad.thwaib@yahoo.com	SystemTest1	<input type="checkbox"/>
Ibrahim	qusaymusa@yahoo.com	SystemTest1	<input type="checkbox"/>
Mohammed	mohammed@ppu.edu		<input type="checkbox"/>
superuser	superuser@ppu.edu		<input type="checkbox"/>
saleem	saleem@ppu.edu	SystemTest1	<input type="checkbox"/>

Below the table is a "Delete users" button. On the right, the "Add a New user:" form contains the following fields and error messages:

- Name: with error message: ** letters and digits only*
- Email: with error message: ** invalid email address*
- Password: with error message: ** must at least 5 digits*
- ProjectID:
- Level:
- Submit button: "Add user"

Figure 6.52: Error handling test for adding new user.

The second is test applied by entering valid inputs; the result is shown in figure 6.53.

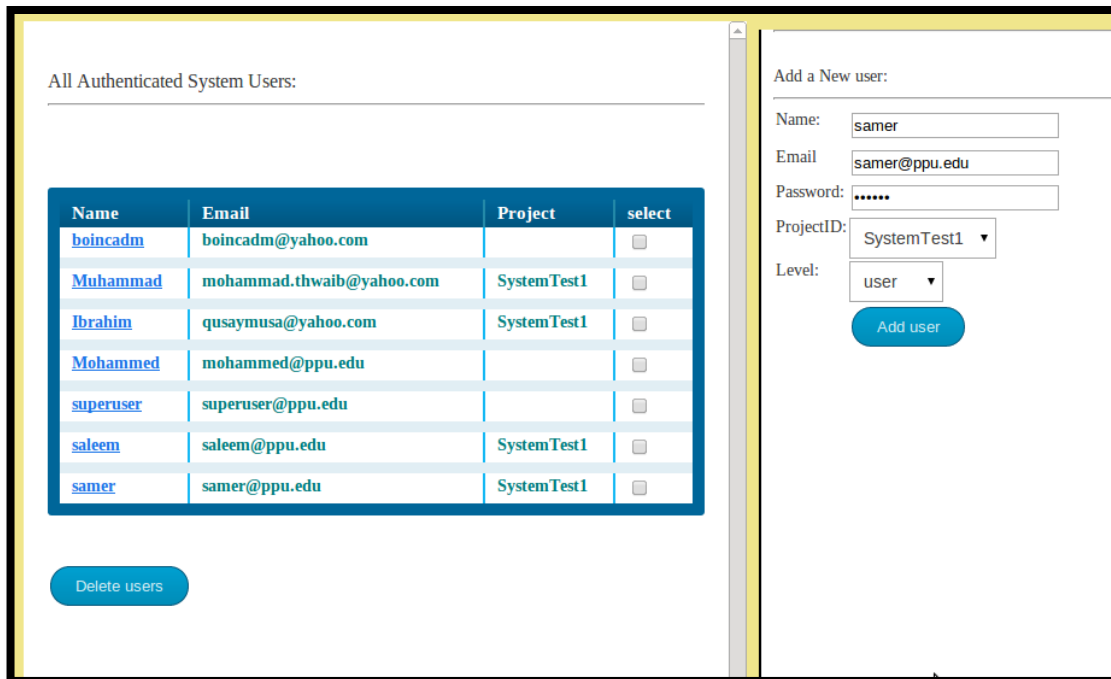


Figure 6.53: Test adding new user.

3. Attaching/ de-attaching client to a project:

Test the error handling was done by entering invalid IP addresses as shown in the figure below.

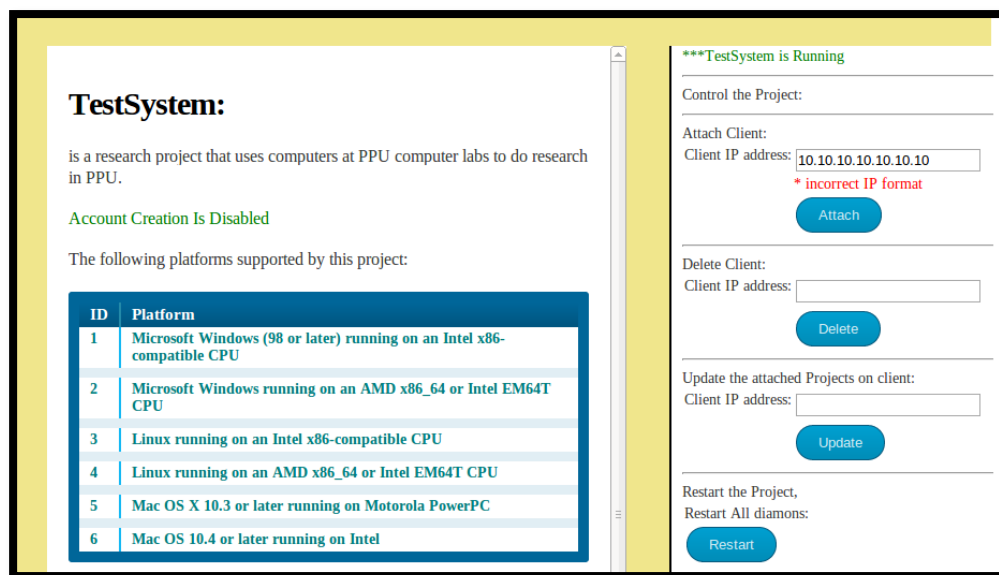


Figure 6.54: Error handling test for attaching client.

These tests applied also on de-attach and update functionalities, the same results retrieved.

Another test was done by entering a valid IP address. Figure 6.55 and Figure 6.56 show the BOINC manager of the target client before and after the attachment.

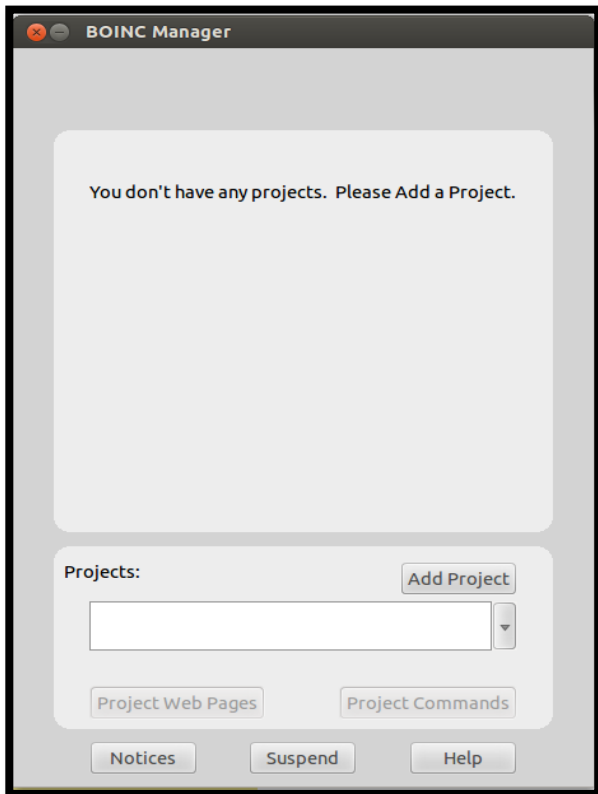


Figure 6.55: before attaching a client.

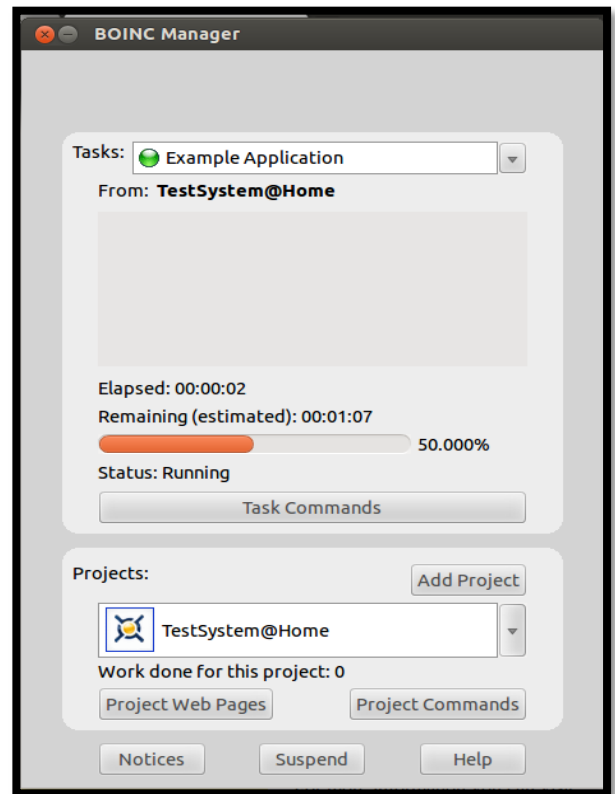


Figure 6.56: After attaching a client.

4. Update client attached projects:

Figure 6.57 show the BOINC manager of the target client. It is appear that the target client is doing contact with the attached project.

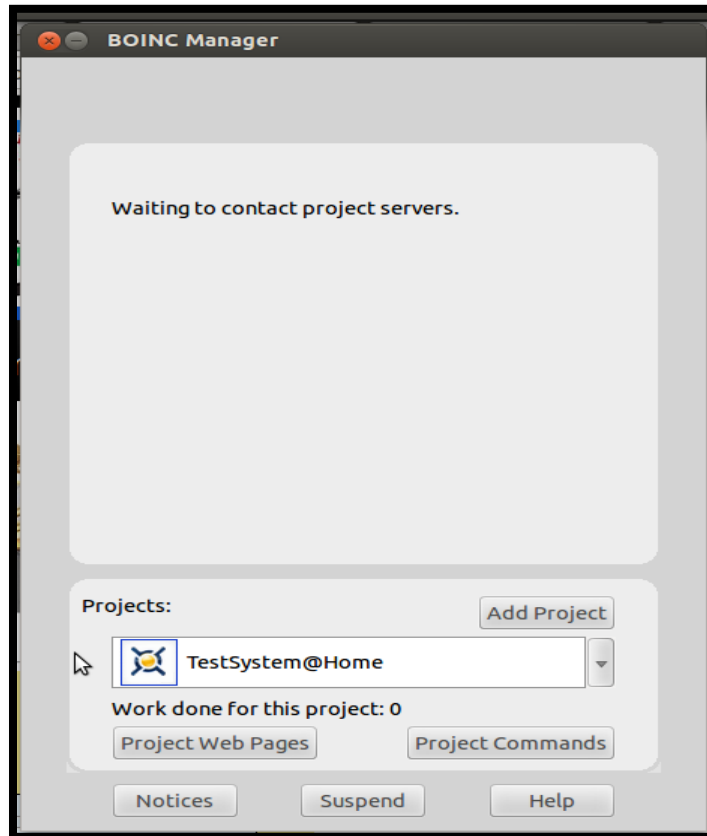


Figure 6.57: BOINC manager of the updated client.

5. Stop, start and restart project:

Figure 6.58 below shows the project status after the admin click on the **stop** project button.

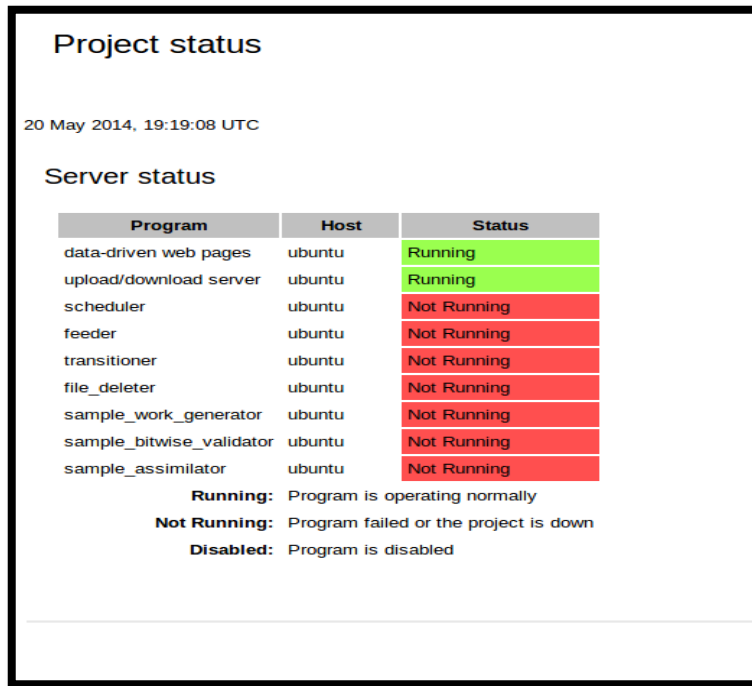


Figure 6.58: Project status page after project stop.

Figure 6.59 shows the project status after the admin click on the either **start** or **restart** project button.

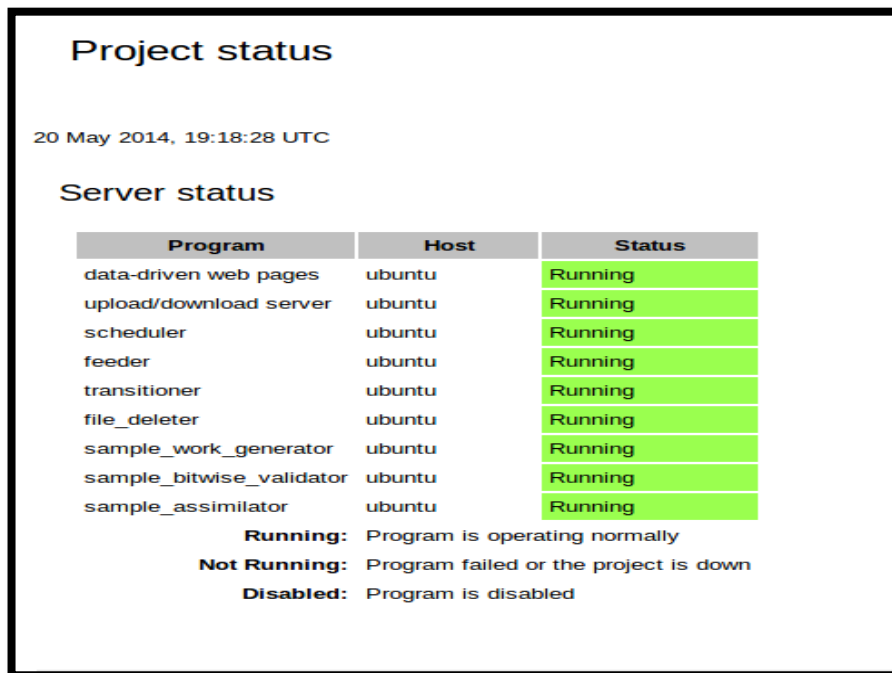


Figure 6.59: Project status page after start/restart project.

6. Delete project:

Project directory and project DB are deleted when admin click on **delete project** button. As a result all the project pages were deleted also. Figure 6.60 shows the project home page after deletion.

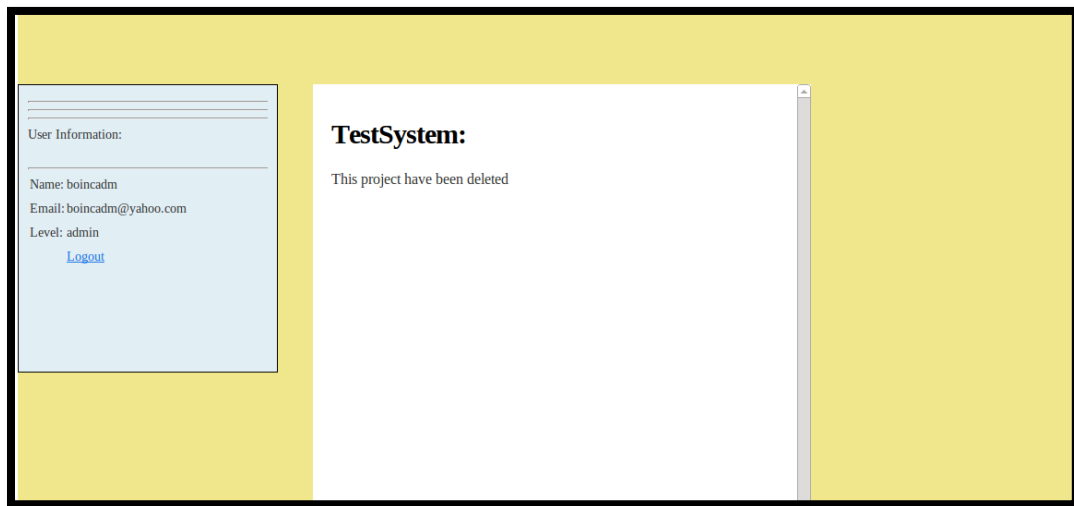


Figure 6.60: Project management page after deleting the project.

7. Modifying Account Information

Any user can modify his account information through account form. First, a test is applied by entering an invalid password and click on modify account. The result is shown in figure 6.61 below.

Account Information:

Name:

Email:

Current Password: * password incorrect

New Password: * new password is required

Confirm Password:

Figure 6.61: Invalid account modification.

Many other tests were applied to all cases of errors; all of them were handled successfully.

Another test of modification with correct inputs was applied, the result is shown in figure 6.62.

information updated successfully

Account Information:

Name:

Email:

Current Password:

New Password:

Confirm Password:

Figure 6.62: Valid account modification.

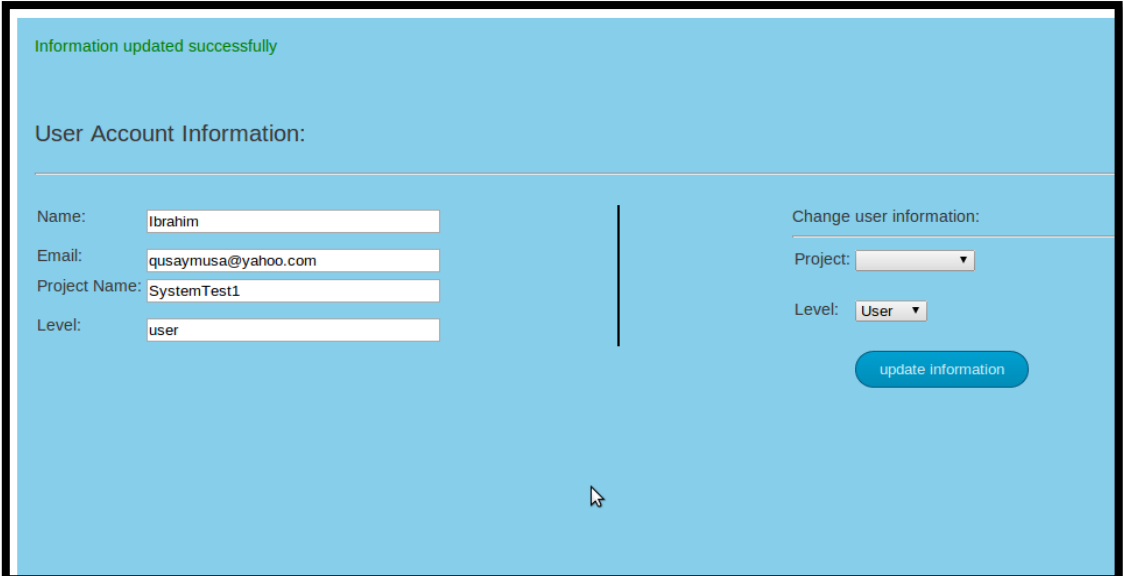
8. Modifying User Information

Admin can modify some information of users through user account form. Figures below show the user information before and after modification.



The screenshot shows a light blue interface with the title "User Account Information:". On the left, there are four input fields: "Name:" with the value "Ibrahim", "Email:" with "qusaymusa@yahoo.com", "Project Name:" which is empty, and "Level:" with "admin". On the right, under "Change user information:", there are two dropdown menus: "Project:" and "Level:" (set to "User"). A blue "update information" button is at the bottom right.

Figure 6.63: User information before modification.

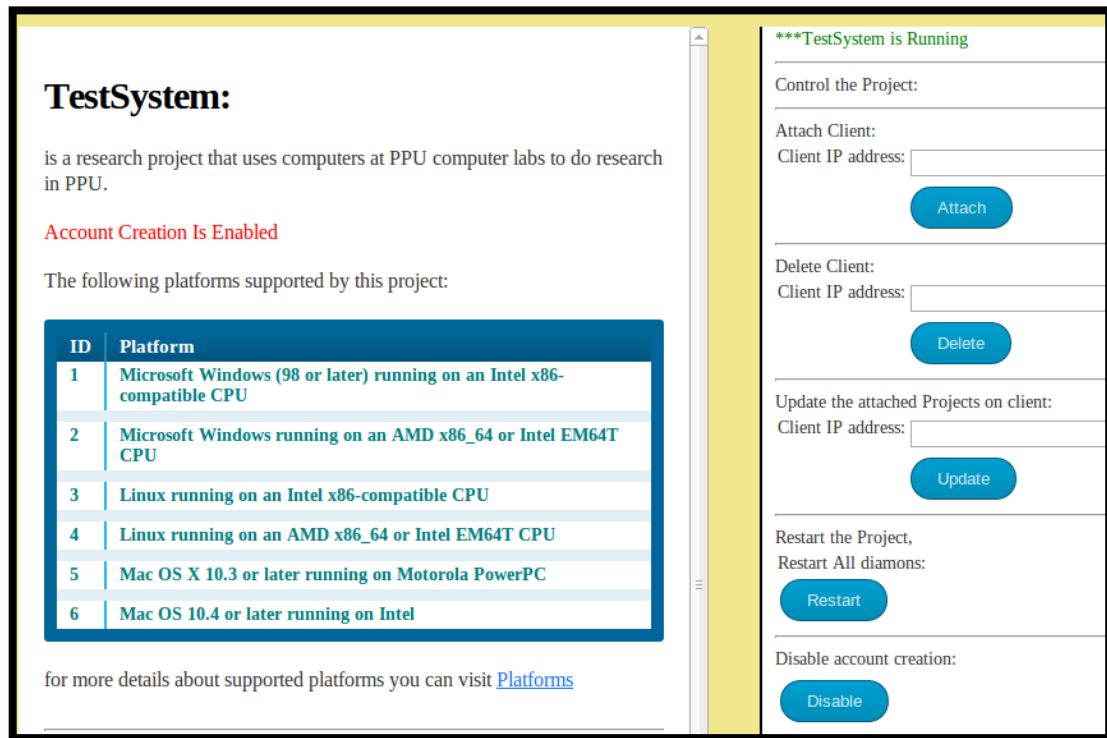


The screenshot shows the same interface as Figure 6.63, but with a green message "Information updated successfully" at the top left. The "Project Name:" field now contains "SystemTest1" and the "Level:" field now contains "user". The "update information" button is still present at the bottom right.

Figure 6.64: User information after modification.

9. Enable/disable account creation

Testing enabling account creation was done by clicking on Enable button exist in the project management page. Result is shown below.



The screenshot displays the 'TestSystem' project management interface. On the left, the 'TestSystem' section includes a description, a red status message 'Account Creation Is Enabled', and a table of supported platforms. On the right, a control panel shows the system is running and provides buttons for 'Attach', 'Delete', 'Update', 'Restart', and 'Disable' account creation, each with an associated IP address input field.

TestSystem:

is a research project that uses computers at PPU computer labs to do research in PPU.

Account Creation Is Enabled

The following platforms supported by this project:

ID	Platform
1	Microsoft Windows (98 or later) running on an Intel x86-compatible CPU
2	Microsoft Windows running on an AMD x86_64 or Intel EM64T CPU
3	Linux running on an Intel x86-compatible CPU
4	Linux running on an AMD x86_64 or Intel EM64T CPU
5	Mac OS X 10.3 or later running on Motorola PowerPC
6	Mac OS 10.4 or later running on Intel

for more details about supported platforms you can visit [Platforms](#)

***TestSystem is Running

Control the Project:

Attach Client:
Client IP address:
Attach

Delete Client:
Client IP address:
Delete

Update the attached Projects on client:
Client IP address:
Update

Restart the Project,
Restart All diamons:
Restart

Disable account creation:
Disable

Figure 6.65: Enabling account creation.

Testing disabling account creation was done by clicking on disable button exist in the project management page. Result is shown in figure 6.66.

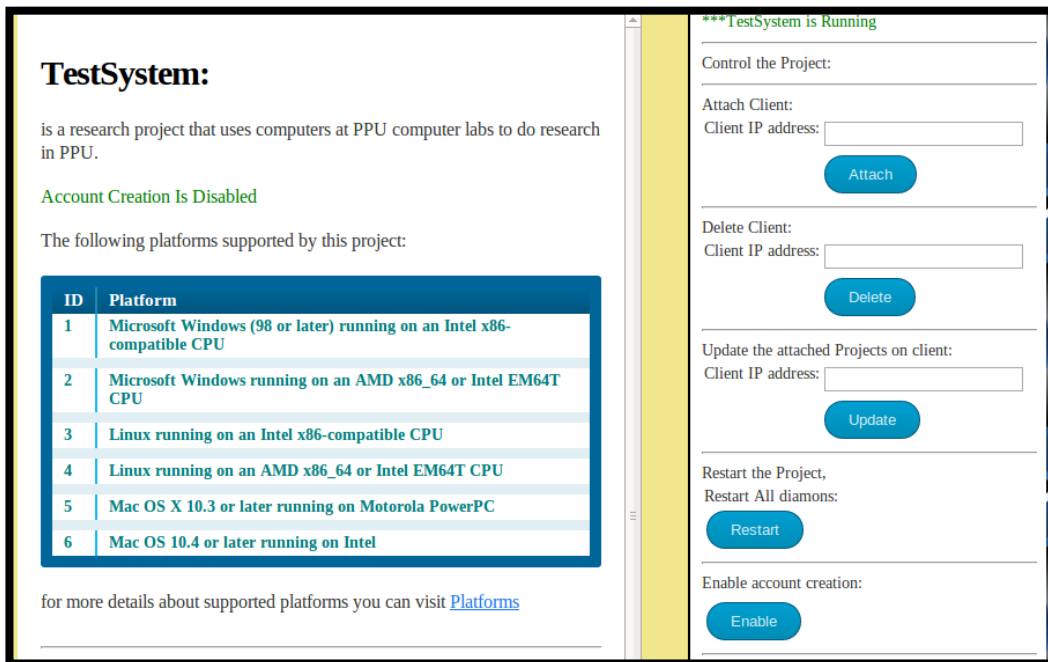


Figure 6.66: Disabling account creation.

10. Delete users

Admin can delete users from Users page by checking the users to be deleted, and then click on “Delete users” button. Figures below show a test applied on this functionality.

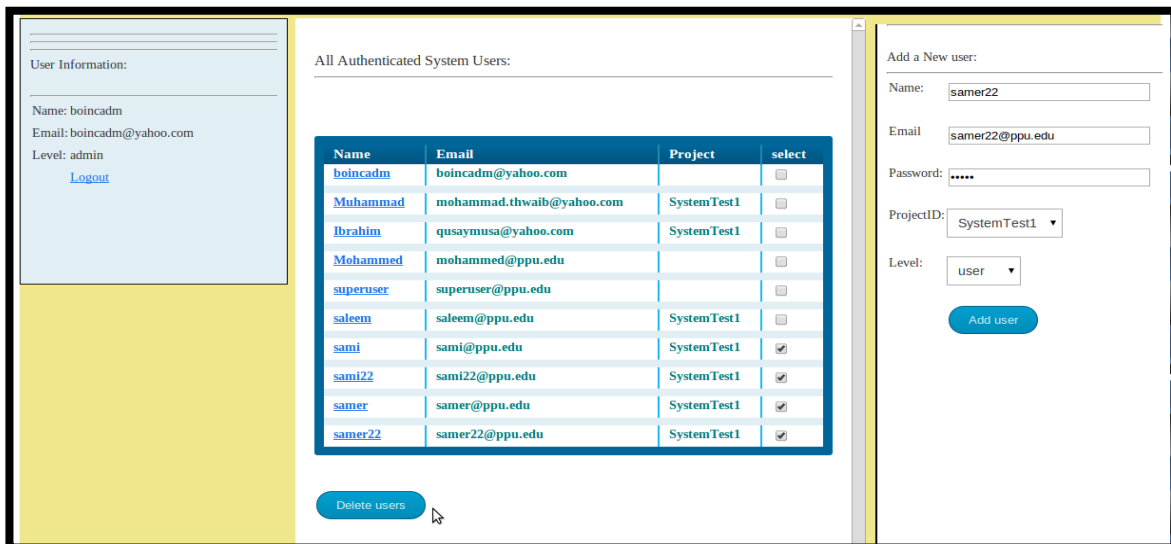


Figure 6.67: Checking users for deletion.

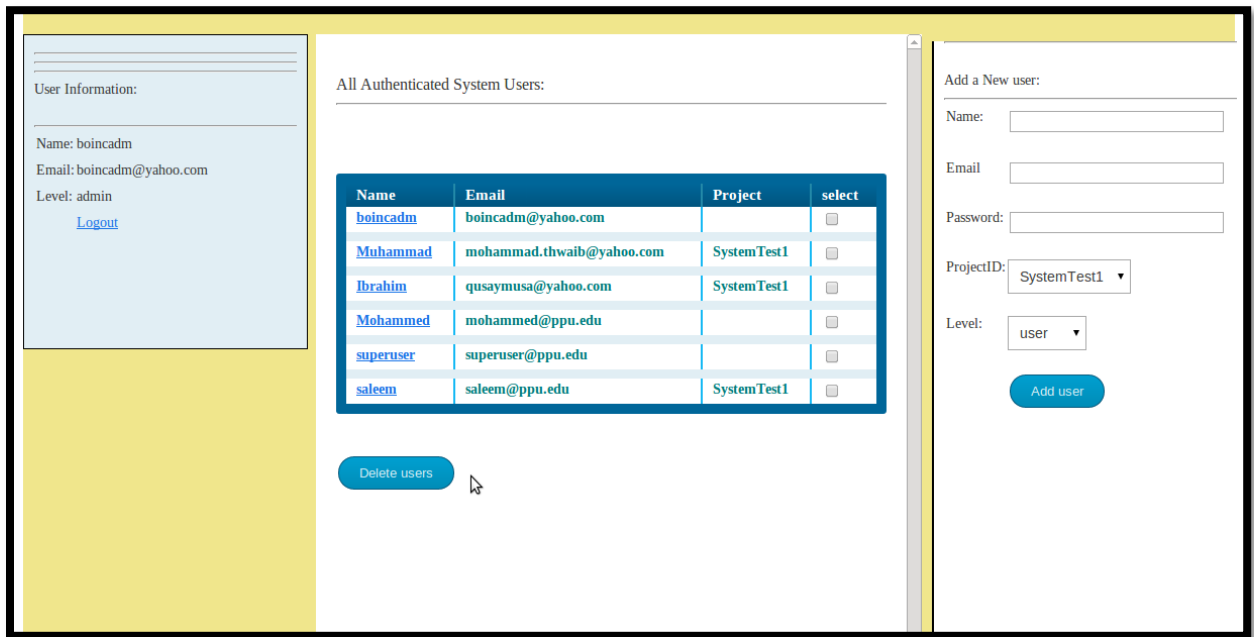


Figure 6.68: Users deletion.

11. Job submission:

Error handling test was applied by choosing an invalid job file and trying to submit the job without selecting a platform, result shown in the figure6.69.

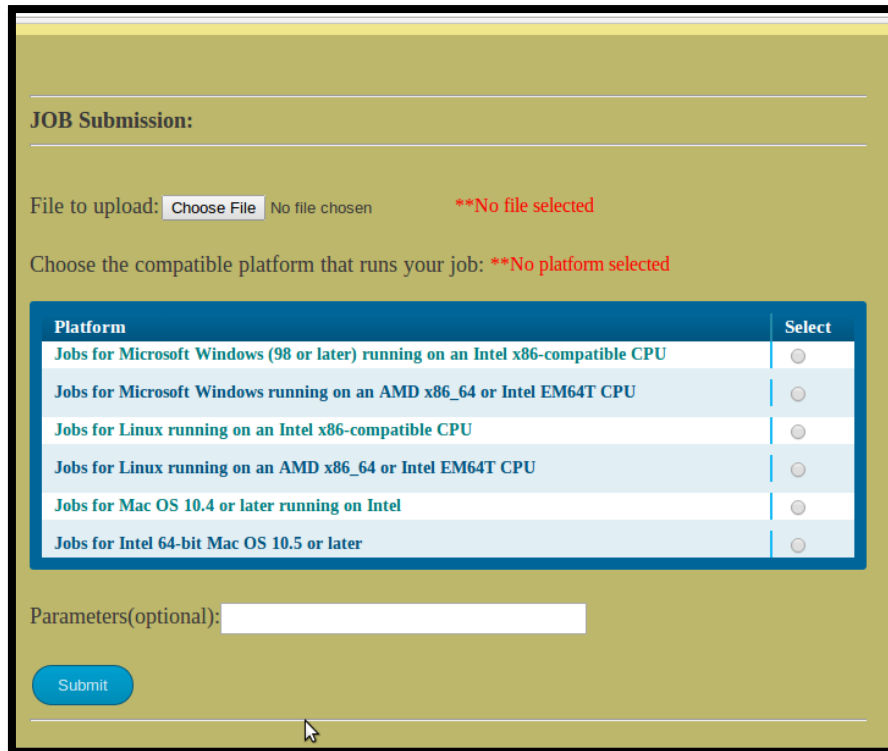


Figure 6.69: Error job submission.

Another test of job submission with valid inputs is shown in figure 6.70.

***Job2 uploaded successfully

JOB Submission:

File to upload: No file chosen

Choose the compatible platform that runs your job:

Platform	Select
Jobs for Microsoft Windows (98 or later) running on an Intel x86-compatible CPU	<input type="radio"/>
Jobs for Microsoft Windows running on an AMD x86_64 or Intel EM64T CPU	<input type="radio"/>
Jobs for Linux running on an Intel x86-compatible CPU	<input type="radio"/>
Jobs for Linux running on an AMD x86_64 or Intel EM64T CPU	<input type="radio"/>
Jobs for Mac OS 10.4 or later running on Intel	<input type="radio"/>
Jobs for Intel 64-bit Mac OS 10.5 or later	<input type="radio"/>

Parameters(optional):

Figure 6.70: Valid job submission.

12. Aborting job execution

When the user clicks on abort button, system displays a confirmation message as shown in figure 6.69. If the user click on “**Ok**” job will be aborted. Figure 6.72 shows the process while aborting a job.

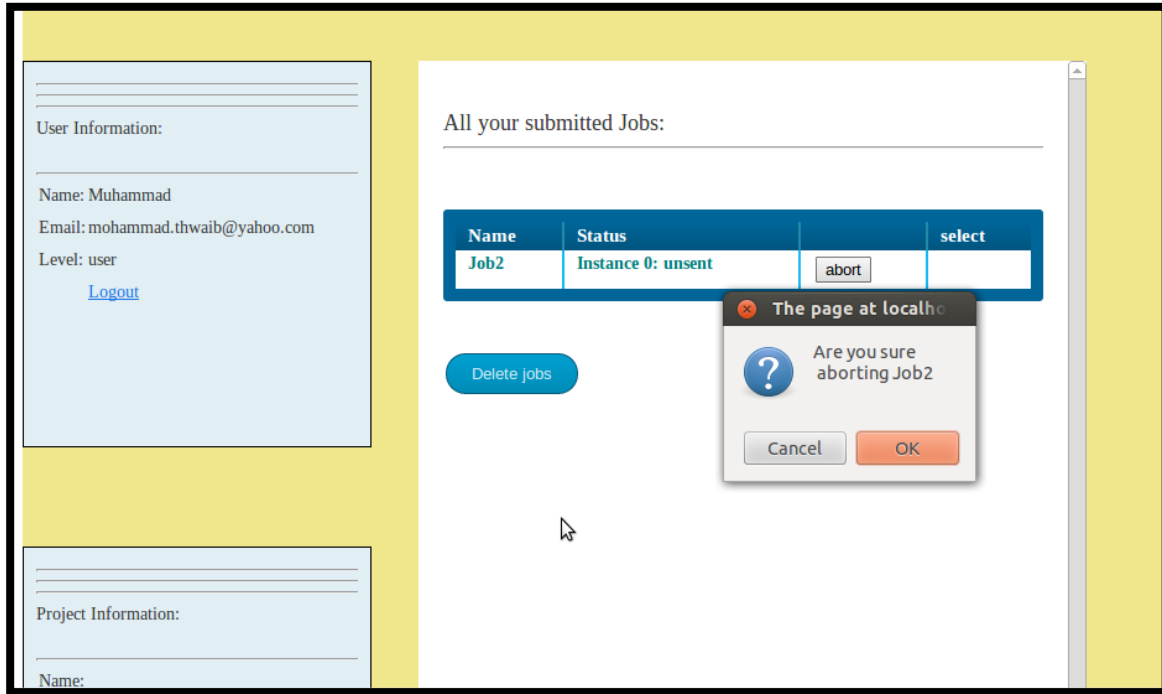


Figure 6.71: Aborting Job confirmation.

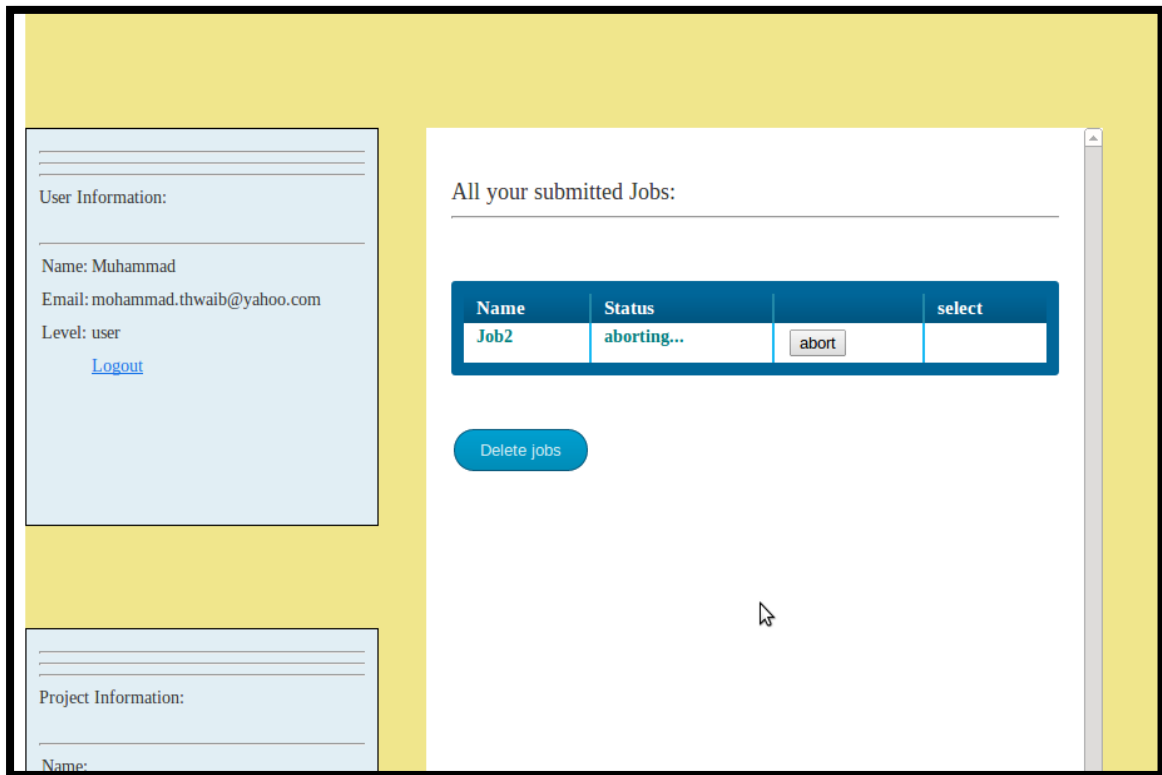


Figure 6.72: Aborting job.

13. Job execution:

Test applied by submitting a job and monitoring its status during the execution. As shown in the figures below that the job was moved successfully through the execution states arriving to the completion state.

User Information:

Name: Muhammad
Email: mohammad.thwaib@yahoo.com
Level: user
[Logout](#)

All your submitted Jobs:

Name	Status		select
Job1	(no instances yet)	Download	<input type="checkbox"/>
Job2	Instance 0: unsent	abort	

Delete jobs

Figure 6.73: Job execution (state 1).

User Information:

Name: Muhammad
Email: mohammad.thwaib@yahoo.com
Level: user
[Logout](#)

All your submitted Jobs:

Name	Status		select
Job1	(no instances yet)	Download	<input type="checkbox"/>
Job2	Instance 0: in progress on host 1	abort	

Delete jobs

Figure 6.74: Job execution (state 2).

User Information:

Name: Muhammad
 Email: mohammad.thwaib@yahoo.com
 Level: user
[Logout](#)

All your submitted Jobs:

Name	Status		select
Job11	(no instances yet)	Download	<input type="checkbox"/>
Job2	Job is being assimilated		

[Delete jobs](#)

Figure 6.75: Job execution (state 3).

User Information:

Name: Muhammad
 Email: mohammad.thwaib@yahoo.com
 Level: user
[Logout](#)

All your submitted Jobs:

Name	Status		select
Job11	(no instances yet)	Download	<input type="checkbox"/>
Job2	Instance 0: completed on host 1	Download	<input type="checkbox"/>

[Delete jobs](#)

Figure 6.76: Job execution (state 4).

14. Download Results

A user can download results of an executed job by clicking on the “**Download**” link. A test applied on one of the completed jobs; download window was lunched as shown in the figure below, and finally result was downloaded successfully.

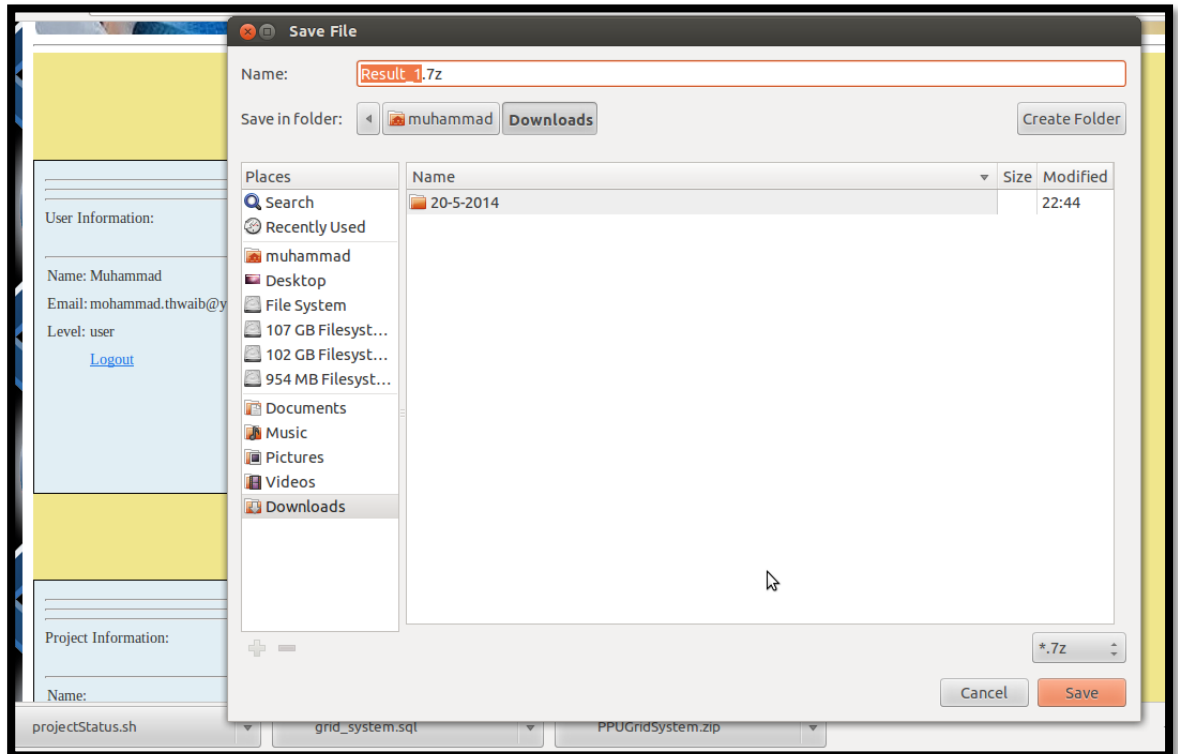


Figure 6.77: Download window.

15. Delete results

A user can delete the results by checking them, and then click on “**Delete jobs**” button. Figures below show a test applied on this functionality.

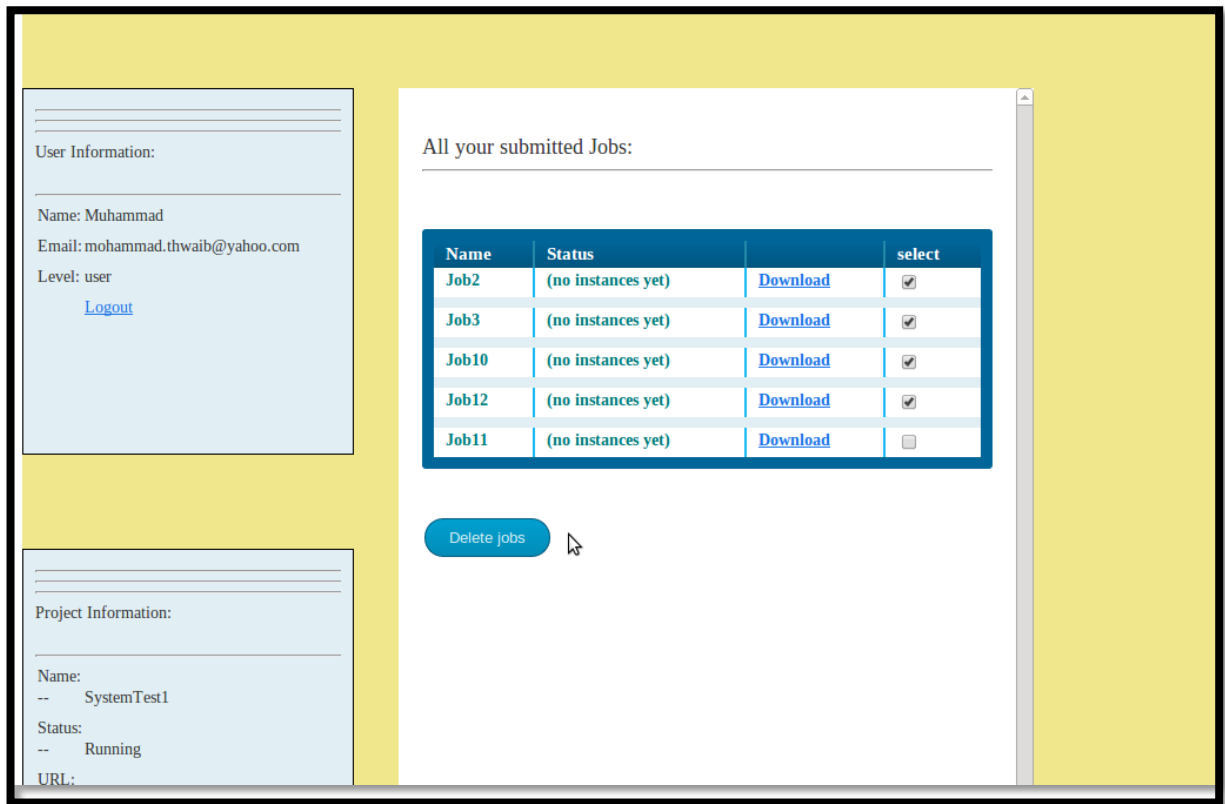


Figure 6.78: Check results to delete.

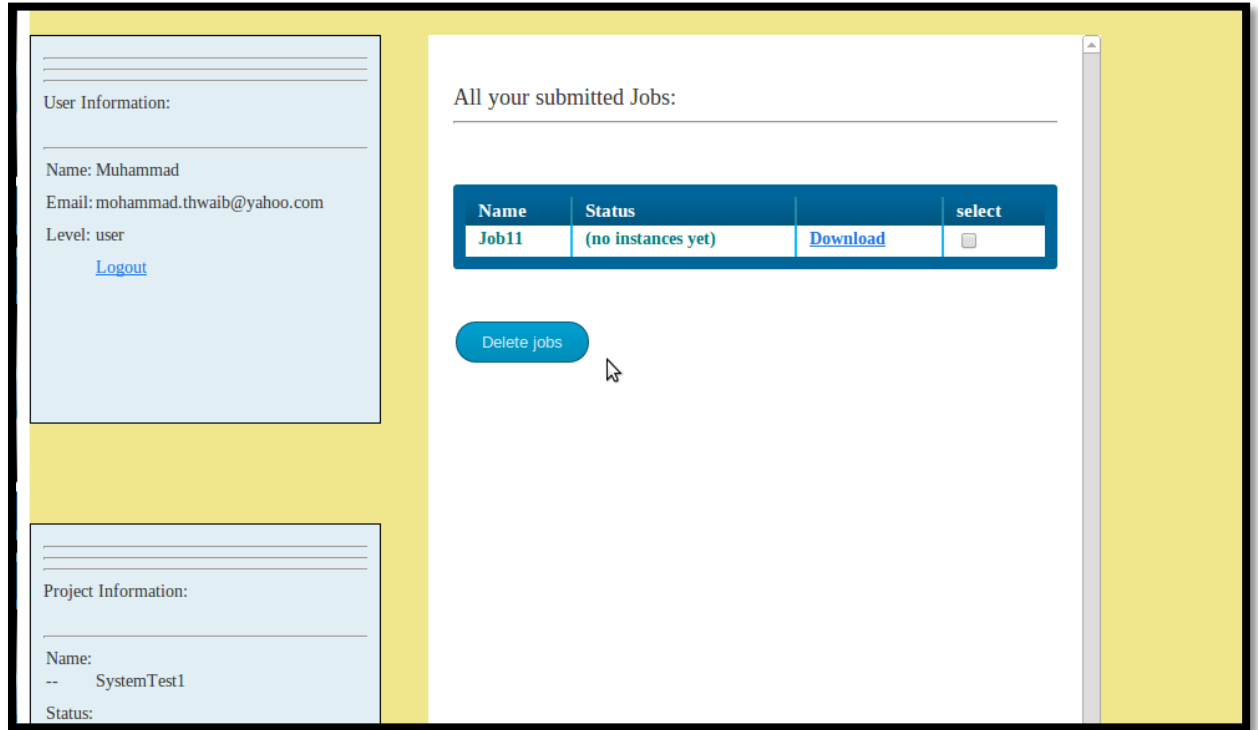


Figure 6.79: Home page after click Delete Jobs.

6.6 Summary

This chapter clarifies the system implementation at the lower level which consists of set of shell scripts that perform the system main functionalities. Also, it describes the implementation of the higher level of the system. The higher level of the system includes performing the system functionalities remotely and through user friendly interfaces. Finally, we stated the procedures followed in testing the system.

Chapter Seven

Experiments and Results

7.1 Overview

In this chapter we talk about the CPU utilization at PPU computer labs. Through this experiment we use CPU Usage Logger and Altra CPU monitor – freeware programs – in order to arrive to approximation to the average CPU usage at PPU computer labs. Also we examine how many floating point operations per second can be obtained from the PPU environment.

7.2 Average CPU usage at PPU computer labs

This section has two main subsections; the first one clarifies the environment, describes the programs used to get the average CPU usage and the difficulties that we face during this stage. The second one states the practical work to get the average CPU usage in PPU labs.

7.2.1 Environment specification and work difficulties

PPU contains approximately 1000 PC computers distributed over computer labs. These computers are interconnected with local area network (LAN) of 100Mbps speed, which is suitable for grid clients. A continuously upgrading and maintaining are performed to these computers; so they almost have a computational power much far exceeds their usage by students for learning and internet access purposes. During this project we monitored the CPU usage for a selected sample of PCs in some of labs. These labs are: Al-Beruni(I), Al-Razi, Al-khwarizmi, PC1, and the security lab. The following table shows the specifications of the PCs in these labs.

Table 7.1: Computers specifications.

Lab	Building	CPU specification
Beruni(I)	B,2 nd floor	Core 2 duo /2.66GHz
Beruni(II)	B,2 nd floor	Core 2 duo/2.66GHz
Al-Razi	B,2 nd floor	Core 2 duo /2.66GHz
Al-khwarizmi	B,2 nd floor	Core i5 /3.2GHz
PC1	B+,1 nd floor	Dual core /2GHz
Security	C,1 nd floor	Core i5 /3.2GHz

Monitoring CPU usage done during working hours (from 08:00am to 04:00pm).the study was applied approximately for 20 days distributed over semester. Two main utility programs were used to log the CPU usage during the study. These programs are ultra CPU usage monitor and CPU usage logger. The result from this study generalized to represent the average CPU usage for all labs at the university.

CPU Usage Logger:

CPU Usage Logger is used to log the CPU usage to a text file. It is recommended by many researchers and developers to perform this work; “since it is small, simple and reliable freeware utility that offers a handy way for developers and software testers to easily monitor and log CPU usage for any period of time”[18].

This program has many advantages to perform this work. Some of them: it logs the CPU usage to a log file so that we can retrieve these log files at any time even if the computer got unexpected shut down during working hours. In addition, it doesn't produce extra load to the CPU or memory usage during its running since it is lightweight program. It doesn't need an installation sequence to be running on the system. Just copy it's folders to any place on the free space of disk and double click on the executable file, it will start running directly. On the other hand it has a critical disadvantage; which is the need for user interaction to direct its output to a specific

log file; this is needed each time the program is running. So it can't be added to the start-up list.

CPU usage logger will display the interface shown in figure 6.1 when it is run; we can direct its output to a specific log file using that interface. It will add entry to the file each 5sec. This entry will contain the columns headers as follow:

- Date of reading
- Time of reading
- Percentage of CPU usage between (90-100) percent during the last 5sec.
- Percentage of CPU usage between (80-90) percent during the last 5sec.
- Percentage of CPU usage between (70-80) percent during the last 5sec.
- Percentage of CPU usage between (60-70) percent during the last 5sec.
- Percentage of CPU usage between (50-60) percent during the last 5sec.
- Percentage of CPU usage between (40-50) percent during the last 5sec.
- Percentage of CPU usage between (30-40) percent during the last 5sec.
- Percentage of CPU usage between (20-30) percent during the last 5sec.
- Percentage of CPU usage between (10-20) percent during the last 5sec.
- Percentage of CPU usage between (0-10) percent during the last 5sec.

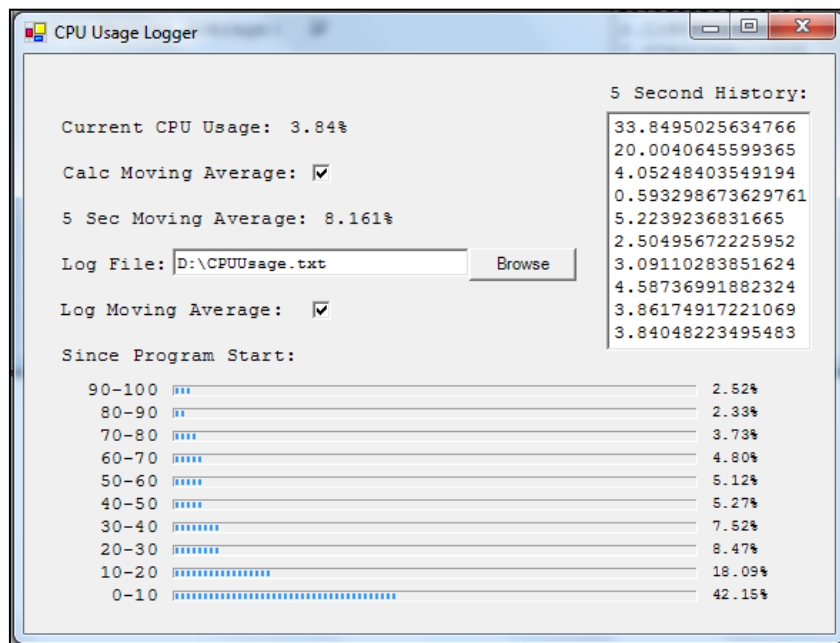


Figure 7.1: CPU usage logger program.

Ultra CPU monitor:

Ultra CPU monitor is a professional application designed to be a small CPU monitoring tool that shows its activity as a diagram in the system tray. This free software tool can display many icons. Each monitor can display its activity as a text or as a diagram. Ultra CPU Monitor works on Windows operating systems [43].

Ultra CPU monitor has important advantages; it can be added to the start-up list, so it runs automatically when the system start. It also averages the CPU usage while the running period. In addition it doesn't produce extra loads to the CPU while it's running. It doesn't need an installation sequence to be running on the system. Just copy it's folders to any place on the free space of disk and double click on the executable file, it will start running directly. On the other hand, this program doesn't produce a log file, so if the computers got unexpected shut down during the running period we can't retrieve any previous results.

Ultra CPU monitor will add 3 basic icons to the system toolbar as shown in figure 6.2; we can get the average CPU usage using those icons.



Figure 7.2: Ultra CPU monitor program.

7.2.2 Practical Work

During this subsection we will perform the computations needed to arrive to a general term that represent the overall average CPU usage at PPU computer labs. This work will basically depend on the data collected by the freeware programs that stated in the previous section. The work goes on 3 levels that are:

- Collect the daily results about the average CPU usage and perform the needed computation to get the average usage in that day.
- Averaging all the data for each lab that represents the average CPU usage in that lab.
- The overall average CPU usage at PPU labs will be the average of all of CPU usage for all labs during the study period

Table 7.2 shows the average CPU usage on the computers at AL-Beruni(I) lab for one day. Figure 7.3 show a part of the log file generated by CPU usage logger which used in the computations to get the results shown in table 7.2.

Table 7.2: Average CPU usage at AL-Beruni(I) lab for one day.

PC name	Average CPU usage(using Ultra CPU monitor program)	CPU usage less than 10% (CPU usage logger program)
PC1	7%	92%
PC2	15%	86%
PC3	9%	91%
PC4	11%	88%
PC5	20%	83%
PC6	3%	96%

Date	Time	(90-100)%	(80-90)%	(70-80)%	(60-70)%	(50-60)%	(40-50)%	(30-40)%	(20-30)%	(10-20)%	(0-10)%
02/13/2014	12:06:08	0	0	0	0	1	1	1	2	4	91
02/13/2014	12:06:13	0	0	0	0	1	1	1	2	4	92
02/13/2014	12:06:18	0	0	0	0	1	1	1	1	4	91
02/13/2014	12:06:23	0	0	0	0	1	1	1	1	4	91
02/13/2014	12:06:28	0	0	0	0	1	1	1	2	4	91
02/13/2014	12:06:33	0	0	0	0	1	1	1	2	4	91
02/13/2014	12:06:38	0	0	0	0	1	1	1	2	4	91
02/13/2014	12:06:43	0	0	0	0	1	1	1	2	4	91
02/13/2014	12:06:48	0	0	0	0	1	1	1	2	4	91
02/13/2014	12:06:53	0	0	0	0	1	1	1	1	4	91
02/13/2014	12:06:58	0	0	0	0	1	1	1	1	4	91
02/13/2014	12:07:03	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:08	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:13	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:18	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:23	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:28	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:33	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:38	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:43	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:48	0	0	0	0	1	1	1	1	4	92
02/13/2014	12:07:53	0	0	0	0	1	1	1	1	4	92

Figure 7.3: Sample of the log file generated by CPU usage logger.

The following table summarizes the average CPU usage in each lab according to both of programs and the number of computers that were monitored in each lab. Where U_CPU_M is the ultra CPU monitor program and CPU_U_L is CPU usage logger program.

Table 7.3 Average CPU usage in PPU computer labs.

Lab Name	Sample size	Avg. CPU usage(U_CPU_M)	Avg. CPU usage(CPU_U_L)
Al-Beruni(I)	6	12%	Less than 10% for 83% of the time.
Al-Razi	6	13%	Less than 10% for 82% of the time
Al-khwarizmi	10	8%	Less than 10% for 91% of the time.
PC1	10	10%	Less than 10% for 86% of the time.
security lab	10	6%	Less than 10% for 93% of the time.

Average CPU usage for each lab is displayed in the chart diagram below.

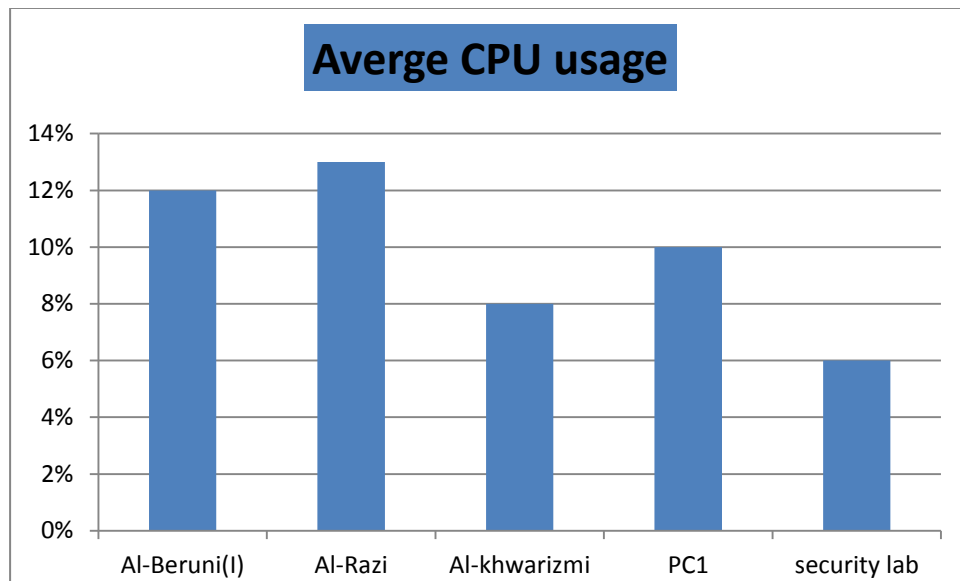


Figure 7.4: Average CPU usage at PPU labs.

The stated averages in the previous table represent the average of CPU usage in the labs during the lectures since most of times the computers are turned off after each lecture. So they are running only when they are under use. This means that the average CPU usage of the computers will be decreased much more than shown, that is if they remain running all the time, and so increase the ability of providing more computational power.

Depending on table 7.3; **the average CPU usage at the stated labs together is:**

- **approximately 9.3% using Ultra CPU monitor**
- **and approximately less than 10% for 87.9% of the time, using CPU usage logger:**

The previous results can be generalized to represent the average CPU usage at PPU computer labs. They are come in favour with many researches on CPU utilization of PCs, some of them:

- Bader Al-Ajrab stated that "the average CPU usage is not far exceed 10% for 90% of the time"[18], his research applied at Al-Quds Open University in 2013.

- Domingues stated CPU idleness is impressively high with an average of 97.93%"[36].

Results show that computers at PPU computer labs have a large computational power that is not utilized, so it can be used to build a grid computing system.

7.3 Examination the performance of PPU Environment

The goal of this experiment is basically to prove that the Grid System is running and working properly. By this experiment we show that our grid computing environment is ready for further studies and researches .In addition, we try to get an approximate estimation of the PPU environment performance. Arriving to an exact estimation of the PPU environment performance is out of the scope of this project.

We used the Giga floating point operations per second (GFLOPS) that is obtained from the system as the performance metric. Larger number of GFLOPS obtained from a system means better performance of this system. GFLOPS can be calculated using the equation below:

$$\mathbf{GFLOPS} = \mathbf{F} \times \mathbf{C} \times \mathbf{n} \dots\dots\dots [44]$$

Where:

- F: CPU frequency.
- C: number of cores.
- n: number of floating point operations per CPU cycle. Most likely equals 4.

Pre-requirements needed for doing this experiment:

- A running BOINC Test project.
- Sample of computing resources ready for attaching to the Test project.

To handle these requirements we do the following:

- Create a BOINC Test project named **PPUTest**.

- Install the BOINC client software on all PCs of Al-Khwarizmi lab and set the appropriate configurations. The details of installation and setting configurations are discussed in appendix A.
- ✱ The specification of Al- Khwarizmi lab PCs are Intel, Core i5-3470 CPU, 3.20GHz. The theoretical number of GFLOPS obtained by one of these PCs can be calculated using the previous equation as follow:

$$\text{GFLOPs} = 4 \text{ (cores)} \times 3.2 \text{ GHz} \times 4 \text{ (floating operations per cycle)} = 51.2 \text{ GFLOPS.}$$

The practical gained GFLOPS is smaller than the theoretical. There are many factors that make the practical GFLOPS smaller than the theoretical:

- A computer cannot run at the maximum utilization all the time; it may be exposed to a hardware failure.
- CPU is not always available for the test computations; it is shared for all purposes.
- Many times computers are not running, or even not connected to the network.
- Many times server itself is not running or not connected to the network.
- The test application used in the experiment is so simple so it doesn't need to a large computational power.

The applied experiment has totally 39 PCs working as grid clients and having the same specifications. This experiment was applied for ten days. We obtained approximately 96 GFLOPS as the average of GFLOPS can be produced from all of these PCs together. Through the analysis of the results we found that:

- Twelve of these PC were not working properly (this found from the number of work units done by these PCs, which was too low comparing with other computers). This may happen because these PCs were disconnected to the network for long periods of time during the experiment. So, we will ignore these PCs and their effect which is about 3 GFLOPS.

As a result we have 27 PCs producing 93 GFLOPS:

$$\text{GFLOPS per PC} = \frac{\text{total number of GFLOPS gained}}{\text{number of PCs}} = \frac{93}{27} = 3.45\text{GFLOPS}$$

PPU has approximately 1000PCs distributed over all the university. If we suppose that all PPU PCs have a similar specification of AL-Khwarizmi lab PCs, then attaching all PPU PCs to the grid system produces:

$$\text{GFLOPS} = \text{GFLOPS per PC} \times \text{number of PC} = 3.45 \times 1000 = 3.45\text{TeraFLOPS}$$

Many limitations must be considered when taking this result:

- It is generalized from one test of a small sample. To determine the performance of the environment many tests with different sample sizes are needed to be performed.
- The test was done over ten days only, which is a short period to generalize on all semester. Computers usage may differ during the semester life cycle, which surely affects the number of GFLOPS obtained from each PC.
- The test is applied only on windows 7 operating system PCs. So, it doesn't take into consideration the effect of the operating system. Tests must be applied on different OS since each OS make different usage to the PC resources.
- Network effect is not taken into consideration since only AL-Khwarizmi lab computers were used in this test. Grid clients may exist in different locations that have different link speeds which surely affect the performance.
- Only one application was used in the test; which is the BOINC test application. CPU usage may differ for different applications that make different access to memory or Input/Output. So, many tests with different applications must be performed.
- The BOINC test application used in the test is so small and simple which is not the case of many real applications. So, it may not increase the utilization of the CPU usage as well.

7.4 Summary

In this chapter we talked about the CPU utilization at PPU computer labs. Through this experiment we found that the average CPU usage at PPU labs doesn't far exceed 9%; this mean there is a huge computational power can be obtained by these resources to perform important researchs. Also we performed a small experiment to estimate how many FLOPs can be obtained from the PPU environment. Through this experiment we found that each PC prooduces approximatly 3.45GFLOPS.

Chapter Eight

Conclusion and Future Work

8.1 Overview

In this chapter, we introduce the overall project conclusion and talk about the challenges appeared during the project. Also, we talk about some fields that are important to be taken in to consideration within the future work.

8.2 Conclusion

This project has two main objectives. The first objective is to highlight the amount of wasted computing power during idle CPU cycles. The second objective is to build a local PC grid computing system in the university to utilize the available idle CPU power in computer labs.

The first objective is achieved by performing a study to find the CPU utilization percentage in the university. The result of this study shows that CPU utilization in computer labs is very low. The average CPU utilization for a whole working day (08:00-16:00) is less than 10%, this agrees with all surveyed researches in this field. Our results emphasize the fact that available idle CPU power is large, and can be employed to build a local Grid computing system.

The second objective is achieved by building the Campus Grid Computing System which is done as follow:

- Installing BOINC server on one of the university PCs.
- Installing BOINC client on a sample of the university PCs.

- Creating shell scripts that provide the core functionalities of the system.
- Creating a web portal for Campus Grid Computing System.

Campus Grid Computing System implements two levels of transparency. The lower level of transparency consists of shell scripts that provide the core functionalities of the system. The higher level of transparency which is the web portal is built over the lower level. This level provides user interfaces that enable performing any functionality without the need to write a terminal commands directly on the server; Users can perform any functionality remotely through user friendly interfaces and without seeing the underlying levels of the system.

One of the experiments that were done using the Campus Grid Computing System is examining how many FLOPs can be obtained from the PPU environment. This study proved that the Grid System is running and working properly. Through this experiment each PC produces approximately 3.45 GFLOPS.

BOINC middleware is centralized system where the server works on managing all resources and jobs. This has some drawbacks, which are:

- The need for a powerful central server to manage all slave computing nodes.
- Single point of failure; if the grid server crashed all the system will stop evaluation.

8.3 Challenges

During the work over this project we face many challenges and difficulties. The following paragraphs talk about these challenges and difficulties.

Project scope is large and requires the team members to be from different backgrounds to work on. Project team must be experienced with parallel

programming, Linux administrative skills, web development, and security considerations for all of these sides.

BOINC is a powerful tool that supports volunteer computing, which we customized to support grid computing. In contrast, there are many challenges for using BOINC. The complexity of BOINC; such that developers need to put large efforts on learning the BOINC environment before being able to start using it.

Moreover, BOINC environment is still under construction. So, BOINC documentation is not complete and many topics are mentioned abstractly. Many times we arrived to solutions based on our overall understanding of BOINC complexity.

In addition, the lack of previous studies in this field increases the difficulty of going on with this project. This project approximately considered the first graduation project specialized in the field of grid computing over BOINC middleware.

Many difficulties also come with the experiments that we perform over PPU environment. We need to enter the labs many times each day of the study of CPU utilization to run the CPU usage logger program and at the end of the day we need to come back to take the results. But, we were not being able to enter to labs freely during the lectures. Moreover, all the labs were heavily loaded with lectures. So we forced to wait the breaks between lectures to do our work that is too short and not enough to finish the work.

In addition, turning off the programs by the users was one of the most annoying difficulties. Both of the programs used in the study of CPU utilization don't have the ability to run in background so many times users close them. When we come back at the end of the day to take the results we shocked that programs are closed and no valid data exist. Since of that we were forced to do the work again and again.

Another difficulty that we face is shutting down the computers at the end of lectures. This affects the results of both of the programs used at the CPU utilization study. It also affects the number of work units performed by the computers which represents their performance at the grid system.

8.4 Future Work

Although, we have obtained promising initial results, but still the following points may help to further contributions in Grid computing:

1. Developing GPU Applications

While our project mainly focuses on utilizing idle CPUs power in our computer labs, BOINC supports applications that use coprocessors. The supported coprocessor types are NVIDIA, AMD, and Intel GPUs [45]. The Graphical Processing Unit (GPU) provides a very large computing power which is usually not utilized well.

The computing power of GPUs has increased rapidly, and they are now often much faster than the computer's main processor, or CPU [46]. The wasted power of computers' GPUs can be utilized by building BOINC GPU applications. You can develop your application using any programming system, e.g. CUDA (for NVIDIA), CAL (for ATI) or OpenCL [45].

2. Grid Data Archival

PCs nowadays have large disk space capacities while only small portion of this space is used by normal users. A study can be performed on the university PCs to find the available disk space capacities and their utilization percentage. In the light of this study, we can determine the efficiency of building a distributed data storage system to make use of unused disk capacities.

BOINC is currently used for computation, but it also provides primitives for distributed data storage: file transfers, queries, and deletion [47]. It is possible to develop a system that uses these primitives to implement a distributed data archival system

3. Testing BOINC client performance

Most of our experiments were done under Windows XP and Windows 7. However, we can test the BOINC client software performance under different platforms like UNIX based operating systems. Also, it is a good idea to test BOINC client under Windows 8 which takes its place in the market and it may be used in our labs in near future.

4. Mass Deployment Of BOINC Client Software

Deploying BOINC client software manually on a large number of computers is not an easy task; it needs a large effort and takes a lot time repeating the same process.

The BOINC installer uses the Microsoft MSI technology framework so we can customize the BOINC installer properties. In addition, we can control

certain aspects on the installation process by launching the BOINC installer with certain command line options. For example, we can automate the process of client deployment using unattended or silent install by executing a installer command with unattended or silent command line argument (see reference [48]).

5. Developing and porting BOINC applications

This involves meeting the researchers, identifying those with computationally-intensive problems that map well to grid computing. The applications used by those researchers are then ported to BOINC.

On the other hand, adopting an existing application to run within BOINC environment can be reached by two approaches:

a. Using BOINC wrappers

This is the simplest way. The existing application can be run with no modifications.

b. Writing Native BOINC applications

With some minor source code modifications, you can run an application directly without need for the wrapper [49]. The changes are [49]:

- Adding calls to BOINC initialization and finalization routines.
- Preceding each fopen() call with a BOINC function that maps logical to physical names.
- Linking it with the BOINC runtime library

6. Test BOINC Wrappers

Any existing application (or sequence of applications) can be run under BOINC using a **wrapper** program supplied by BOINC. The wrapper runs the applications as sub-processes, and handles all communication with the BOINC client (e.g., to report CPU time and fraction done) [50].

A study can be performed to test the effect of using BOINC wrappers to run the applications. The study should determine the overhead, effectiveness, advantages and disadvantages of wrappers. The result of this study may suggest mechanisms to enhance wrappers performance.

In addition, it will be a good idea to compare the performance of an application; one time is run under BOINC wrapper and the other time is run as native BOINC application.

7. Determine Best Parameters (BOINC experiments)

BOINC provides different parameters to control and specify preferences that limit when and how BOINC uses the computers under a certain account. Many experiments can be performed to determine the best values for these parameters like processor, disk and memory usage.

8. Use BOINC with different programming languages

BOINC is originally designed to work with **C/C++** applications, but it provides some mechanisms to adopt other programming languages like **Java** and **Python**. We can try to develop applications for these programming languages. In addition, we can enhance the single job submission to include new programming languages other than **C/C++**.

9. Build BOINC software add-ons using BOINC API

BOINC Project is still under work and it is developed on volunteer bases. One can participate in different fields of this project as a volunteer and can be part of the world wide effort in this area.

10. Build a volunteer computing system (Investigate Volunteer Computing).

Because of the huge number (more than one billion) of PCs in the world, volunteer computing can supply more computing power to science than does any other type of computing [51]. BOINC is originally designed for volunteer computing; so it can be used to build a volunteer system.

11. Deploy BOINC server on different UNIX and Linux distributions.

The BOINC server must be a UNIX computer, generally running Linux [52]. In this project we deployed the BOINC server on Ubuntu 12.04, but we can try to get the experience of deploying the BOINC server on different Linux distributions.

12. Build a tool to control BOINC clients remotely

The BOINC manager provides a graphical user interface to facilitate the communication with only one BOINC core client locally or remotely. In a real campus grid system, it is essential to control the BOINC core client remotely on all computers participate in the grid. For this purpose, an add-on tool can be built using **boinccmd** tool or **BOINC APIs**.

13. Build a Palestine Universities Grid System (PUGS)

We can generalize the project idea by building a large scale grid system that encloses all the Palestinian universities. This grid system will provide a large computational power that will develop and enhance scientific researches in Palestine.

14. Launch our first BOINC based real project that does real computation or performs a scientific research.

This can be done by helping researchers in the university to develop BOINC based projects that serve their research fields. The publication of such projects and their results can increase the publicity of the university.

15. BOINC For Android

Mobile devices such as smartphones and tablets are small, but they have serious computing power - as much as 25% of an average desktop computer [53]. In addition there are a huge number of android devices in the world and their market is growing rapidly.

BOINC client software is now available for android devices; so it is important to develop applications for android devices exploit their computational power which may play an important role in the future for scientific computing.

16. Investigating other grid computing middle-wares and tools (e.g. Globus, Alchemi).

17. Mobile Agent Grid System

BOINC follows a client-server centralized approach. This centralization of the system produces some limitations on the BOINC based systems.

Mobile agent works on decentralization manner. It also supports the interaction and the communication between agents during their execution on different hosts. So, mobile agent based systems can avoid the limitations come from centralization. But in the other hand, different limitations will appear since of the nature of mobile agents.

As an example of mobile agent based middleware systems is Agent Teamwork “is a grid-computing middleware system that dispatches a collection of mobile agents to coordinate a user job over remote computers in a decentralized manner”[69].

8.5 Summary

In this Chapter, we summarized the main results that were found through this project, then talk about the challenges that appeared during the project. Finally, we stated the main issues that are considered as a future work.

References

- [1] BOINC: Berkeley Open Infrastructure for Network Computing, <http://boinc.berkeley.edu/>, (accessed on 13/11/2013).
- [2] What is grid computing?, <http://www.gridcafe.org/EN/what-is-the-grid.html>, (accessed in September, 2013).
- [3] Bader Ahmed Bader Ajrab, "PC Grid Computing Environment In Higher Education Institutions", master thesis at AlQuds university, Palestine, 2013.
- [4] Grid computing in 30 seconds, <http://www.gridcafe.org/EN/grid-in-30-sec.html> , (accessed in September, 2013).
- [5] D.P. Vidyarthi, B.K. Sarker, L.T. Yang, "Scheduling in Distributed Computing Systems Analysis, Design & Models", A Research Monograph, pp.(244-245), 2009
- [6] Grid architecture, <http://www.gridcafe.org/EN/grid-architecture.html> , (accessed in October, 2013).
- [7] Middleware, <http://www.gridcafe.org/EN/middleware.html>, (accessed in October, 2013).
- [8] Ault, M. and Tumma, M., "Oracle10g Grid Computing with RAC", Oracle RAC - Types of Grid computing, 2004.
- [9] Stanoevska K., Wozniak T., Ristol S., "Grid and Cloud Computing: A Business Perspective on Technology and Applications", Springer, 2010.
- [10] Goyal B, Lawande S (2005) Grid Revolution: An Introduction to Enterprise Grid Computing. McGraw-Hill, Emeryville, CA, 2005
- [11] Joseph J., Fellenstein C., "Grid Computing", Pearson Education, 2004.
- [12] Ian Foster, Carl Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 2004.
- [13] Fox, G., Furmanski, W. (1998) High performance commodity computing, Chapter 10, in Foster, I. and Kesselman, C. (eds) The Grid: Blueprint for a New Computing Infrastructure. San Francisco, CA: Morgan Kaufmann Publishers.
- [14] Foster, I. and Kesselman, C., "The grid: Blueprint for a new computing infrastructure", Morgan Kaufmann, San Francisco, CA, 1998.

- [15] Fran Berman, Anthony J.G. Hey, Geoffrey C. Fox,
"Grid Computing Making the Global Infrastructure a Reality", Wiley Series in
Communications Networking and Distributed Systems, pp.(722-723),2003.
- [16] Grid-powered projects, <http://www.gridcafe.org/EN/grid-powered-project.html> ,
(accessed in October, 2013).
- [17] Volunteer computing , <http://www.gridcafe.org/EN/volunteer-computing.html>
,(accessed on September, 2013).
- [18] Desktop Grid, <http://boinc.berkeley.edu/trac/wiki/DesktopGrid>, (accessed in
October, 2013).
- [19] Volunteer computing vs. cloud vs. grid vs. HPC ,
<http://www.volunteer-computing.org/EN/volunteer-computing-vs-cloud-vs-grid-vs-HPC.html>, (accessed in October, 2013).
- [20] Volunteer computing , http://en.wikipedia.org/wiki/Volunteer_computing,
(accessed in October, 2013).
- [21] Desktop_Grid:Westminster_Local_DG,
http://wgrass.wmin.ac.uk/index.php/Desktop_Grid:Westminster_Local_DG,
(accessed in October, 2013).
- [22] NEW DIY SUPERCOMPUTER SAVES £1,000S,
<http://www.westminster.ac.uk/news-and-events/news/2011/university-of-westminster-launches-new-diy-supercomputer-saving-hundreds-of-thousands-of-pounds>, (accessed in October, 2013).
- [23] Middleware - Volunteer garage, <http://www.volunteer-computing.org/EN/middleware.html>, (accessed in October, 2013).
- [24] Anderson," BOINC: A system for public-resource computing and storage", 5th
IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA, pp. 4-
10,Dec,2004.
- [25] Grid MP - Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Grid_MP, (accessed in October, 2013).
- [26] Alchemi v0.6.1 Documentation,
http://www.cloudbus.org/~alchemi/doc/0_6_1/index.html, (accessed in October,
2013).

- [27] Alchemi [.NET Grid Computing Framework], <http://www.cloudbus.org/~alchemi/>, (accessed in October, 2013).
- [28] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchem Framework", Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.
- [29] SETI, <http://setiathome.berkeley.edu/>, (accessed on 13/11/2013)
- [30] BOINCstats, <http://boincstats.com/>, (accessed on 13/11/2013)
- [31] TOP500 list, <http://www.top500.org/list/2012/11/>, (accessed on 13/11/2013).
- [32] Anderson, D., Korpela, E. and Walton, R., —High-Performance Task Distribution for Volunteer Computing, Proceedings of the First IEEE International Conference on e-Science and Grid, . Melbourne, Australia, 2005.
- [33] JobSubmission-Boinc. <http://boinc.berkeley.edu/trac/wiki/JobSubmission>, (accessed in January, 2014).
- [34] JobTemplates-Boinc. <http://boinc.berkeley.edu/trac/wiki/JobSubmission>, (accessed in January, 2014).
- [35] Free CPU usage monitor programs, <http://softwaresolution.informer.com/Free-CPU-Usage-Monitor/>.(accessed in January, 2014).
- [36] Patricio Domingues, Paulo Marques, Luis Silva,"Resources Usage of Windows Computer Laboratories", α Escola Superior de Tecnologia e Gest3o – Instituto Polit3cnico de Leiria – Portugal β Departamento Eng. Inform3tica, Universidade de Coimbra – Portugal, Jan, 2005.
- [37] BasicConcepts-BOINC, <http://boinc.berkeley.edu/trac/wiki/BasicConcepts>, (accessed on 06/05/2014).
- [38] Example applications, <http://boinc.berkeley.edu/trac/wiki/ExampleApps#no1>, (accessed on 8/5/2014).
- [39] SingleJob-BOINC, <http://boinc.berkeley.edu/trac/wiki/SingleJob>, (accessed on 07/05/2014).
- [40] HtmlOps-BOINC, <http://boinc.berkeley.edu/trac/wiki/HtmlOps>, (accessed on 04/04/2014).
- [41] M. Alfalayleh and L. Brankovic, "an overview of security issues and techniques in mobile agents", The University of Newcastle, 2004.

- [42] SecureHttp-BOINC, <http://boinc.berkeley.edu/trac/wiki/SecureHttp>, (accessed on 02/04/2014).
- [43] Free CPU Usage Monitor, <http://softwaresolution.informer.com/Free-CPU-Usage-Monitor/>, (accessed on 10/01/2014).
- [44] FLOPS - Wikipedia, the free encyclopedia, <http://en.wikipedia.org/wiki/FLOPS>, (accessed 22/05/2014).
- [45] AppCoprocesor, <http://boinc.berkeley.edu/trac/wiki/AppCoprocesor>, (accessed on 06/05/2014).
- [46] GPU Computing-BOINC, http://boinc.berkeley.edu/wiki/GPU_computing, (accessed on 06/05/2014).
- [47] ResearchProjects-BOINC, <http://boinc.berkeley.edu/trac/wiki/ResearchProjects>, (accessed on 06/05/2014).
- [48] Creating custom installers, http://boinc.berkeley.edu/wiki/Creating_custom_installers, (accessed on 06/05/2014).
- [49] AppIntro-BOINC, <http://boinc.berkeley.edu/trac/wiki/AppIntro>, (accessed on 08/05/2014).
- [50] WrapperApp-BOINC, <http://boinc.berkeley.edu/trac/wiki/WrapperApp>, (accessed on 08/05/2014).
- [51] VolunteerComputing-BOINC, <http://boinc.berkeley.edu/trac/wiki/VolunteerComputing>, (accessed on 10/05/2014).
- [52] Creating and Configuring a BOINC Project, http://www.spy-hill.net/myers/help/boinc/Create_Project.html#server, (accessed on 10/05/2014).
- [53] Android FAQ-BOINC, http://boinc.berkeley.edu/wiki/Android_FAQ, (accessed on 10/05/2014).
- [54] Setting up a BOINC server, <http://boinc.berkeley.edu/trac/wiki/ServerIntro>, (accessed on 10/1/2014).
- [55] BOINC server guide installation, <https://wiki.debian.org/BOINC/ServerGuide/Initialisation>, (accessed 15/1/2014).
- [56] Installing BOINC, http://boinc.berkeley.edu/wiki/Installing_BOINC, accessed on 02-May- 2014
- [57] Installing BOINC On Ubuntu, http://boinc.berkeley.edu/wiki/Installing_BOINC_on_Ubuntu, (accessed on 02/05/2014).

- [58] Controlling BOINC Remotely, http://boinc.berkeley.edu/wiki/Controlling_BOINC_remotely, (accessed on 02/05/2014).
- [59] BOINC DB, <http://boinc.berkeley.edu/trac/wiki/DataBase>, (accessed on 8/5/2014).
- [60] Server directory structure, <http://boinc.berkeley.edu/trac/wiki/ServerDirs>, (accessed on 8/5/2014).
- [61] Project configuration file, <http://boinc.berkeley.edu/trac/wiki/ProjectConfigFile>, (accessed on 8/5/2014).
- [62] Server Components, <http://boinc.berkeley.edu/trac/wiki/ServerComponents>, (accessed on 8/5/2014).
- [63] Setting up a BOINC server, <http://boinc.berkeley.edu/trac/wiki/ServerIntro>, (accessed on 8/5/2014).
- [64] anonscm.debian.org Git - pkg-boinc, http://anonscm.debian.org/gitweb/?p=pkg-boinc/scripts.git;a=blob:f=server-examples/boinc_project_maker.sh, (accessed in January, 2014).
- [65] Code signing, <http://boinc.berkeley.edu/trac/wiki/CodeSigning>, (accessed on 02/04/2014).
- [66] KeySetup-BOINC, <http://boinc.berkeley.edu/trac/wiki/KeySetup>, (accessed on 02/04/2014).
- [67] StartTool-BOINC, <http://boinc.berkeley.edu/trac/wiki/StartTool>, (accessed on 12/05/2014).
- [68] Boinccmd tool-BOINC, http://boinc.berkeley.edu/wiki/Boinccmd_tool, (accessed on 07/04/2014).
- [69] The design concept and initial implementation of Agent Teamwork grid computing middleware, <http://www.academicpub.com/map/items/3933371.html>, (accessed on 28/5/2014).
- [70] BOINC Security-BOINC, http://boinc.berkeley.edu/wiki/BOINC_Security, (accessed on 30/05/2014).

Appendix A

Server and Client Software Installation

This appendix describes the deployment process of BOINC software. First, we show the installation and configuration process of BOINC server, then we describe the installation of BOINC client software on different platforms.

A.1 BOINC server Pre-installation requirements

There are pre-installation requirements that must be satisfied before installing the BOINC server. These requirements are classified as hardware requirements and software requirements.

A.1.1 Hardware requirements

Hardware requirements which are needed for running a BOINC server vary according to the size of the grid system and the type of services provided by the system. In general, any computer can be used as a BOINC server if its usage just for experiments and debugging purposes. However, before deploying the system more widely, we have to make sure that the server has adequate performance, availability, and security. Here are some factors [54]:

- Internet connection should have adequate performance and reliability.
- Server must have a static IP address.
- Server should have:
 - Good CPU speed(dual Xeon or Opteron):
 - At least 2 GB of RAM,
 - At least 40 GB of free disk space.
 - For a high-traffic project, use a machine with 8 GB of RAM or more.
- All these factors suppose that the server used to serve only one BOINC project, more details about BOINC project stated in Appendix B.

In our case the server has:

- Processor: core i5 3.2GHz.

- RAM: 4GB
- Disk storage capacity: 320GB.
- Network: 100Mbps.
- Global IP address: 195.3.191.24
- Local IP address: 10.10.16.12

A.1.2 Software requirements

The currently stable and up to date version of BOINC server runs on any 64-bit UNIX operating system. In our project we used Ubuntu12.04 LTS.

There are some additional software dependences needed for BOINC server like apache server, PHP, MYSQL, and other packages that must be installed before BOINC server installation. These dependences can be installed by running the following terminal command:

- `sudo apt-get install git build-essential apache2 php5 \`
`mysql-server php5-gd php5-cli php5-mysql python-mysqldb \`
`libtool automake autoconf pkg-config libmysql++-dev libssl-`
`dev`

A.2 BOINC server installation process

System installation process goes into the following steps:

1. Download BOINC source:
 - `git clone git://boinc.berkeley.edu/boinc-v2.git boinc`
2. Compile BOINC:
 - `cd boinc`
 - `./_autosetup`
 - `./configure --disable-client --disable-manager`
 - `make`

A.3 Trouble Shooting

This section shows some problems that may arise during the compilation process of BOINC server and provides solutions for these problems.

- Fatal error: curl/curl.h: No such file or directory:

Solution: `sudo apt-get install libcurl4-gnutls-dev`

- Problem: error with (m4) :

Solution: download and install m4-1.4.1

- `sudo apt-get install m4.1`

- Problem: configure: WARNING: fcgi-stdio.h not found.

Disabling FCGI. Will not build components that require FCGI

Solution: Install the following library [55]: libfcgi-dev

- `sudo apt-get install libfcgi-dev`

- Problem with openssl directory.

Solution: use instead of the Compile BOINC commands (step 2 in section A.2) the following one:

```
cd boinc; ./_autosetup; ./configure --with-ssl=/usr/include/openssl --disable-client --disable-manager; make
```

- For assurance that we have updated version of the BOINC source code we need to execute the command:

- `git pull`

- After fixing all the problems we need to re-compile the BOINC source code.

The following figures show the compilation process that has some problems:

```
muhammad@ubuntu: ~/boinc
checking if CFLAG '-include fcgi_stdio.h' works... no
configure: WARNING: fcgi_stdio.h not found.
-----
Disabling FCGI. Will not build components that require FCGI
-----
checking for pkg-config... /usr/bin/pkg-config
checking for openssl... yes
OpenSSL found in /usr
checking for shmget in dynamic library cygipc... no
checking for aio_fork in dynamic library aio... no
checking for dlopen in dynamic library dl... -ldl
checking for gethostbyname in static library nsl... -lnsl
checking for bind in static library socket... no
checking for bind in dynamic library socket... no
checking for gzopen in static library z... -lz
checking for md5_finish in dynamic library cups... no
checking for the pthreads library -lpthreads... no
checking whether pthreads work without any flags... no
checking whether pthreads work with -Kthread... no
checking whether pthreads work with -kthread... no
checking for the pthreads library -llthread... no
checking whether pthreads work with -pthread... yes
checking for joinable pthread attribute... PTHREAD_CREATE_JOINABLE
```

Figure A.3.1 BOINC installation problems 1.

```
checking for fopen in -lXi... no
checking for GLUT library... no
configure: WARNING:
-----
WARNING: Development libraries and headers ("-dev") of {openGL, GLU, glut} needed!

The GL, GLU and glut libraries are required in order to build the graphical parts
of the BOINC application API library.

==> only building non-graphical parts of the BOINC API Library for now.

HINT: on MacOS X/Darwin you might consider running configure with the option
'./configure --with-apple-opengl-framework'
in order to use the Mac-native openGL framework
-----
checking for XScreenSaverAllocInfo in -lXss... no
```

Figure A.3.2 BOINC installation problems 2.

```
muhammad@ubuntu: ~/boinc
make[2]: Leaving directory `/home/muhammad/boinc/apps'
Making all in tools
make[2]: Entering directory `/home/muhammad/boinc/tools'
CXXLD cancel_jobs
CXX create_work.o
CXXLD create_work
CXX dir_hier_move.o
CXXLD dir_hier_move
CXX dir_hier_path.o
CXXLD dir_hier_path
CXX remote_submit.o
./lib/remote_submit.cpp:24:23: fatal error: curl/curl.h: No such file or directory
compilation terminated.
make[2]: *** [remote_submit.o] Error 1
make[2]: Leaving directory `/home/muhammad/boinc/tools'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/home/muhammad/boinc'
make: *** [all] Error 2
muhammad@ubuntu: ~/boinc
```

Figure A.3.3 BOINC installation problems 3.

When the compilation finished properly, the result will be as shown in figure A.3.4 below:

```
muhammad@ubuntu: ~/boinc2
Making all in vda
make[2]: Entering directory `/home/muhammad/boinc2/vda'
CXX vda.o
CXX vda_lib2.o
CXX stats.o
CXXLD vda
CXXLD vdad
CXXLD ssm
make[2]: Leaving directory `/home/muhammad/boinc2/vda'
Making all in html
make[2]: Entering directory `/home/muhammad/boinc2/html'
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/home/muhammad/boinc2/html'
Making all in doc
make[2]: Entering directory `/home/muhammad/boinc2/doc'
make[3]: Entering directory `/home/muhammad/boinc2/doc'
make[3]: Nothing to be done for `all-am'.
make[3]: Leaving directory `/home/muhammad/boinc2/doc'
make[2]: Leaving directory `/home/muhammad/boinc2/doc'
make[2]: Entering directory `/home/muhammad/boinc2'
cd . && sh generate_svn_version.sh
make[2]: Leaving directory `/home/muhammad/boinc2'
make[1]: Leaving directory `/home/muhammad/boinc2'
muhammad@ubuntu:~/boinc2$
```

Figure A.3.4 Proper BOINC installation.

A.4 BOINC Client Installation

This section describes the deployment process of BOINC client software on different platforms.

A.4.1 Microsoft Windows

Most of installation work is done on Microsoft Windows operating system which is used by most of our computers. First, we need to download the BOINC installer for Windows, and then double-click the installer icon. After that, we follow the installation process that is described in the figures below (Figure A.4.1 to Figure A.4.8).

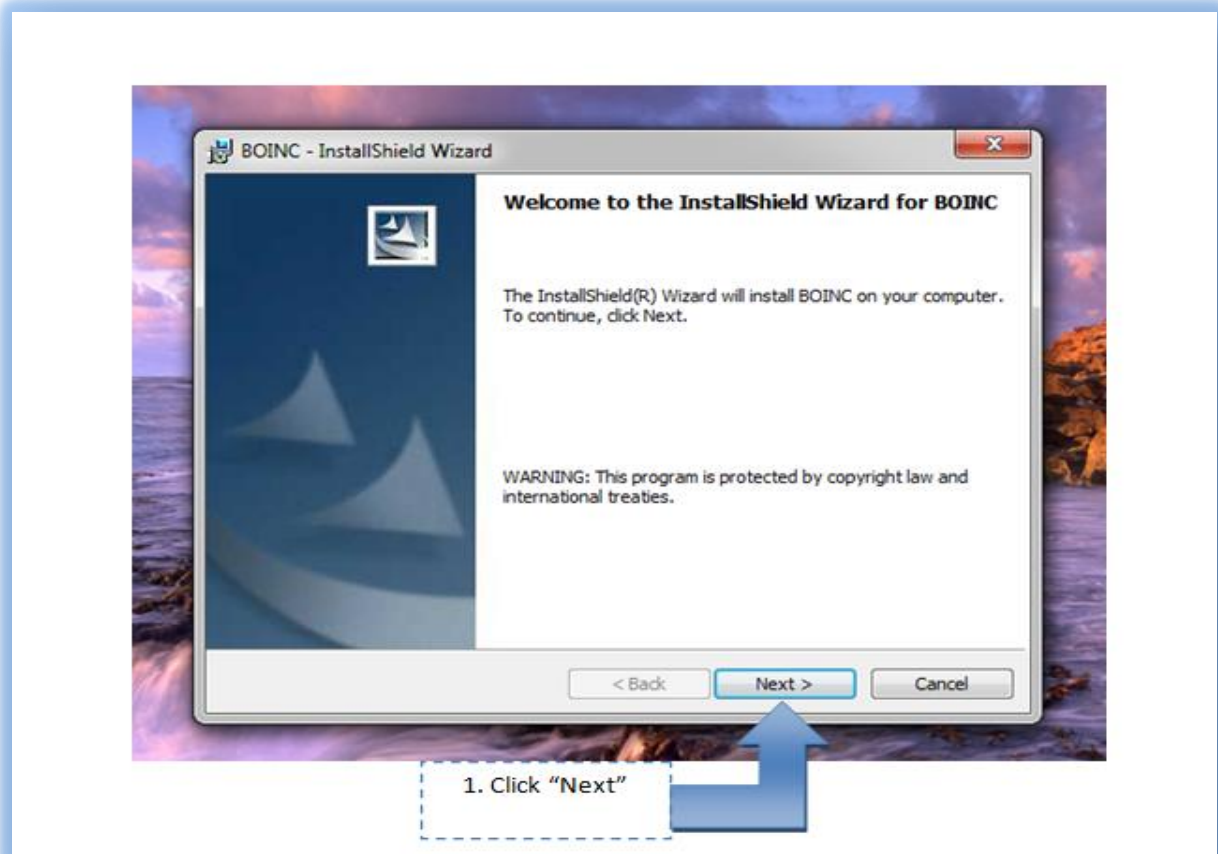


Figure A.4.1: BOINC Client Deployment Step 1

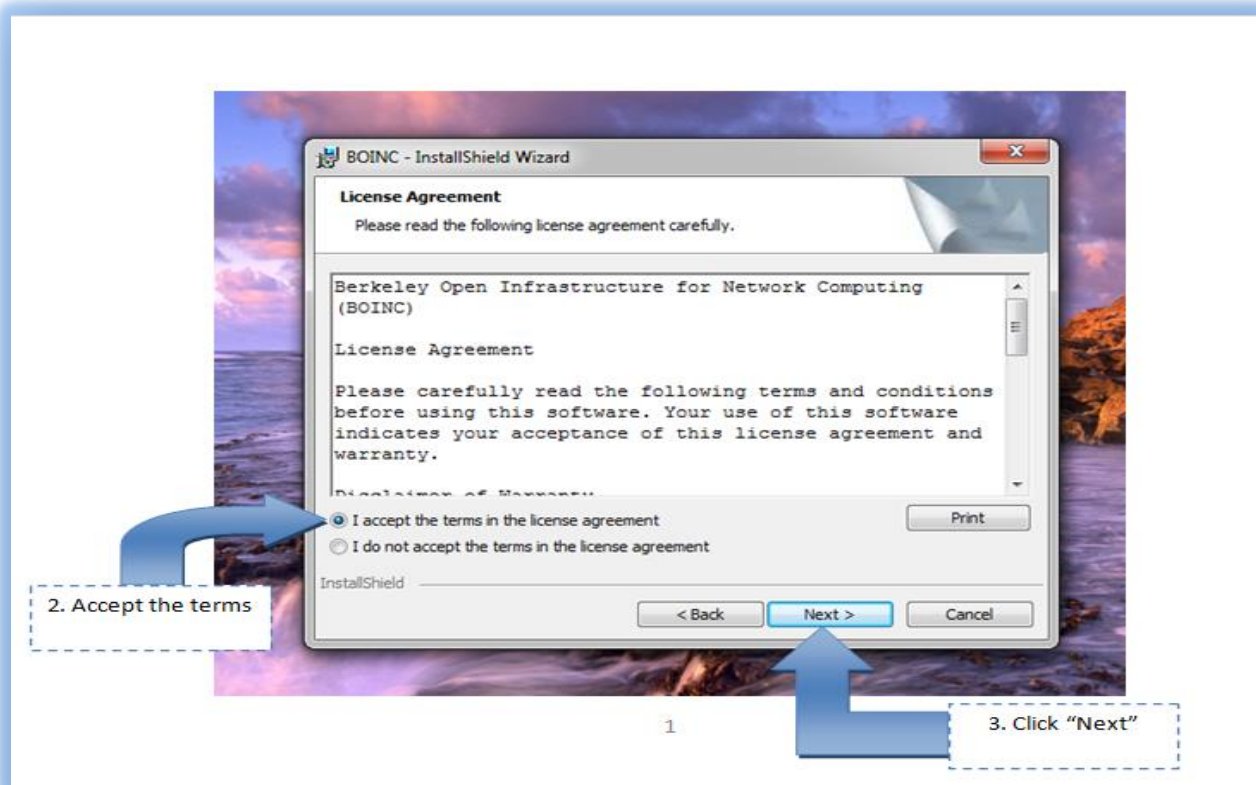


Figure A.4.2: BOINC Client Deployment Step 2

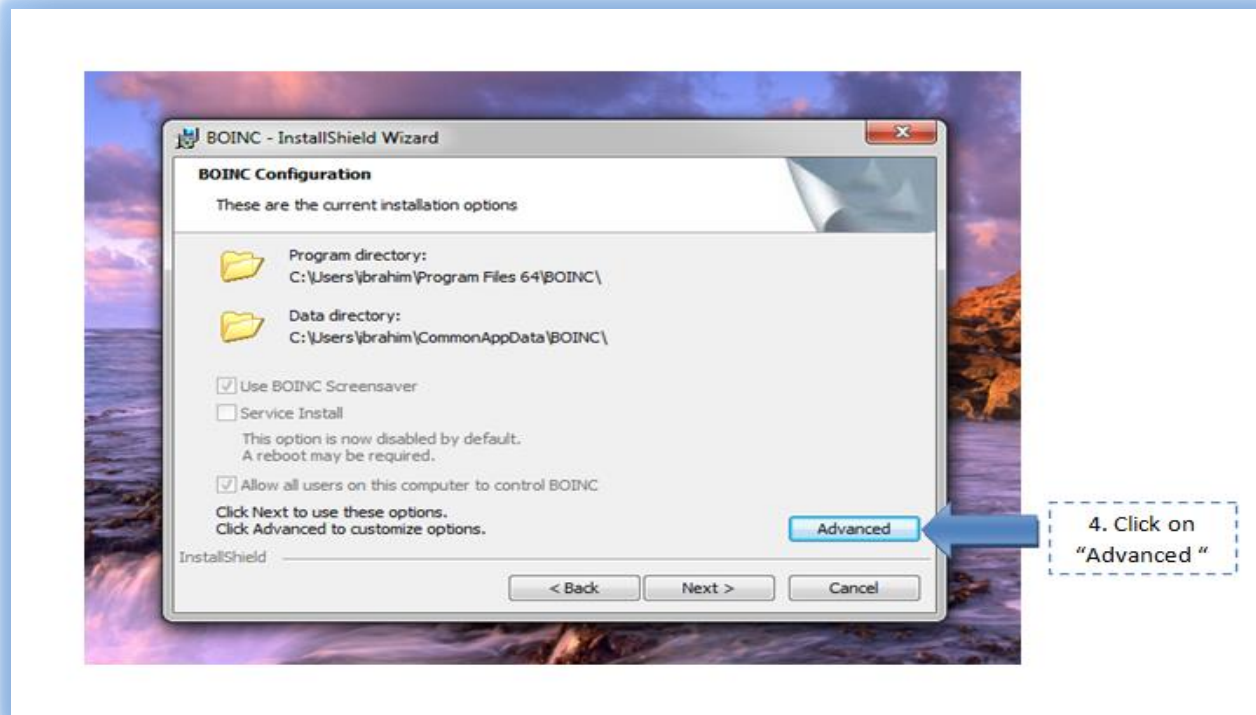


Figure A.4.3: BOINC Client Deployment Step 3

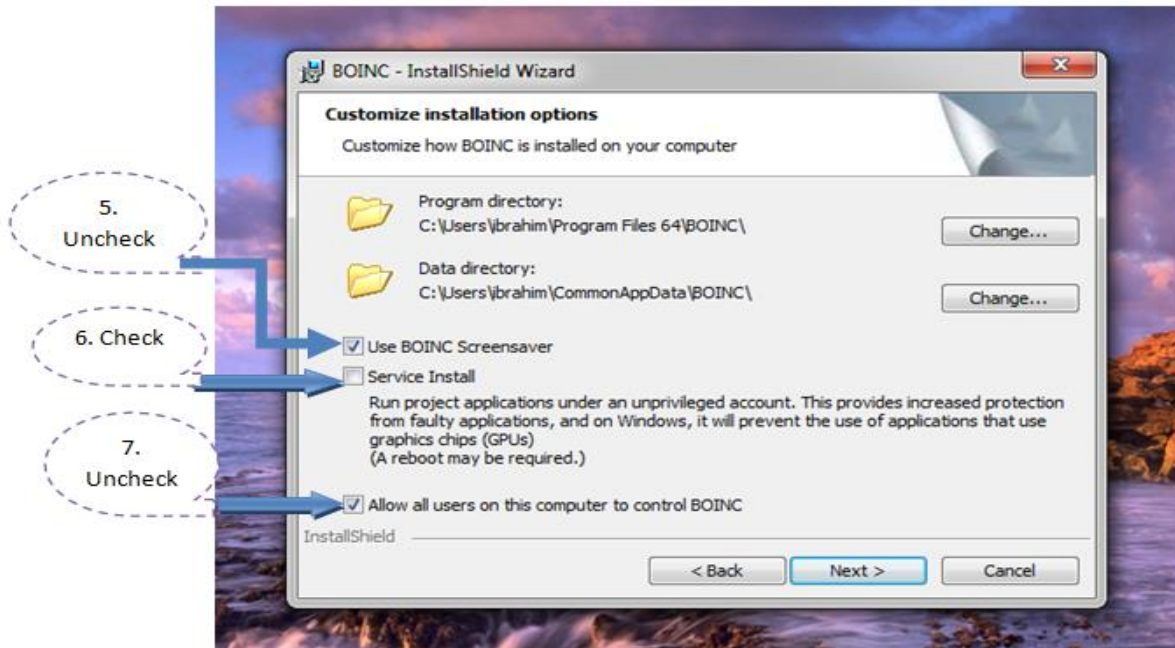


Figure A.4.4: BOINC Client Deployment Step 4

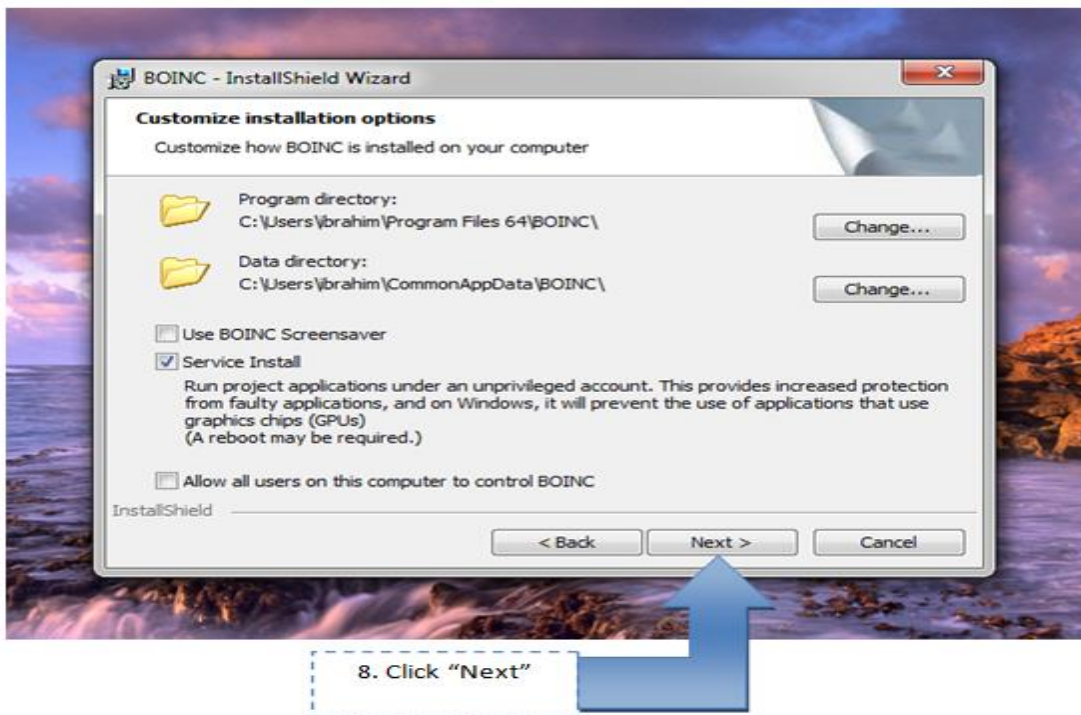


Figure A.4.5: BOINC Client Deployment Step 5

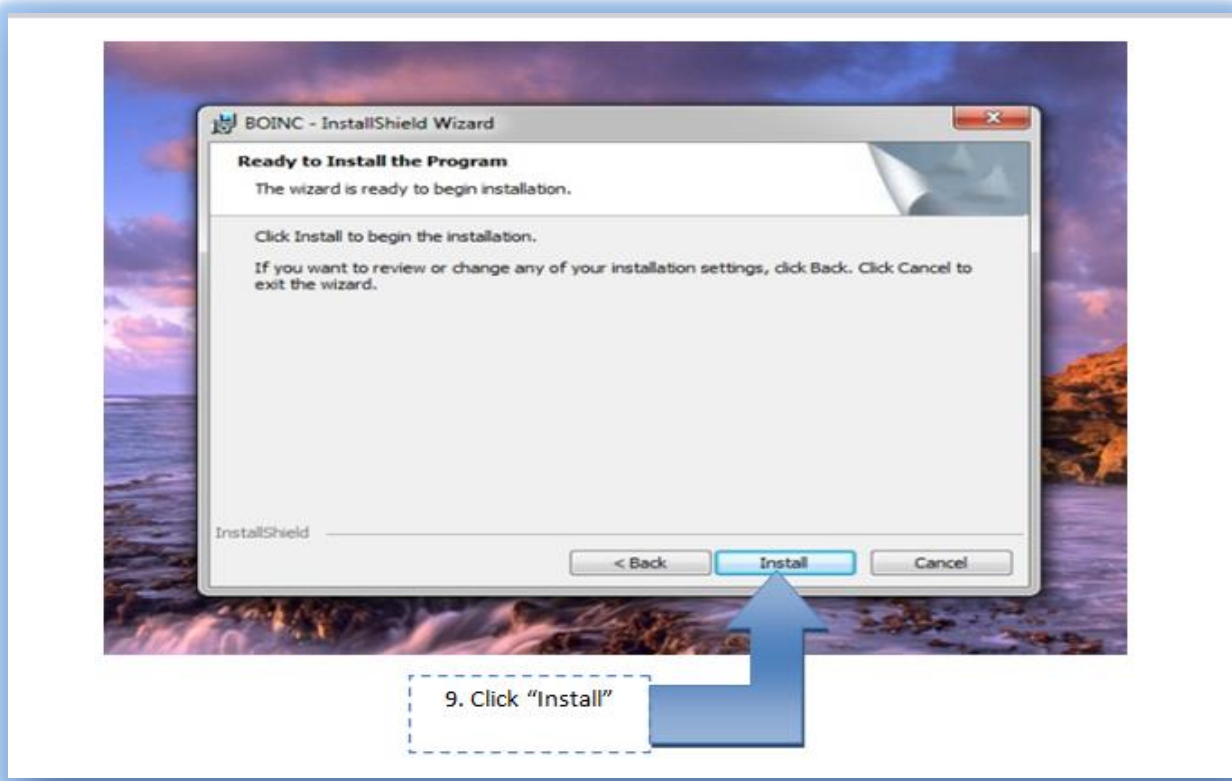


Figure A.4.6: BOINC Client Deployment Step 6

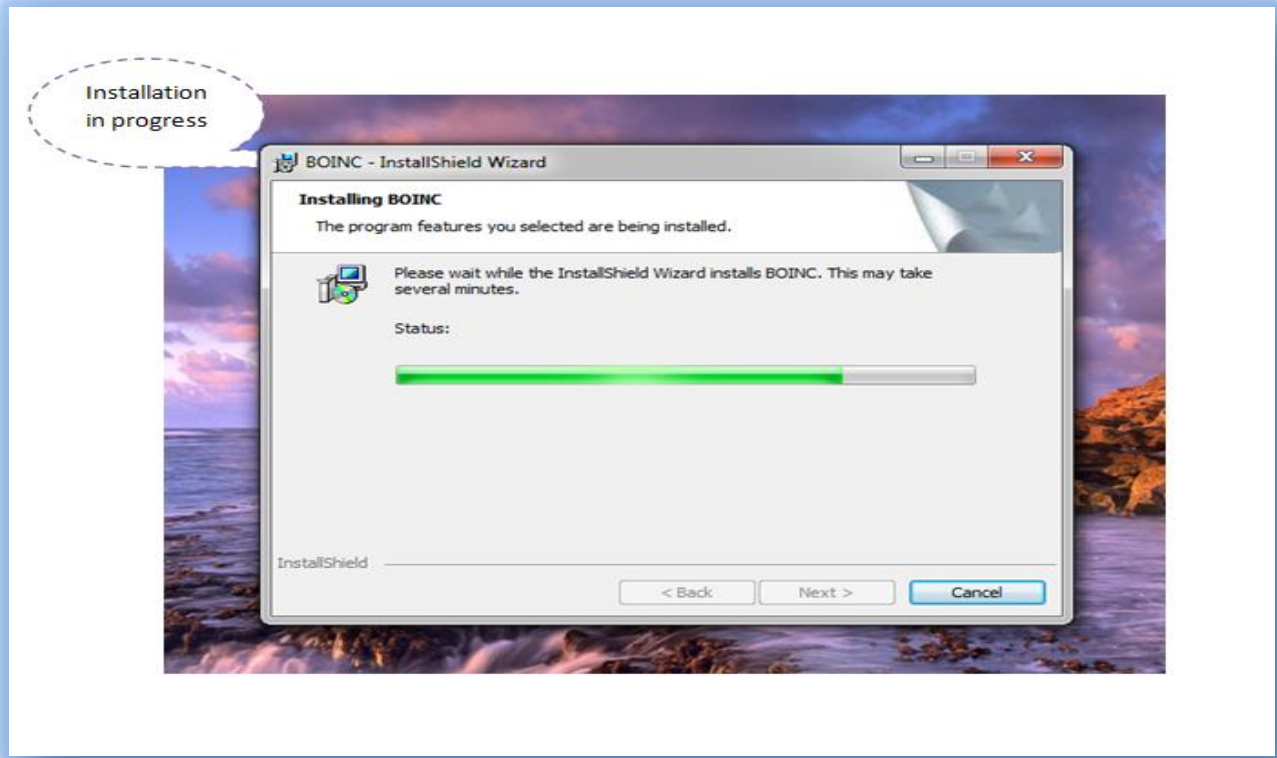


Figure A.4.7: BOINC Client Deployment Step 7

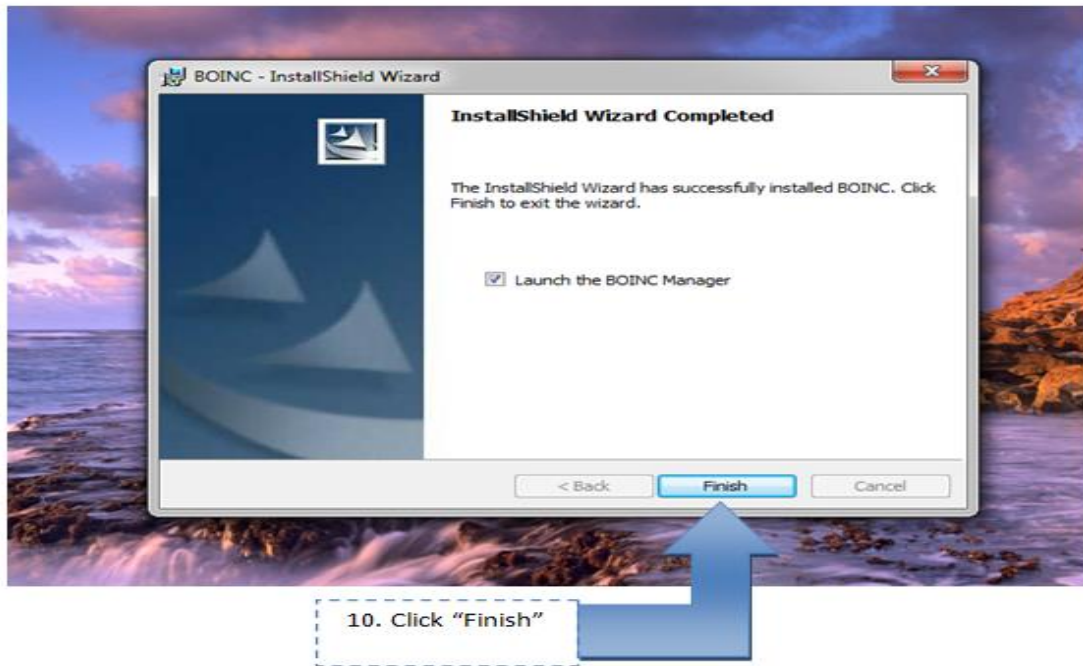


Figure A.4.8: BOINC Client Deployment Step 8

- **Note:**

In step 4 which is shown in Figure A.4.4 we chose to install BOINC client as service by checking the **Service Install** checkbox. In addition, we disabled screen saver option and prevented other users (usually students) from controlling BOINC client software.

A.4.2 Linux

You can install BOINC on a Linux computer in any of three ways [6]:

- Use the package management system of your Linux distribution;
- Use the "Berkeley installer" provided by BOINC (a self-extracting archive, not distro-specific);
- Build directly from source code.

- **Installing BOINC as a package** [6]

Some Linux distributions (Fedora, Ubuntu, Debian, Gentoo, possibly others) have BOINC packages that you can install using your distro's package manager. Compared to using the Berkeley Installer, this has several advantages [6]:

- The resulting BOINC installation runs applications under an unprivileged account, and is therefore more secure.
- The BOINC binaries are dynamically linked; therefore they require less memory than the binaries in the Berkeley Installer.
- The package manager checks for dependencies and installs any additional libraries required to run BOINC on your Linux distro.
- BOINC is installed as a daemon (BOINC runs automatically at boot time even if no user is logged in).
- BOINC updates can be automated if your Linux distro has automated package update capability (most popular distros do)

- **Installing BOINC On Ubuntu**[57]

Ubuntu is a popular distribution of the GNU/Linux operating system. We used Ubuntu as an example of Linux distributions to deploy BOINC client as a package:

- **Basic Installation**

You can easily install the BOINC client software on Ubuntu Linux to run as a daemon, which autostarts the BOINC client at boot time, and to put a BOINC Manager icon on the applications menu. Simply open a Terminal window (via the menu Applications -> Accessories -> Terminal) to get a command line (aka "shell") and give the following command:

```
sudo aptitude install boinc-client boinc-manager
```


- Non-graphics Installation

If you have a "headless" computer with no graphics then you do not want or need the BOINC Manager. In this case just install the client by itself, with the command

```
sudo aptitude install boinc-client
```

A.4.3 Other Platforms

BOINC client is also set to work with other platforms and operation systems other than Microsoft Windows and Linux. Examples of supported platforms are: Mac OS X, FreeBSD and OpenBSD. The process of installing BOINC client on these platforms and other can be found in reference [56].

A.4.4 BOINC Client Security

To make the communication between the boinc core client and a remote computer secure, two files are added to **BOINC data directory** (where BOINC's data files will be stored). These files are:

1. **gui_rpc_auth.cfg file**

This file contains the BOINC client password. Any remote computer wants to communicate with BOINC core client must provide this password in its communication commands.

2. **remote_hosts.cfg file**

This file contains the IPs or DNS names of remote hosts that are allowed to communicate with BOINC core client if they provide the correct password stored in **gui_rpc_auth.cfg file**. Any other host will be prevented from communicating with BOINC core client.

In our case, we used this mechanism to protect our machines and to secure the communication between the client and the server. We added a unified password inside **gui_rpc_auth.cfg** on all of our clients (computers). In addition,

we added only the server IP to **remote_hosts.cfg** file so only the server can communicate with BOINC core clients remotely and any other remote communication is prevented.

- **Boinc data directory**[58]

The files: **gui_rpc_auth.cfg** and **remote_hosts.cfg** need to be placed in the BOINC data directory.

- **Windows XP/2000**

On Windows XP/2000 the BOINC data directory is by default

C:\Documents and Settings\All Users\Application Data\BOINC

This is a hidden directory so if you can't navigate to it via Windows Explorer Folder view, just paste the whole path name, "C:\Documents and Settings\All Users\Application Data\BOINC" into the Windows Explorer address line and it'll jump you there.

- **Windows Vista/7/8/8.1**

On Windows Vista/7/8/8.1 the BOINC data directory is by default

C:\Programdata\BOINC

This is a hidden directory so if you can't navigate to it via Windows Explorer Folder view, just paste the whole path name, "C:\Programdata\BOINC", into the Windows Explorer address line and it will jump you there.

- **Linux**

If you installed the boinc-client package from a package manager in Debian or Ubuntu, client data is stored in /var/lib/boinc-client

- **Mac OS X**

On a Mac, the client data is in: /Library/Application Support/BOINC Data

- **All platforms**

If you are in any doubt, to find the BOINC data directory, go to the client message log. Near the top of the start-up section there will be a line similar to these examples:

```
11/02/2010 9:04:24 AM           Data directory: C:\Documents and Settings\All
Users\Application Data\BOINC
26-Apr-2010 13:08:53 [---] Data directory: /var/lib/boinc-client
```

Appendix B

Project Creation

This appendix describes what the BOINC project is, shows the pre-requirements of the project creation, then it clarifies the project creation process. Finally, it includes a description for installing phpMyAdmin on Ubuntu12.04LTS with solutions for some problems that may arise.

B.1 BOINC Project

BOINC project is built over BOINC middleware. It is used to do distributed computing and/or storage by making the use of the available computing resources. Each project has its own applications, database, web site, and servers.

BOINC project consists of:

- A MySQL database.
- A directory structure.
- A configuration file, which specifies options, daemons, and periodic tasks.

B.1.1 Project DB

Information is stored in a MySQL database. It has the following main tables [59]:

- Platform: A platform is a compilation target (combination of CPU architecture and an operating system). BOINC defines a set of standard platforms.
- App (Applications): An application includes several programs (for different platforms) and a set of work units and results. A project can include multiple applications.
- App_version (application versions): An application program may go through a sequence of versions. A particular version compiled for a particular platform is called an application version.

- User: Describes users, including their email address, name, password, and authenticator.
- Host: Describes hosts.
- Workunit: A workunit is a computation to be performed (Job). It may include any number of input files. It has various attributes, such as resource requirements and deadline.
- Result: A result describes an instance of a computation, either not started, in progress, or completed. Each result is associated with a workunit. In some cases there may be several instances of a given workunit.
- Account: Each volunteer in a project has an account, identified by an email address and password. An account has an associated amount of credit; a numerical measure of the work done by that volunteer's computers. In our case, volunteers are disabled; all hosts (computing resources) are following to the university. So, all computing resources will follow to the same account.

B.1.2 Project Directory

The directory structure for a typical BOINC project looks like shown below [60]:

```
PROJECT/  
  apps/  
  bin/  
  cgi-bin/  
  log_HOSTNAME/  
  pid_HOSTNAME/  
  download/  
  html/  
    inc/  
    ops/  
  project/
```

```
stats/  
user/  
user_profile/  
keys/  
upload/
```

Where: PROJECT is the name of the project.

Main folders at the project directory are:

- apps: application and core client executables.
- bin: server daemons and programs.
- download: storage for data server downloads.
- upload: storage for data server uploads.
- html: PHP files for public and private web interfaces.

B.1.3 Project configuration file

Project configuration is described by a config.xml file exist within the project directory.

The config.xml file has the format[61]:

```
<boinc>  
  <config>  
    [ configuration options ]  
  </config>  
  <daemons>  
    [ list of daemons ]  
  </daemons>  
  <tasks>  
    [ list of periodic tasks ]
```

```
</tasks>
```

```
</boinc>
```

More details about BOINC project can be found at reference [62].

B.2 Project creation pre-requirements

As we know that each project has its own DB. To create and manage the DB we need a MySQL user. This MySQL user must be granted the right privileges that enable him creating and managing DB. This user also must be created before going to the project creation process, since the project DB will be under the control of this user account. In our case we will give this user the name 'boincadm'.

Creating DB user is done by:

- 1- going to mysql through the terminal using the command:
 - Mysql -u root -p
- 2- Creating the user named 'boincadm' by running the mysql terminal command:
 - CREATE USER 'boincadm'@'localhost' IDENTIFIED BY 'Boincadm2014';
- 3- Granting privileges to boincadm:
 - GRANT ALL PRIVILEGES ON * . * TO 'boincadm'@'localhost' IDENTIFIED BY ' Boincadm2014' WITH GRANT OPTION ;
 - Where ' Boincadm2014' is the password for boincadm.

B.3. Project Creation Process

This section talks about creating a BOINC project. Firstly, it describes creating empty project (project doesn't have any application). Secondly, it describes creating BOINC project having the test application example.

B.3.1 Creating an Empty BOINC Project

We need to set some parameters to be used during the project creation process. These parameters are:

- PPUTest: The name of the project.
- boincadm: the name of DB user; the owner of project DB. Also it is the name of the system user; the owner of the project directory.
- Boincadm2014: the password for boincadm DB user and the password for the system user.
- 195.3.191.24: is the IP address of the server.

Project creation process:

Project creation goes into the following steps:

1. Creating project directory and DB, setting the default configurations and deleting any previous project having the same name. This is done by executing the commands:

- `cd boinc/tools`
- `sudo ./make_project --url_base http://195.3.191.24 --db_name PPUTest --db_user boincadm --delete_prev_inst --drop_db_first --db_passwd Boincadm2014 --project_root home/boincadm/projects/PPUTest --srmdir home/boincadm/boinc/ PPUTest "PPU Test"`

2. Some specific files in the project directory need custom setting to their permissions to allow the project works well. These permissions are set by executing the commands:

- `cd home/boincadm/projects/PPUTest`
- `sudo -S chown boincadm:boincadm -R . <<<Boincadm2014`
- `sudo chmod g+w -R .`
- `sudo chmod 02770 -R upload`
- `sudo chmod 02770 -R html/cache`

- `sudo chmod 02770 -R html/inc`
- `sudo chmod 02770 -R html/languages`
- `sudo chmod 02770 -R html/languages/compiled`
- `sudo chmod 02770 -R html/user_profile`
- `sudo chgrp -R www-data log_ubuntu upload`
- `sudo chmod o+x html/inc`
- `sudo chmod -R o+r html/inc`
- `sudo chmod o+x html/languages/`
- `sudo chmod o+x html/languages/compiled`

More information about setting permissions can be found at reference [63].

3. Appending the project http.conf file to the apache server http.conf file to allow browsing the project web site:

- Open the file `home/boincadm/projects/PPUTest/html/http.conf`.
- Copy its content.
- Open the file `/etc/apache2/http.conf`.
- Paste that content there and save changes. Note that you need to open the file as administrator to be able to save changes.
- Restart apache server to make the changes take effect.
 - You can use the command: `sudo service apache2 restart`

After doing these three steps, we can open the link 195.3.191.24/PPUTest/ to see the project home page as shown in figure B.1.

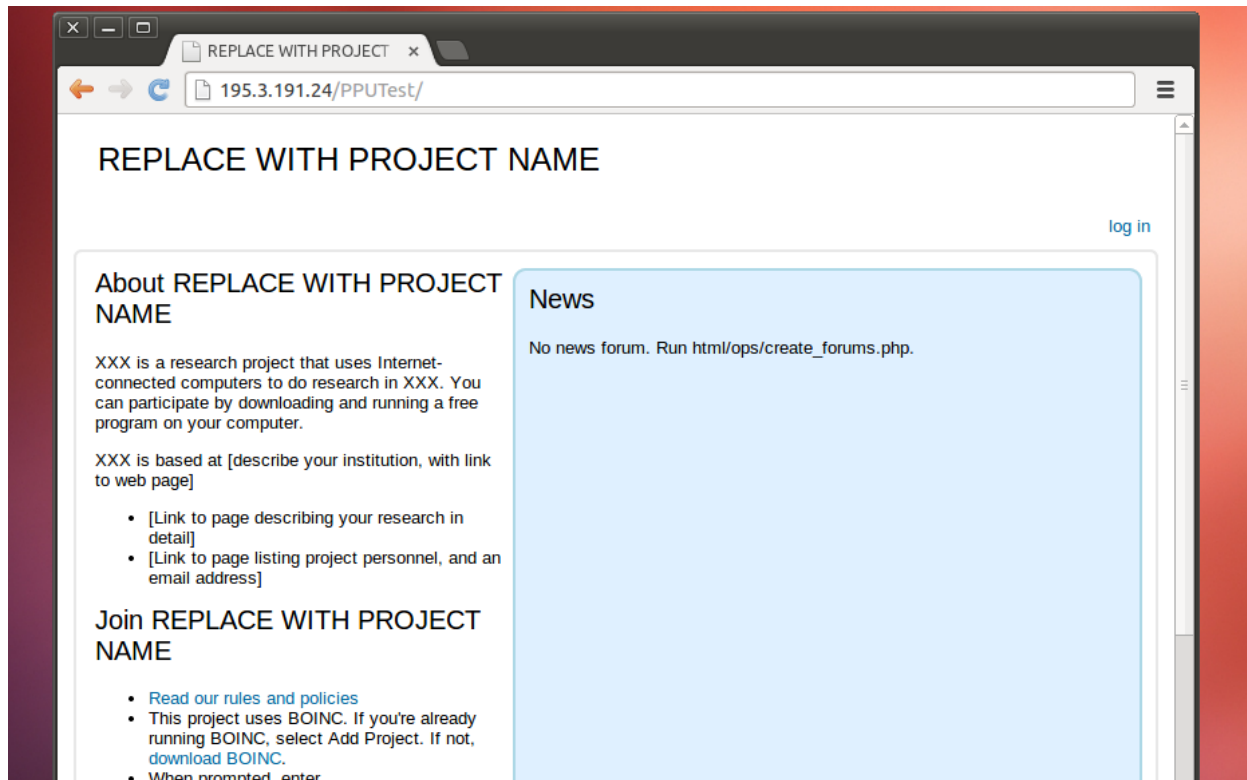


Figure B.1 Project home page.

4. Setting the project name to appear in project url:

- Open the file
home/boincadm/projects/PPUTest/html/project/project.inc
- Change the statements "REPLACE WITH PROJECT NAME" with "PPUTest" and save changes.

Before the changes the home page looks like shown in figure B.2:

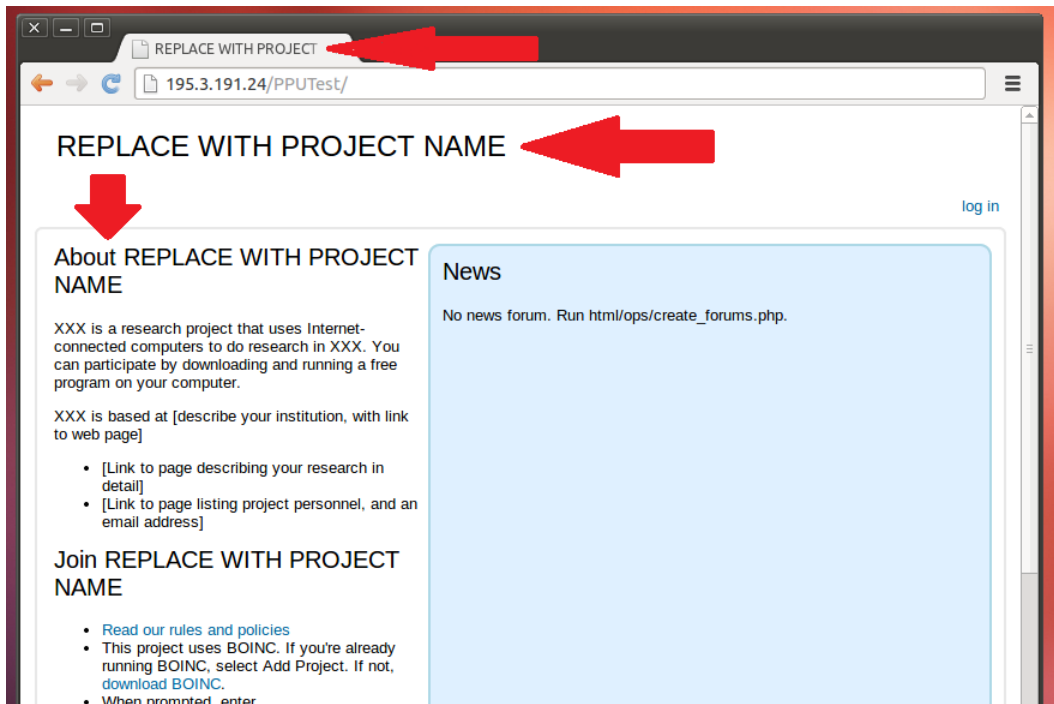


Figure B.2: Adding project name 1.

After the changes you can see the difference as shown in figure B.3 below:

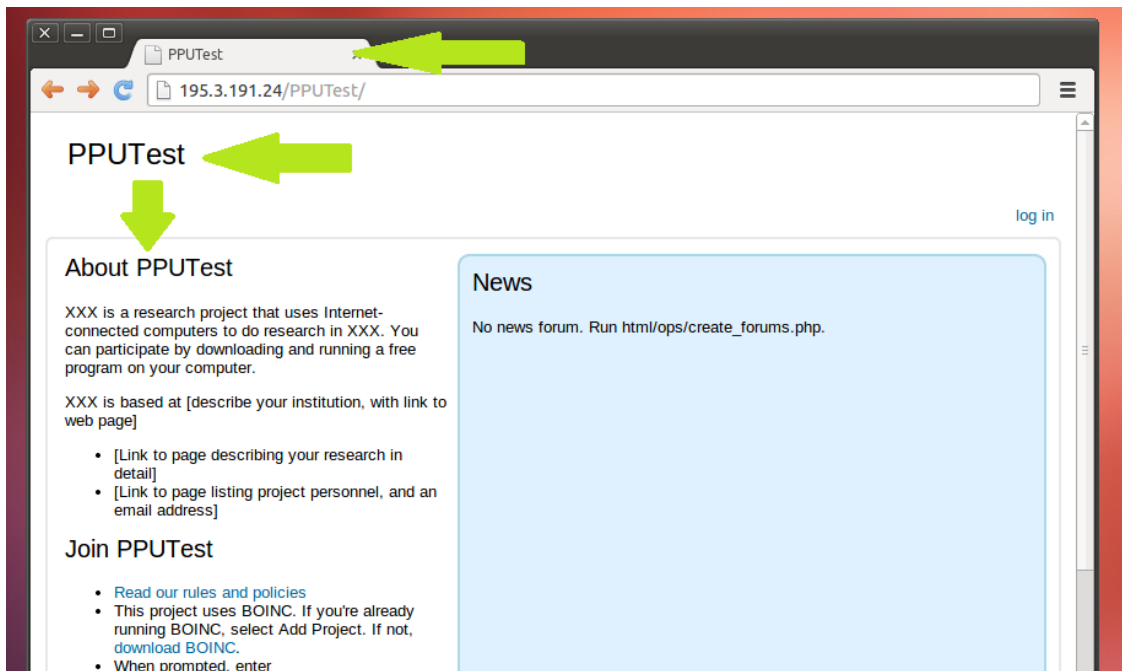


Figure B.3: Adding project name 2.

5. Setting the copy right holder in project URL:

- Open the file
home/boincadm/projects/PPUTest/html/project/project.inc
- Change the statement "REPLACE WITH COPYRIGHT HOLDER" with "PPU Grid Team: Ibrahim Qdemat and Muhammad Dwaib" and save changes.

Before these changes, the page will be as shown in figure B.4.

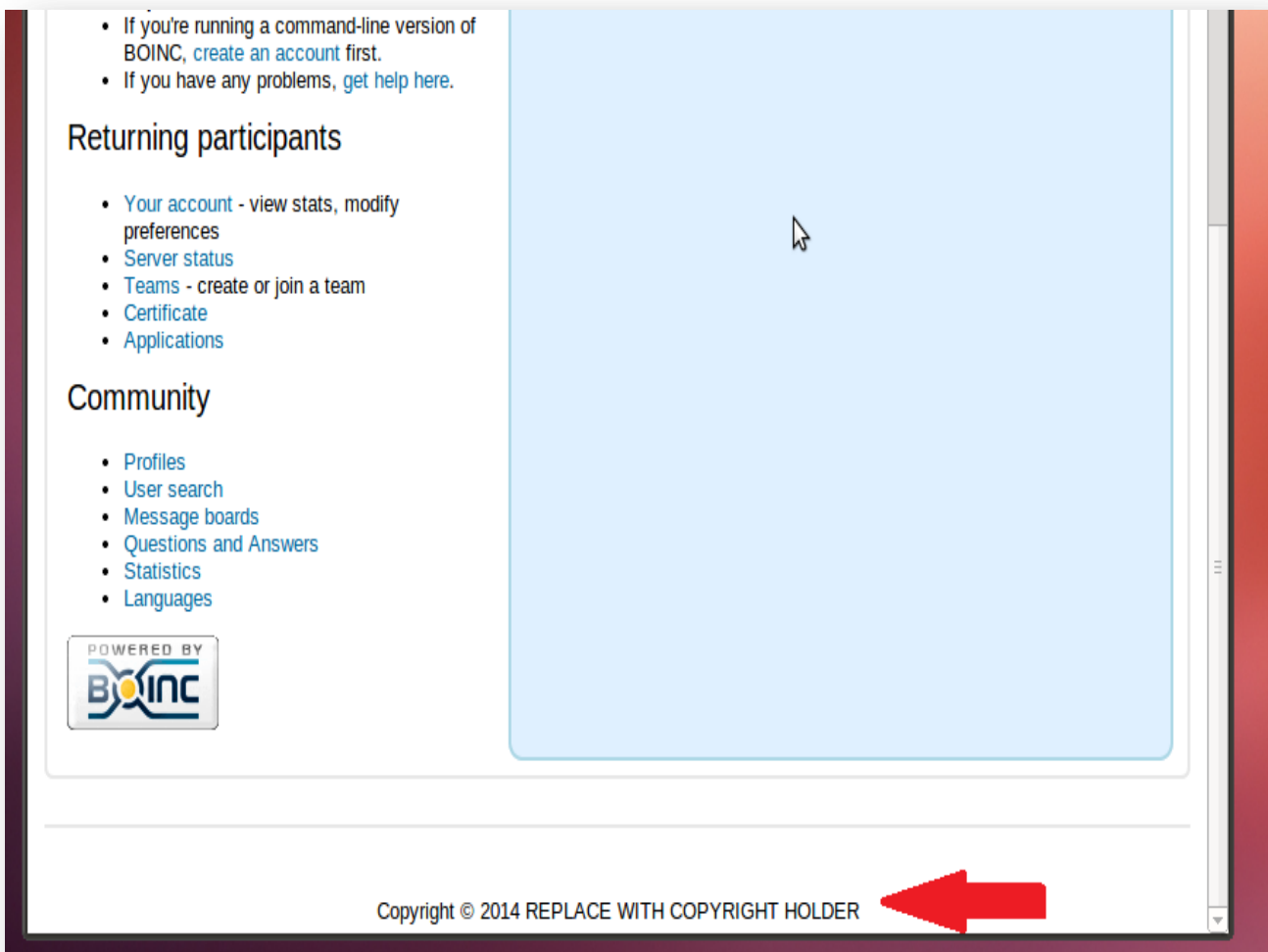


Figure B.4: Adding copy rights 1.

After the changes are done, you can see the difference as shown in figure B.5

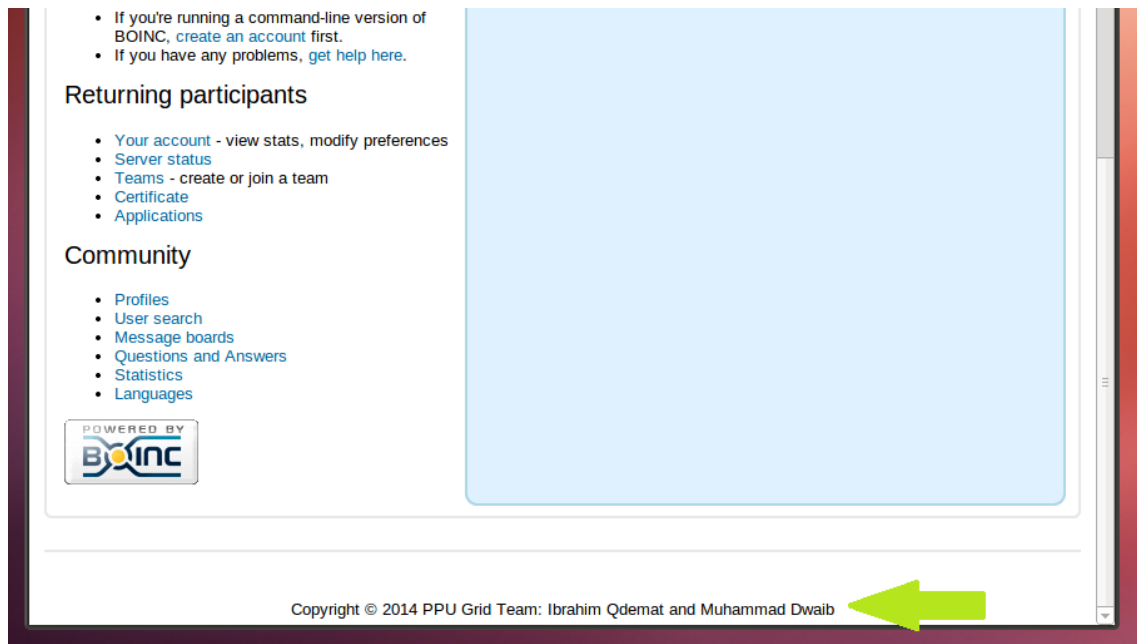


Figure B.5: Adding copy rights 2.

6. Setting password for project administrative webpage.

This can be done by executing the commands:

- `cd home/boincadm/projects/PPUTest/html/ops/`
- `htpasswd -b -c .htpasswd boincadm Boincadm2014`
 - Where: boincadm is the admin username and Boincadm2014 is the admin password.

Before doing this step you will not be able to open the project administrative page 195.3.191.24/PPUTest_ops/. It will ask you for a login username and password but they are not set yet. So, the administrative page will just show a message as in the figure B.7.

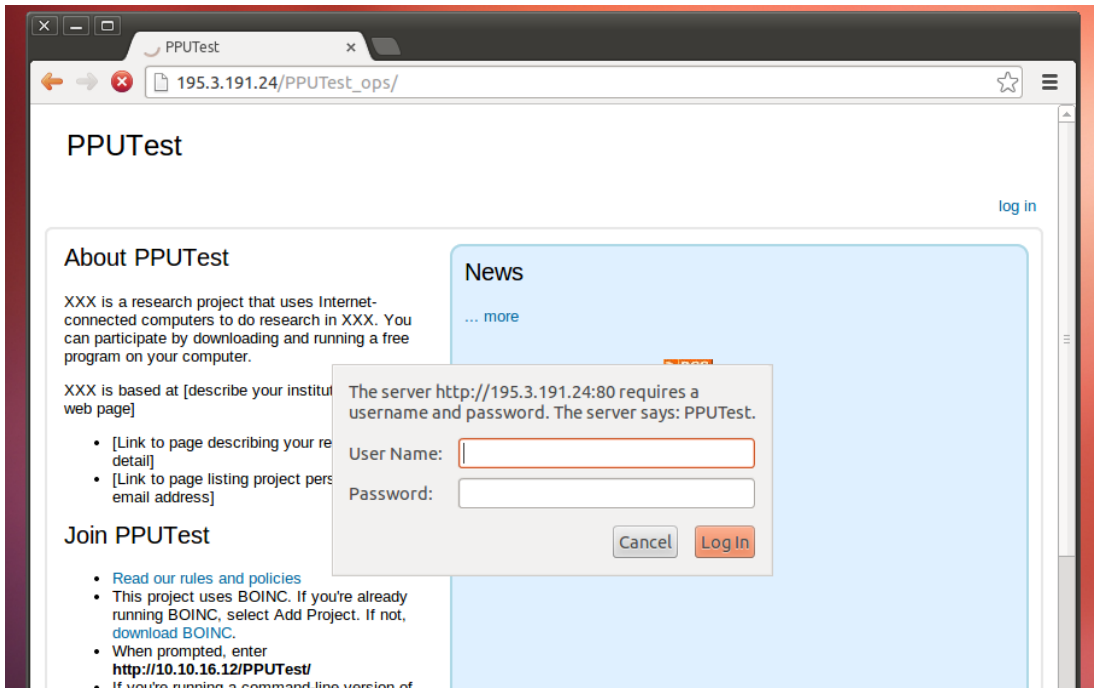


Figure B.6: Setting admin. Account 1.

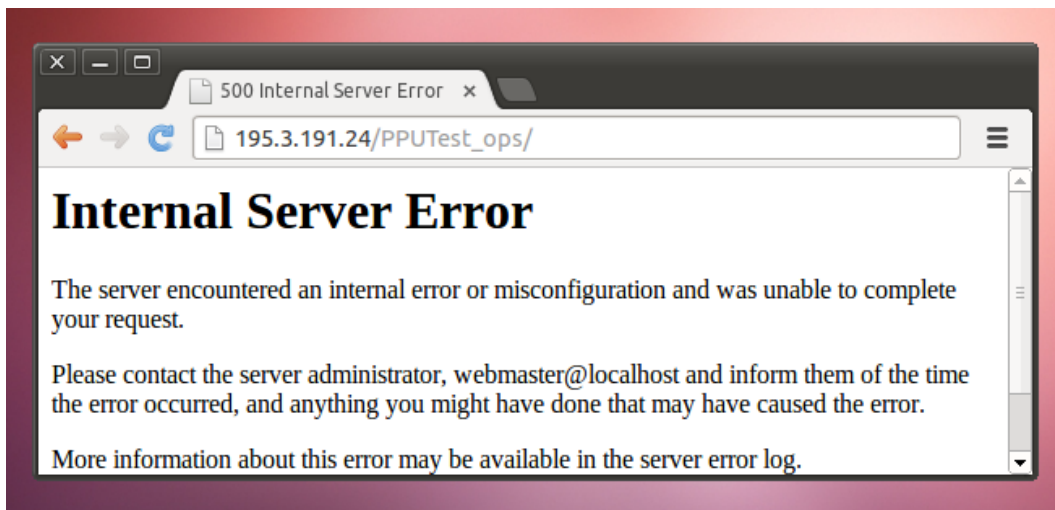


Figure B.7: Invalid login to admin page.

After setting the administrator username and password, you can point the browser to 195.3.191.24/PPUTest_ops/. Enter the username 'boincadm' and the password 'Boincadm2014', then you will see the administrative page shown in figure B.9.

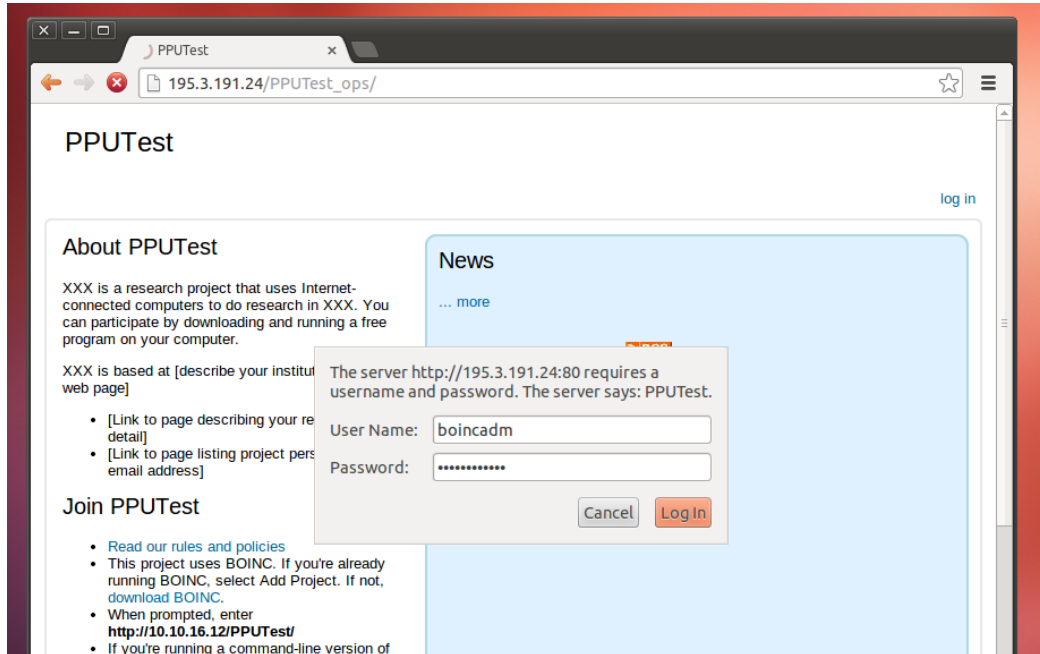


Figure B.8: Setting admin Account 2.

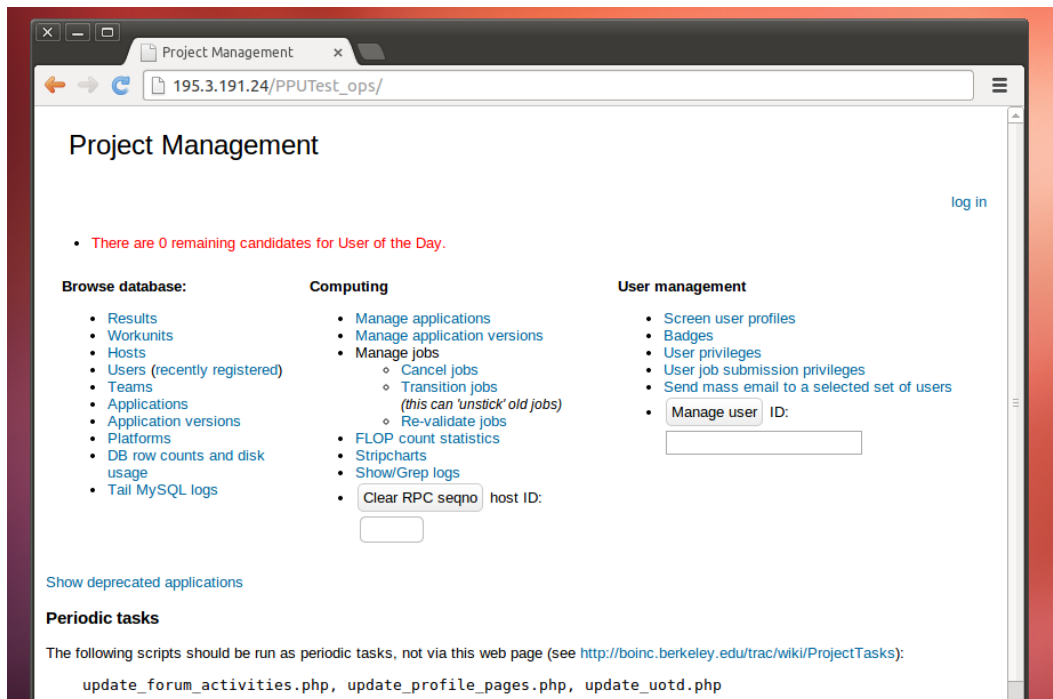


Figure B.9: Administrative page.

7. By default the example application provided by BOINC will be added to the project, so you need to remove this application as follow:

- Open the file `home/boincadm/projects/PPUTest/project.xml`.
- Remove the lines that define example application from this file and save changes. Those lines are:
 - `<app>`
 - `<name>example_app</name>`
 - `<user_friendly_name>Example Application</user_friendly_name>`
 - `</app>`

8. Starting PPUTest project daemons is done by executing:

- `cd home/boincadm/projects/PPUTest`
- `./bin/start`

Before doing this step, the project status page http://195.3.19.124/PPUTest/server_status.php will look like shown in figure B.10

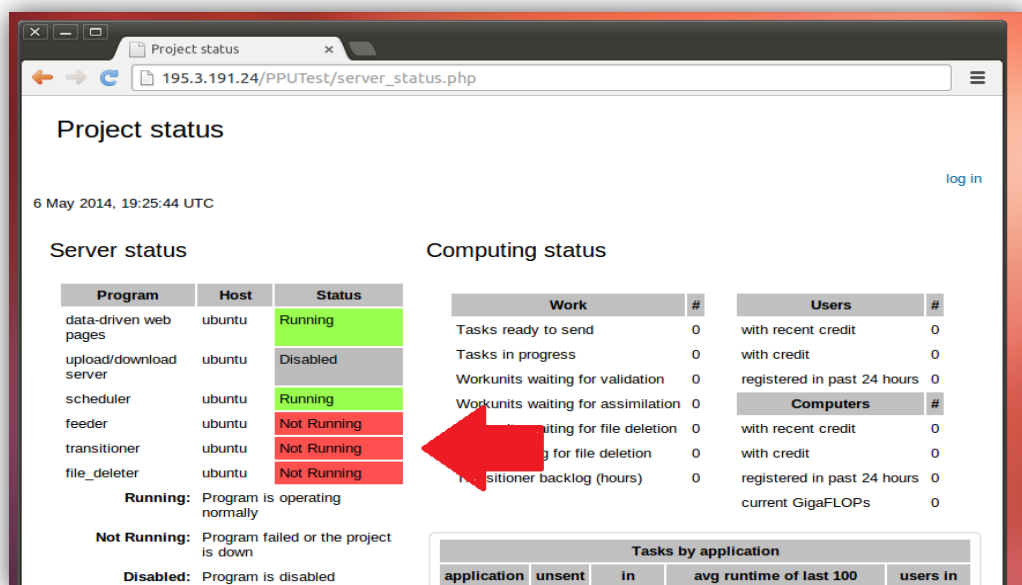


Figure B.10: Project status 1.

After starting daemons the project status page will appear as shown below:

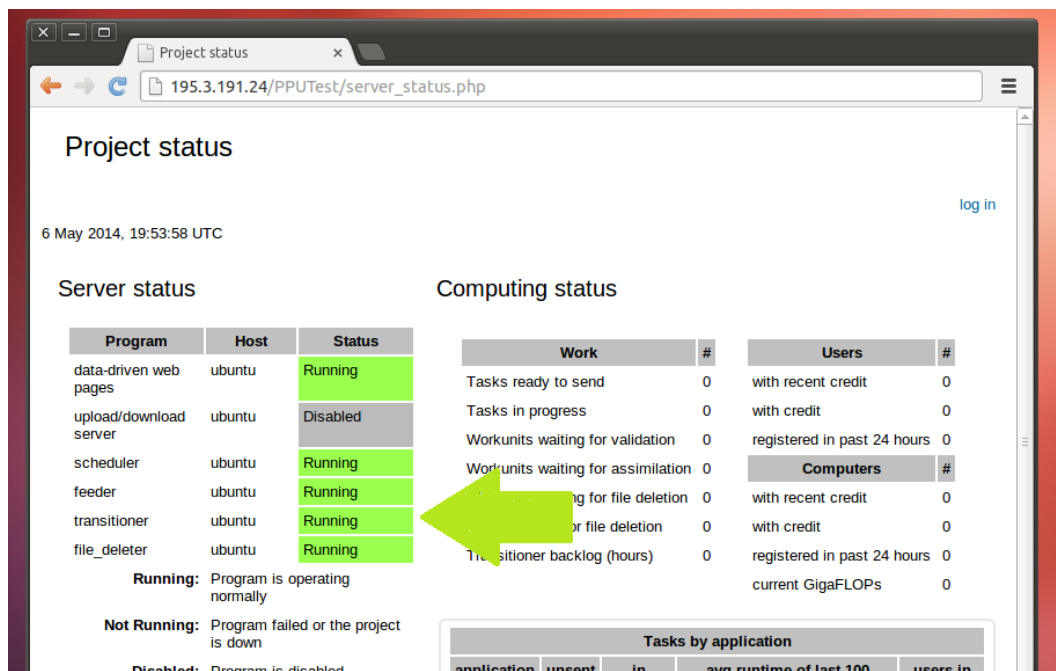


Figure B.11: Project status 2.

9. If the path of pid file of upload/download server is not determined in project's config.xml as following: `<uldl_pid>path</uldl_pid>`, then the default which is `/etc/httpd/run/httpd.pid` will be used. In our case: the pid file of upload/download server is `/var/run/apache2.pid`. To resolve this problem do the following:

- Open the file `home/boincadm/projects/PPUTest/config.xml`
- Copy the line `"<uldl_pid>/var/run/apache2.pid</uldl_pid>"` and paste it within the `"<config>...</config>"` tag
- Save changes to enable upload/download.

Before:

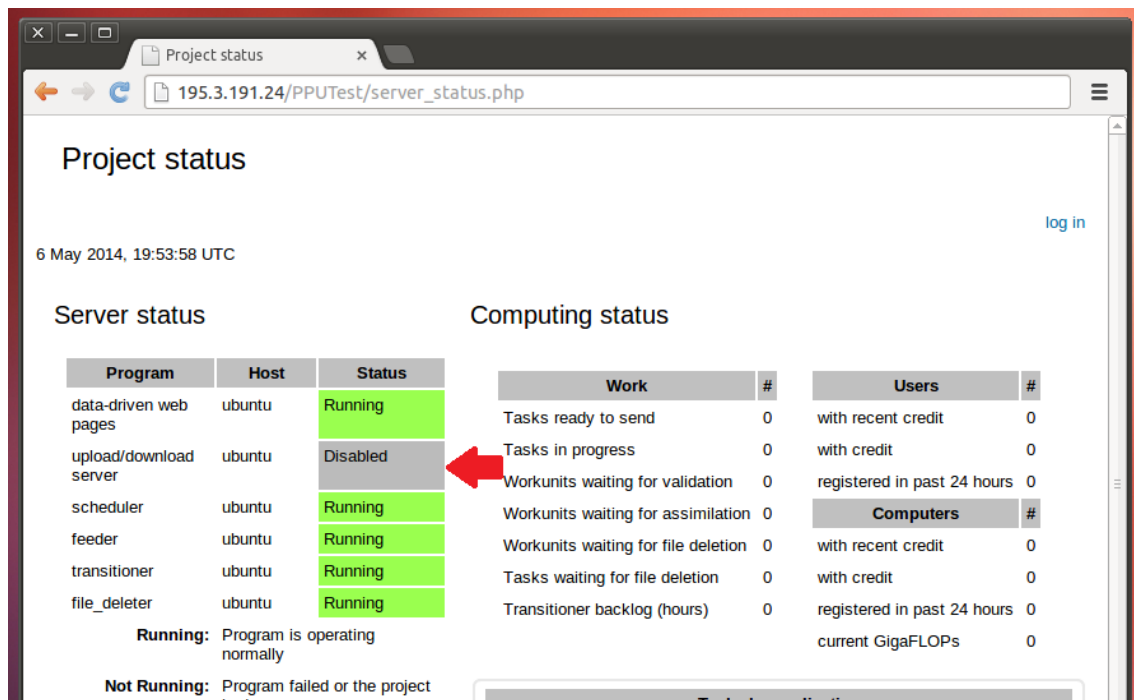


Figure B.12: Project status 3.

After:

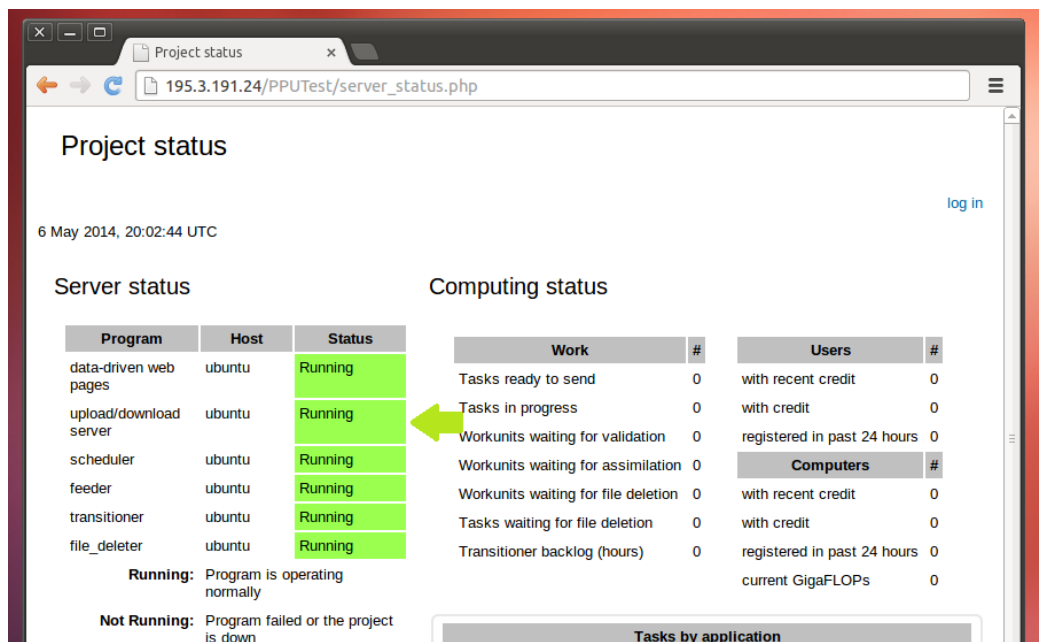


Figure B.13: Project status 4.

10. Solving “ the antique file deleter is not working” problem:

This problem can be found at the antique_file_deleter.out log file which has the path '/home/boincadm/projects/log_ubuntu/test/antique_file_deleter.out'.

The problem appears as follows:

```
2014-03-26 11:45:02.7498 Starting
2014-03-26 11:45:02.7545 [CRITICAL] Couldn't find http_user 'apache' in passwd
2014-03-26 11:45:02.7545 [CRITICAL] delete_antiques() returned with error -1
2014-03-26 11:45:02.7545 Done
```

File deleter problem is caused by the BOINC default configurations set the user of the web-server to be 'apache' but in Ubuntu the default web-server user is 'www-data'.

In order to solve this problem, do the following:

- Open the file home/boincadm/projects/PPUTest/config.xml
- Add the line `<httpd_user>www-data</httpd_user>` within the `<config></config>` tag and save changes.

11. Enabling the project forum:

- Open the file
home/boincadm/projects/PPUTest/html/ops/create_forums.php
- Delete the line starting with “**die();**”
- Run create_forums.php. You can do that by execute the commands:
 - `cd home/boincadm/projects/PPUTest /html/ops/`
 - `php5 create_forums.php`

Before:

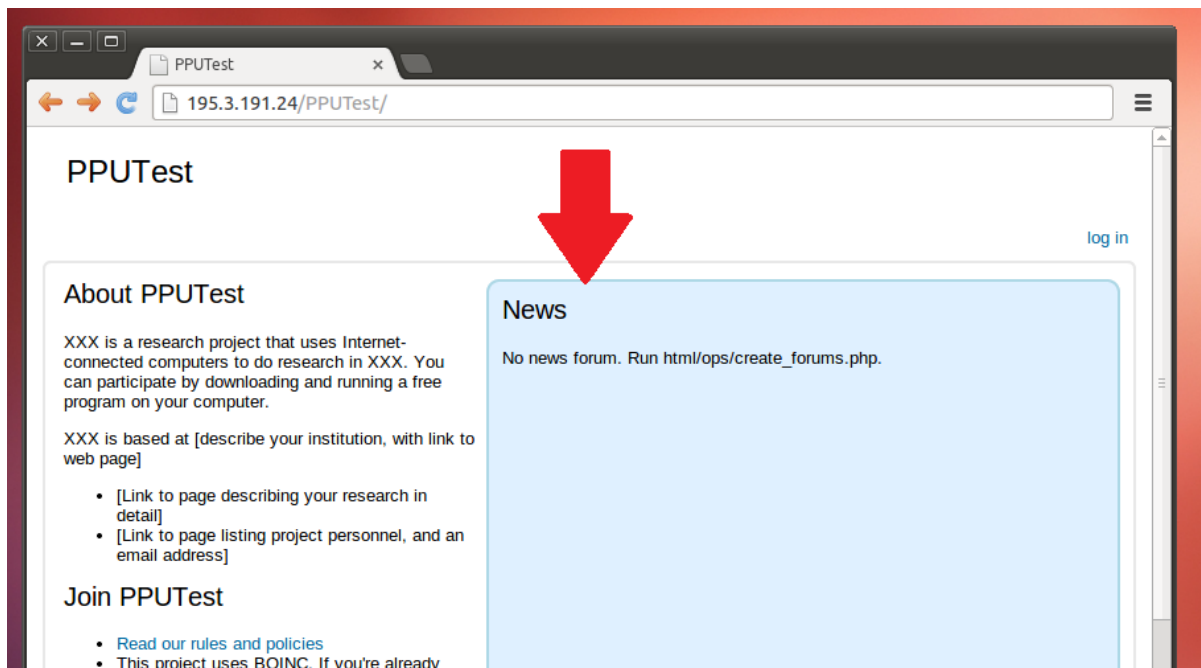


Figure B.14: Project forum 1.

After:

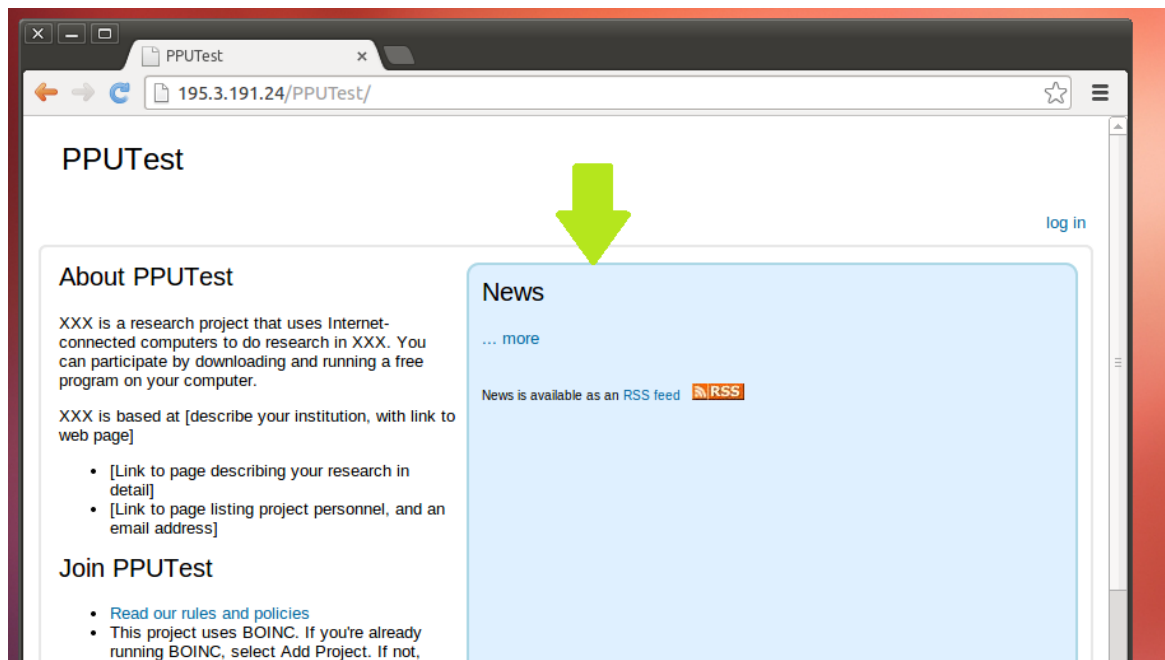


Figure B.15: Project forum 2.

Now the project is running and ready for adding applications. As it can be inferred from the previous steps, the process of creating BOINC project is not an easy task. In addition to the long steps must be followed, there are a lot of problems and bugs that need to be handled and solved. In order to simplify this process, we created a shell script named **creatProject.sh**. This script is responsible for performing all the steps of creating BOINC project and solving the problems that appear during the project creation process.

B.3.2 Creating a project with a test application example

Test application example is an example single-thread native BOINC application [38]. It is used to perform tests for environments that use the BOINC as a middleware. This application has application versions that run on computers with different well known platforms. The example application reads an input file, converts the file to upper case and writes it to output file.

To create a BOINC project running the test application example you can follow the same scenario with creating empty BOINC project with some changes stated below:

1. Replace the second command at step 1 to include the example application at the project creation. The command will be as follow:

```
sudo ./make_project --url_base http://195.3.191.24/ --test_app PPUTest --db_name PPUTest --db_user boincadm --delete_prev_inst --drop_db_first --db_passwd Boincadm2014 --project_root home/boincadm/projects/PPUTest --srcdir home/boincadm/boinc/ PPUTest "PPU Test"
```

2. Follow the same steps until arriving to step 7. Now you need to replace this step with the following one:

Preparing the application example for running by executing the commands:

- `cd home/boincadm/project/PPUTest/`
- `crontab PPUTest.cronjob`
 - Make the tasks running periodically.
- `./bin/xadd`

- Adds the application example.
- `./bin/update_versions`
 - Adds the application versions.
 - You need to answer yes to all questions.

Follow all remaining steps without changes.

You can see the difference between the empty project and this one directly at the page http://195.3.191.24/PPUTest/server_status.php. Additional three daemons were added. See figure below:

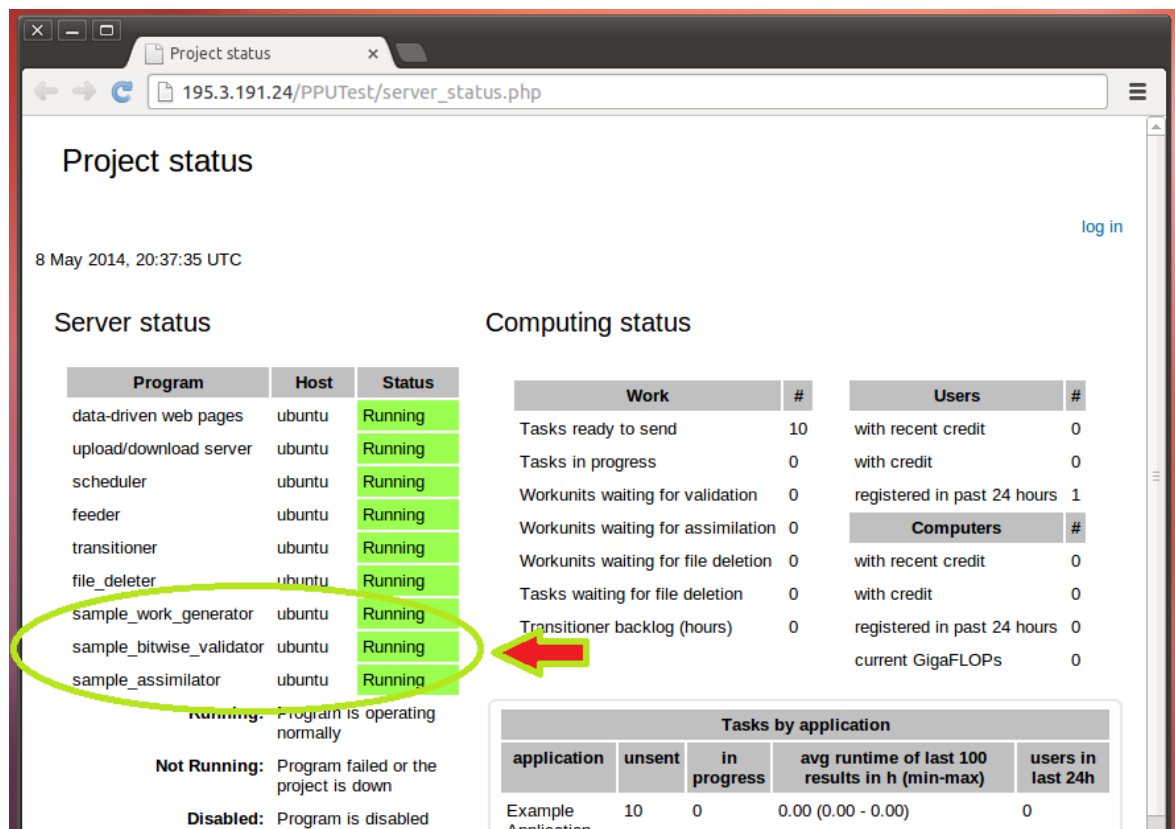


Figure B.16: Test example application project status.

Since it is not an easy task to go through all of the previous steps to create a test application project; we created another shell script named `createTestProject.sh`. This script creates the test application project taking in consideration all of previous steps.

The **createTestProject.sh** script was built to work on Ubuntu Linux distribution by making use of a similar script that was made for Debian Linux distribution. The similar script is called `boinc_project_maker` and can be found at reference [64].

B.4 phpMyAdmin Installation

phpMyAdmin provides a GUI interfaces for **MySQL** databases. You can use it to simplify monitoring and controlling **MySQL** databases and reduce the need for writing SQL queries as a terminal commands.

To get phpMyAdmin running on Ubuntu follow the steps below:

1. Install the phpMyAdmin:
 - `sudo apt-get install phpMyAdmin`
2. Include the phpMyAdmin configurations by appending the file `/etc/apache2/apache2.conf` with the following line:
 - Include `/etc/phpmyadmin/apache.conf`

Once previous steps are done, a user can point his browser to 195.3.191.24/phpmyadmin (in our case) to start using phpMyAdmin. A user should be able to login using the account created in MySQL (see section B.2). phpMyAdmin login page is shown below in **Figure B.17**.

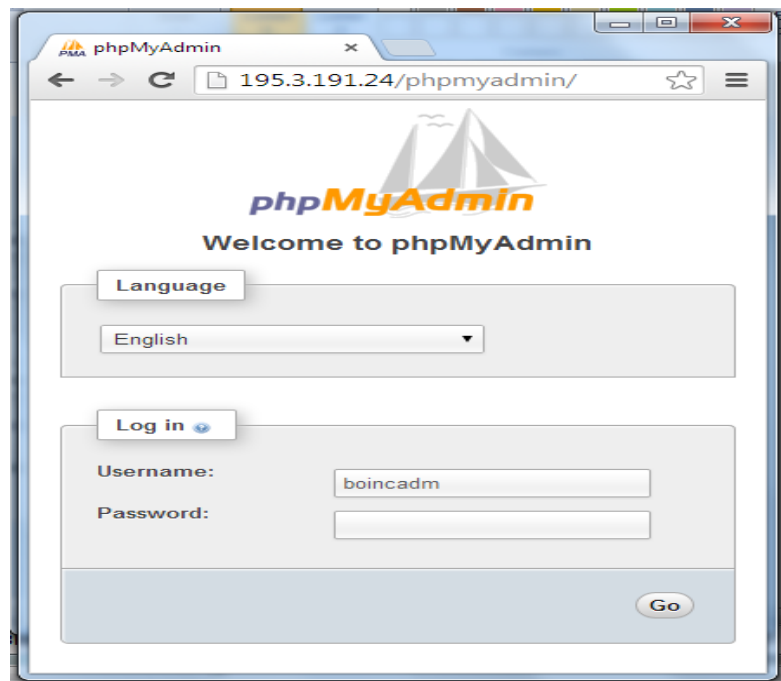


Figure B.17: phpMyAdmin login

Some problems may arise after installing phpMyAdmin, some of them are discussed below:

1. Getting a 404 "Not Found" error when pointing the browser to the location of phpMyAdmin. The issue is likely caused by either:
 - Not checking the 'Apache 2' selection during installation. To handle this problem run the following commands:
 - a. `sudo dpkg-reconfigure -plow phpMyAdmin`
 - b. Then select Apache 2 for the webserver you wish to configure.
 - Or if this does not work, then you can do the following to include the phpMyAdmin-shipped Apache configuration into Apache:
 - a. `sudo ln -s /etc/phpmyadmin/apache.conf /etc/apache2/conf.d/phpmyadmin.conf`
 - b. `sudo /etc/init.d/apache2 reload`
2. Sometimes at the development stage, one may set the root user without a password, this may cause another problem. This problem appears when you attempt to login to the root user:

“Login without a password is forbidden by configuration”.

The solution of this problem is done by uncommenting or adding the following line (if it is not exist) to the file `/etc/phpmyadmin/config.inc.php`:

- `$cfg['Servers'][$i]['AllowNoPassword'] = TRUE;`

Note that you need to open this file as administrator to be able to save changes. Also you have to make sure to retrieve these configurations and setting a password for root user before deploying the system.

Appendix C

Security

In this appendix, we show the technical details of enabling different security mechanisms supported by BOINC. In addition, we show our technical security measures to make our grid system project secure.

C.1 Introduction

BOINC was originally designed for volunteer computing. Volunteer computing has a lot of security issues that need to be handled properly to ensure security. Because of this, BOINC provides security mechanisms that address the major issues, making volunteer computing safe.

Although it was originally designed for volunteer computing, BOINC works very well for grid computing. So as we build our grid system using BOINC middleware, we can make use of its rich security mechanisms to make our grid system more safe and secure.

If we do not use these mechanisms correctly, our projects will be vulnerable to a variety of attacks. In the worst case, these projects could be used as a vector to distribute malicious software to large numbers of computers.

In our grid system, the computing resources are under our control so they can be trusted. In other words, we can assume that the PCs do not return results that are intentionally wrong. Hence there is typically no need for replication.

We deal with the security issues from different sides, securing the server and the clients' machines, authentication between the client and the server and securing the

software components. If we deal with all of these security issues properly, we end up with a secure system as a whole.

C.2 Protecting Administrative web interface

Each BOINC project has an administrative web interface. If a project's URL is for example **http://a.b.c.d/test**, then the URL of the admin web interface is **http://a.b.c.d/test_ops**. The directory containing the admin pages is **~/projects/test/html/ops/**. Because the admin interface lets you do things like see user email addresses, it's extremely important that it be secure. There are two levels of protection [40]:

C.2 .1 Protection by **.htaccess**

The “**.htaccess**” file is a configuration file for use on web servers running the Apache Web Server software. The original purpose of **.htaccess** file was to allow per-directory access control, by for example requiring a password to access the content.

When a project is created, a file **html/ops/.htaccess** is created that disallows access to the admin web interface. You can use **htpasswd** to create a **.htpasswd** file containing credentials for yourself:

```
htpasswd -c .htpasswd username
```

The previous line code creates a hidden password file named “.htpasswd” for user “username” and prompts the user to enter the password. The password is hashed and stored in “.htpasswd” file. When you want to visit the administrative webpage, you must enter the username and the password entered in the previous step.

The following chunk of code is an example of setting the administrative webpage password in our grid system,

```
#setting password for project administrative webpage
installroot=/home/ibrahim/projects
fileprojectname=PPU_Project
#Project administrative webpage password
adminWebPagePasswd=boincadm
#name of database user (project admin)
dbuser=boincadm
cd "$installroot"/"$fileprojectname"/html/ops/
htpasswd -b -c .htpasswd $dbuser $adminWebPagePasswd
```

C.2 .2 Project-defined protection policy

The config file **html/project/project.inc** can specify a function **auth_ops()** that defines a project-specific policy for protecting the admin interface. Possible policies:

- Access only if logged in as user from a given list.
- Access only to users with ADMIN or DEV flag set in `forum_preferences.privileges`.
- Access only from specific IP addresses.
- Any other policy you can think of.

Some examples are given in the function **auth_ops_example()** in the default config file.

C.3 Other Techniques

In addition to previous security measures, there are other techniques that can be used to ensure more security. Code signing and secure socket layer two examples of such techniques.

C.3.1 Code Signing

The Code Signing technique ensures the integrity of the code downloaded from the Internet. It enables the platform to verify that the code has not been modified since it was signed by its creator. Code Signing cannot reveal what the code can do or guarantee that the code is in fact safe to run [41].

BOINC uses digital signatures to allow the core client to authenticate executable files. It is important that we use a proper code-signing procedure for publicly-accessible projects. If we don't, and our server is broken into, hackers will be able to use our BOINC project to distribute malware. This could result in the end of the project, and will negatively impact all BOINC projects.

BOINC advises to follow a given procedure to properly use code signing and provides some file signing utilities. We explain the BOINC advisable code signing procedure in addition to technical details of code signing utilities in appendix C.

BOINC uses digital signatures to allow the core client to authenticate executable files. It is important that we use a proper code-signing procedure for publicly-accessible projects. If we don't, and our server is broken into, hackers will be able to use our BOINC project to distribute malware. This could result in the end of our project, and will negatively impact all BOINC

projects. BOINC advises to follow the following procedure to properly use code signing [65]:

- Choose a computer (an old, slow one is fine) to act as your "code signing machine". After being set up, this computer must remain physically secure and disconnected from the network (i.e. keep it in a locked room and put duct tape over its Ethernet port). You'll need a mechanism for moving files to and from the code-signing machine, such as a USB memory stick.
- Install `crypt_prog`(see appendix D) on the code signing machine (it's easiest if the machine runs Linux or Mac OS X; Windows can be used but requires Visual Studio 2005).
- Run `crypt_prog -genkey` to create a code-signing key pair. Copy the public key to your server. Keep the private key on the code-signing machine, make a permanent, secure copy of the key pair (e.g. on a CD-ROM that you keep locked up), and delete all other copies of the private key.
- To sign an executable file, move it to the code-signing machine, run `crypt_prog -sign` to produce the signature file, then move the signature file to your server.
- Use `update_versions` to install your application, including its signature files, in the download directory and database.

- **File signing utilities** [66]

Use **sign_executable** to sign executable files:

```
sign_executable file_to_sign private_key_file >
signature_file
```

`sign_executable` is compiled in the `lib/` directory, and installed in your project's `bin/` directory. It writes the signature to stdout.

- **Creating encryption keys** [66]

The program `lib/crypt_prog` performs various encryption tasks.

crypt_prog -genkey nbits private_keyfile public_keyfile

Create a key pair with `nbits` bits (always use 1024). Write the keys in encoded ASCII form to the indicated files.

The following commands generate the file upload and code signing key pairs. `BOINC_KEY_DIR` is the directory where the keys will be stored. The code signing private key should be stored only on a highly secure (e.g., a disconnected, physically secure) host.

```
crypt_prog -genkey 1024 BOINC_KEY_DIR/upload_private
BOINC_KEY_DIR/upload_public

crypt_prog -genkey 1024 BOINC_KEY_DIR/code_sign_private
BOINC_KEY_DIR/code_sign_public
```

Other functions of `crypt_prog`:

crypt_prog -sign file private_keyfile

Create a digital signature for the given file (same as `sign_executable`).

crypt_prog -sign_string string private_keyfile

Create a digital signature for the given string, write it to stdout.

crypt_prog -verify file signature_file public_keyfile

Verify a signature for the given file.

crypt_prog -test_crypt private_keyfile public_keyfile

Perform an internal test, checking that encryption followed by decryption works.

crypt_prog -cert_verify file signature_file certificate_dir ca_dir

Verify a certificate-based signature for the given file.

crypt_prog -convsig o2b/b2o input_file output_file

Convert a signature from OpenSSL form to/from BOINC form.

crypt_prog -convkey o2b/b2o priv/pub input_file output_file

Convert a key from OpenSSL form to/from BOINC form.

C.3.2 Secure Socket Layer (SSL)

BOINC supports and encourages enabling SSL on project's web servers. To use SSL, we will need to buy an SSL certificate. Self-signed certificates can't be used. To enable SSL will then need to configure BOINC projects and change our Apache configuration as described below [42]:

- **BOINC configuration**

For each project, add the following line to **html/project/project.inc** file:

```
define("SECURE_URL_BASE", "https://your_url/");
```

where the URL is that of our HTTPS server (typically our project's master URL with "<https://>" at the start).

- **Apache configuration**

Use the Linux "wget" program to test your HTTPS; it uses libcurl, same as the BOINC client. If you use virtual hosts your Apache config file will need an entry like the following:

```
<VirtualHost *:443>

ServerName setiathome.berkeley.edu

DocumentRoot ... path to your /html/user

SSLEngine On

SSLCertificateFile
/etc/pki/tls/certs/setiathome.berkeley.edu.SAN.cert

SSLCertificateKeyFile
/etc/pki/tls/private/setiathome.berkeley.edu.SAN.key

SSLCertificateChainFile /etc/httpd/conf/ssl.crt/in_common.crt

</VirtualHost>
```

If we do this, and follow the previous instructions, the following communication will be protected from man-in-the-middle attacks [42]:

- The web RPCs used for account creation, which carry volunteer email addresses.
- HTTP requests that carry volunteer email addresses and passwords, such as the login form.

If, in addition, we use HTTPS for our scheduler URLs, scheduler requests (which carry account authenticators, which can be used to log in to accounts) will be encrypted.

Appendix D [68]

Boinccmd tool

The BOINC command tool (**boinccmd**) provides a command-line interface to a running **BOINC client** (local or remote). This provides an alternative to the BOINC Manager, e.g. on systems with no graphics display.

The usage of boinccmd is:

```
boinccmd [--host hostname] [--passwd passwd] command
```

If you run boinccmd in the same directory as the BOINC client, you don't need to supply either a host name or a password.

Otherwise you need to supply (as password) the string stored in the file `gui_rpc_auth.cfg` in the client's data directory. If you run boinccmd remotely you also need to configure the client to accept remote control.

If the client uses a non-default GUI RPC port, you can specify it as `hostname: port`.

D.1 Account query and attach

--lookup_account URL email password

Look up account and print account key.

--create_account URL email password name

Create account with the given email address, password, and user name

--project_attach URL account_key

Attach to an account

--join_acct_mgr URL name password

Attach to an account manager (or do RPC if already attached).

--quit_acct_mgr

Detach from the current account manager.

D.2 State queries**--get_cc_status**

Show CPU/GPU/network run modes and network connection status (version 6.12+)

--get_state

Show complete client state

--get_tasks

Show tasks

--get_simple_gui_info

Show projects and active tasks

--get_file_transfers

Show file transfers

--get_project_status

Show status of all projects

--get_disk_usage

Show disk usage by project

--get_proxy_settings

Get proxy settings

--get_messages seqno

Show messages with sequence numbers beyond the given seqno

--get_host_info

Show host info

--version, -V

Show core client version

D.3 Control operations

--task URL task_name operation {--window_station ws} {--desktop dt} {--display dp}

Do operation on a task, identified by the project master URL and the task name.
operations:

- suspend: temporarily stop work on task
- resume: allow work on task
- abort: permanently stop work on task
- graphics_window: open graphics in a window. The optional desktop/window_station (Windows) or display (X11) arguments specify the display.
- graphics_fullscreen: open graphics fullscreen

--project URL operation

Do operation on a project, identified by its master URL. Operations:

- reset: delete current work and get more;
- detach: delete current work and don't get more;
- update: contact scheduling server;
- suspend: stop work for project;
- resume: resume work for project;
- nomorework: finish current work but don't get more;
- allowmorework: undo nomorework
- detach_when_done: detach project

--file_transfer URL filename {retry | abort}

Do operation on a file transfer

--set_run_mode {always | auto | never} [duration]

Set run mode.

- always: do CPU work always
- auto: do work only when allowed by preferences
- never: don't do work

If duration is zero or absent, this mode is permanent. Otherwise, after 'duration' seconds elapse, revert to last permanent mode.

--set_gpu_mode {always | auto | never} [duration]

Set GPU mode. Like set_run_mode but applies to GPU computation.

--set_network_mode {always | auto | never} [duration]

Set network mode. Like set_run_mode but applies to network transfers

**--set_proxy_settings http_server_name http_server_port http_user_name
http_user_passwd socks_server_name socks_server_port socks_version
socks5_user_name socks5_user_passwd**

Set proxy settings (all fields are mandatory). (exists but doesn't work before 6.6.12).

--run_benchmarks

Run CPU benchmarks

**--set_screensaver_mode on|off blank_time [--desktop desktop] [--
window_station window_station] [--display display]**

Tell the core client to start or stop doing fullscreen graphics, and going to black after blank_time seconds. The optional arguments specify which desktop/windows_station (Windows) or display (X11) to use.

--read_global_prefs_override

Tell the core client to read the [PrefsOverride global_prefs_override.xml] file, and incorporate any global preferences indicated there.

--quit

Tell the core client to quit

--read_cc_config

Reread the configuration file (cc_config.xml)

--set_debts URL1 STD1 LTD1 {URL2 STD2 LTD2 ...}

Set the short- and long-term debts of one or more projects. Note: if you adjust the debts of a project, the debts of other projects are changed, so if you want to set the debts of multiple projects, do it in a single command.

--help, -h

Show options and commands

D.4 Examples

It's not hard to write useful scripts based on boinccmd, as long as you know your way around Unix tools. Here's one to run 'update' on all attached projects on your client:

```
for url in $(boinccmd --get_project_status | sed -n 's/\s*master URL: //p'); do
  boinccmd --project ${url} update;
done
```

If you have remote RPCs set up on your clients, it's easy to, for example, attach a project on all 50 machines, by looping over a list of IPs instead of a list of projects:

```
for num in $(seq 2 50); do
  boinccmd --host 192.168.42.${num} --passwd 1234 \
  --project_attach http://project_url/ a84dc0bec631cbf81e25e6e7cd9ca826;
done;
```

That will connect to the machines 192.168.42.2 - 192.168.42.50 using the RPC password '1234' and make them attach to http://project_url/ with the specified account key.

Appendix E

Glossary

PC: Personal Computer

UK: United Kingdom

BOINC: Berkeley Open Infrastructure for Network Computing

GFLOPS: Giga Floating Point Operations per second

LAN: Local Area Network

CPU: Central Processing Unit

CRC: Class Responsibility Collaborator.

RAM: Read Access Memory

ROM: Read Only Memory

I/O: Input/Output

GUI: Graphical User Interface

APIs: Application Programming Interfaces

PPU: Palestine Polytechnic University

IT: Information Technology

NSF: National Science Foundation

PACI: Partnership for Advanced Computational Infrastructure

NIH: National Institute of Health

OS: Operating System

DB: Database

HW: Hardware

SW: Software

RPC: Remote Procedure Call

GUI: Graphical User Interface

IDE: Integrated Development Environment

GFLOPS: Giga Floating-point Operations Per Second

References

- [1] BOINC: Berkeley Open Infrastructure for Network Computing, <http://boinc.berkeley.edu/>, (accessed on 13/11/2013).
- [2] What is grid computing?, <http://www.gridcafe.org/EN/what-is-the-grid.html>, (accessed in September, 2013).
- [3] Bader Ahmed Bader Ajrab, "PC Grid Computing Environment In Higher Education Institutions", master thesis at AlQuds university, Palestine, 2013.
- [4] Grid computing in 30 seconds, <http://www.gridcafe.org/EN/grid-in-30-sec.html> , (accessed in September, 2013).
- [5] D.P. Vidyarthi, B.K. Sarker, L.T. Yang, "Scheduling in Distributed Computing Systems Analysis, Design & Models", A Research Monograph, pp.(244-245), 2009
- [6] Grid architecture, <http://www.gridcafe.org/EN/grid-architecture.html> , (accessed in October, 2013).
- [7] Middleware, <http://www.gridcafe.org/EN/middleware.html>, (accessed in October, 2013).
- [8] Ault, M. and Tumma, M., "Oracle10g Grid Computing with RAC", Oracle RAC - Types of Grid computing, 2004.
- [9] Stanoevska K., Wozniak T., Ristol S., "Grid and Cloud Computing: A Business Perspective on Technology and Applications", Springer, 2010.
- [10] Goyal B, Lawande S (2005) Grid Revolution: An Introduction to Enterprise Grid Computing. McGraw-Hill, Emeryville, CA, 2005
- [11] Joseph J., Fellenstein C., "Grid Computing", Pearson Education, 2004.
- [12] Ian Foster, Carl Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 2004.
- [13] Fox, G., Furmanski, W. (1998) High performance commodity computing, Chapter 10, in Foster, I. and Kesselman, C. (eds) The Grid: Blueprint for a New Computing Infrastructure. San Francisco, CA: Morgan Kaufmann Publishers.
- [14] Foster, I. and Kesselman, C., "The grid: Blueprint for a new computing infrastructure", Morgan Kaufmann, San Francisco, CA, 1998.

- [15] Fran Berman, Anthony J.G. Hey, Geoffrey C. Fox,
"Grid Computing Making the Global Infrastructure a Reality", Wiley Series in
Communications Networking and Distributed Systems, pp.(722-723),2003.
- [16] Grid-powered projects, <http://www.gridcafe.org/EN/grid-powered-project.html> ,
(accessed in October, 2013).
- [17] Volunteer computing , <http://www.gridcafe.org/EN/volunteer-computing.html>
,(accessed on September, 2013).
- [18] Desktop Grid, <http://boinc.berkeley.edu/trac/wiki/DesktopGrid>, (accessed in
October, 2013).
- [19] Volunteer computing vs. cloud vs. grid vs. HPC ,
<http://www.volunteer-computing.org/EN/volunteer-computing-vs-cloud-vs-grid-vs-HPC.html>, (accessed in October, 2013).
- [20] Volunteer computing , http://en.wikipedia.org/wiki/Volunteer_computing,
(accessed in October, 2013).
- [21] Desktop_Grid:Westminster_Local_DG,
http://wgrass.wmin.ac.uk/index.php/Desktop_Grid:Westminster_Local_DG,
(accessed in October, 2013).
- [22] NEW DIY SUPERCOMPUTER SAVES £1,000S,
<http://www.westminster.ac.uk/news-and-events/news/2011/university-of-westminster-launches-new-diy-supercomputer-saving-hundreds-of-thousands-of-pounds>, (accessed in October, 2013).
- [23] Middleware - Volunteer garage, <http://www.volunteer-computing.org/EN/middleware.html>, (accessed in October, 2013).
- [24] Anderson," BOINC: A system for public-resource computing and storage", 5th
IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA, pp. 4-
10,Dec,2004.
- [25] Grid MP - Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Grid_MP, (accessed in October, 2013).
- [26] Alchemi v0.6.1 Documentation,
http://www.cloudbus.org/~alchemi/doc/0_6_1/index.html, (accessed in October,
2013).

- [27] Alchemi [.NET Grid Computing Framework], <http://www.cloudbus.org/~alchemi/>, (accessed in October, 2013).
- [28] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchem Framework", Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.
- [29] SETI, <http://setiathome.berkeley.edu/>, (accessed on 13/11/2013)
- [30] BOINCstats, <http://boincstats.com/>, (accessed on 13/11/2013)
- [31] TOP500 list, <http://www.top500.org/list/2012/11/>, (accessed on 13/11/2013).
- [32] Anderson, D., Korpela, E. and Walton, R., High-Performance Task Distribution for Volunteer Computing , Proceedings of the First IEEE International Conference on e-Science and Grid, . Melbourne, Australia, 2005.
- [33] JobSubmission-Boinc. <http://boinc.berkeley.edu/trac/wiki/JobSubmission>, (accessed in January, 2014).
- [34] JobTemplates-Boinc. <http://boinc.berkeley.edu/trac/wiki/JobSubmission>, (accessed in January, 2014).
- [35] Free CPU usage monitor programs, <http://softwaresolution.informer.com/Free-CPU-Usage-Monitor/>.(accessed in January, 2014).
- [36] Patricio Domingues, Paulo Marques, Luis Silva,"Resources Usage of Windows Computer Laboratories", Escola Superior de Tecnologia e Gestão – Instituto Politécnico de Leiria – Portugal Departamento Eng. Informática, Universidade de Coimbra – Portugal, Jan, 2005.
- [37] BasicConcepts-BOINC, <http://boinc.berkeley.edu/trac/wiki/BasicConcepts>, (accessed on 06/05/2014).
- [38] Example applications, <http://boinc.berkeley.edu/trac/wiki/ExampleApps#no1>, (accessed on 8/5/2014).
- [39] SingleJob-BOINC, <http://boinc.berkeley.edu/trac/wiki/SingleJob>, (accessed on 07/05/2014).
- [40] HtmlOps-BOINC, <http://boinc.berkeley.edu/trac/wiki/HtmlOps>, (accessed on 04/04/2014).
- [41] M. Alfalayleh and L. Brankovic, "an overview of security issues and techniques in mobile agents", The University of Newcastle, 2004.

[42] SecureHttp-BOINC, <http://boinc.berkeley.edu/trac/wiki/SecureHttp>, (accessed on 02/04/2014).

[43] Free CPU Usage Monitor, <http://softwaresolution.informer.com/Free-CPU-Usage-Monitor/>, (accessed on 10/01/2014).

[44] FLOPS - Wikipedia, the free encyclopedia, <http://en.wikipedia.org/wiki/FLOPS>, (accessed 22/05/2014).

[45] AppCoprocesor, <http://boinc.berkeley.edu/trac/wiki/AppCoprocesor>, (accessed on 06/05/2014).

[46] GPU Computing-BOINC, http://boinc.berkeley.edu/wiki/GPU_computing, (accessed on 06/05/2014).

[47] ResearchProjects-BOINC, <http://boinc.berkeley.edu/trac/wiki/ResearchProjects>, (accessed on 06/05/2014).

[48] Creating custom installers, http://boinc.berkeley.edu/wiki/Creating_custom_installers, (accessed on 06/05/2014).

[49] AppIntro-BOINC, <http://boinc.berkeley.edu/trac/wiki/AppIntro>, (accessed on 08/05/2014).

[50] WrapperApp-BOINC, <http://boinc.berkeley.edu/trac/wiki/WrapperApp>, (accessed on 08/05/2014).

[51] VolunteerComputing-BOINC, <http://boinc.berkeley.edu/trac/wiki/VolunteerComputing>, (accessed on 10/05/2014).

[52] Creating and Configuring a BOINC Project, http://www.spy-hill.net/myers/help/boinc/Create_Project.html#server, (accessed on 10/05/2014).

[53] Android FAQ-BOINC, http://boinc.berkeley.edu/wiki/Android_FAQ, (accessed on 10/05/2014).

[54] Setting up a BOINC server, <http://boinc.berkeley.edu/trac/wiki/ServerIntro>, (accessed on 10/1/2014).

[55] BOINC server guide installation, <https://wiki.debian.org/BOINC/ServerGuide/Initialisation>, (accessed 15/1/2014).

[56] Installing BOINC, http://boinc.berkeley.edu/wiki/Installing_BOINC, accessed on 02-May- 2014

[57] Installing BOINC On Ubuntu, http://boinc.berkeley.edu/wiki/Installing_BOINC_on_Ubuntu, (accessed on 02/05/2014).

- [58] Controlling BOINC Remotely, http://boinc.berkeley.edu/wiki/Controlling_BOINC_remotely, (accessed on 02/05/2014).
- [59] BOINC DB, <http://boinc.berkeley.edu/trac/wiki/DataBase>, (accessed on 8/5/2014).
- [60] Server directory structure, <http://boinc.berkeley.edu/trac/wiki/ServerDirs>, (accessed on 8/5/2014).
- [61] Project configuration file, <http://boinc.berkeley.edu/trac/wiki/ProjectConfigFile>, (accessed on 8/5/2014).
- [62] Server Components, <http://boinc.berkeley.edu/trac/wiki/ServerComponents>, (accessed on 8/5/2014).
- [63] Setting up a BOINC server, <http://boinc.berkeley.edu/trac/wiki/ServerIntro>, (accessed on 8/5/2014).
- [64] anonscm.debian.org Git - pkg-boinc, http://anonscm.debian.org/gitweb/?p=pkg-boinc/scripts.git;a=blob:f=server-examples/boinc_project_maker.sh, (accessed in January, 2014).
- [65] Code signing, <http://boinc.berkeley.edu/trac/wiki/CodeSigning>, (accessed on 02/04/2014).
- [66] KeySetup-BOINC, <http://boinc.berkeley.edu/trac/wiki/KeySetup>, (accessed on 02/04/2014).
- [67] StartTool-BOINC, <http://boinc.berkeley.edu/trac/wiki/StartTool>, (accessed on 12/05/2014).
- [68] Boinccmd tool-BOINC, http://boinc.berkeley.edu/wiki/Boinccmd_tool, (accessed on 07/04/2014).
- [69] The design concept and initial implementation of Agent Teamwork grid computing middleware, <http://www.academicpub.com/map/items/3933371.html>, (accessed on 28/5/2014).
- [70] BOINC Security-BOINC, http://boinc.berkeley.edu/wiki/BOINC_Security, (accessed on 30/05/2014).