Universidad Carlos III de Madrid
Escuela Politécnica Superior
Bachelor's Degree in Computer Science and Engineering

Bachelor Thesis

A Complete Simulator for Volunteer Computing Environments

Author: Saúl Alonso Monsalve
Supervisor: Félix García Carballeira

Leganés, Madrid, Spain
June 2016

Insanity: doing the same thing over and over again and expecting different results.

Albert Einstein

Agradecimientos

En primer lugar, me gustaría agradecer a mi madre Toñi y a mi hermana Elba el enorme apoyo que me han brindado durante estos cuatro años. Mamá, gracias por aguantarme todo este tiempo, por todas las veces que he intentado explicarte sin éxito en qué consistía mi trabajo, aunque finalmente parece que te has hecho una idea. Elba, aunque estemos cada uno en una punta del mundo, sabes que siempre estaré a tu lado. Mil gracias por tu ayuda a lo largo de este tiempo. Os quiero mucho a las dos.

Por otro lado, querría dedicar este trabajo a mis abuelos. Aunque ya no estéis aquí, sé que os habría hecho muy felices verme terminar mis estudios universitarios. Os recordaré siempre.

Una mención especial es para Félix, mi tutor. Te agradezco de corazón haberme dado la oportunidad de realizar este trabajo y de iniciarme en el mundo de la investigación. Cada día que pasa aprendo algo de ti.

También me gustaría dar las gracias a todos los miembros del grupo ARCOS por acogerme tan bien. A Javi, por estar siempre ahí. De pasar tanto tiempo juntos nos han bautizado como Zipi y Zape. A Carlos y a Silvina, por mostrarse siempre dispuestos a echarme un cable cuando lo he necesitado. A Alex, por ser el primero en ofrecer su ayuda en cualquier momento (y por esas largas conversaciones en el laboratorio cuando ya no quedaba nadie en la universidad). A los Albertos, a Estefanía y a Fran, por contar siempre con una sonrisa que anima a cualquiera.

Este trabajo ha sido la culminación a cuatro intensos años. Ha habido momentos malos, pero también muchos buenos. Me gustaría destacar a Víctor, que ha sido posiblemente el compañero de clase con el que más he compartido. A Alejandro, porque aunque seas un poco pesado, nunca te he visto poner una mala cara. A Juanlu, por ser la persona con mejor corazón que he conocido. A Guille, por nuestras largas conversaciones sobre política. A Sandra y a Alex, porque sigáis siendo así siempre. Es un placer haber compartido estos años con vosotros, espero que esto no sea un adiós.

Por último, y no por ello menos importante, me gustaría destacar a Érik y a Fran. Perdonad todas esas veces que me habéis llamado para vernos y estaba en la universidad haciendo prácticas. Érik, nos conocemos desde los 4 años y este año nos graduamos los dos. ¡Qué mayores nos hacemos! Sabes que eres y siempre serás un amigo de verdad. Fran, supongo que te sentará mal ser la última persona que mencione, pero recuerda que "los últimos serán los primeros". Espero que no cambies nunca (bueno, en realidad me gustaría que aprendieras a tocar la guitarra de verdad). Sabes que tienes un amigo para toda la vida.

A Complete Simulator for Volunteer Computing Environments

by

Saúl Alonso Monsalve

Abstract

Volunteer computing is a type of distributed computing in which ordinary people donate their idle computer time to science projects like SETI@home, Climateprediction.net and many others. BOINC provides a complete middleware system for volunteer computing, and it became generalized as a platform for distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, environmental science, and astrophysics. In this document we present the whole development process of ComBoS, a complete simulator of the BOINC infrastructure. Although there are other BOINC simulators, our intention was to create a complete simulator that, unlike the existing ones, could simulate realistic scenarios taking into account the whole BOINC infrastructure, that other simulators do not consider: projects, servers, network, redundant computing, scheduling, and volunteer nodes. The output of the simulations allows us to analyze a wide range of statistical results, such as the throughput of each project, the number of jobs executed by the clients, the total credit granted and the average occupation of the BOINC servers. This bachelor thesis describes the design of ComBoS and the results of the validation performed. This validation compares the results obtained in ComBoS with the real ones of three different BOINC projects (Einstein@home, SETI@home and LHC@home). Besides, we analyze the performance of the simulator in terms of memory usage and execution time. This document also shows that our simulator can guide the design of BOINC projects, describing some case studies using ComBoS that could help designers verify the feasibility of BOINC projects.

Keywords: BOINC · Simulation · Throughput · Volunteer Computing

Supervisor: Félix García Carballeira
Title: Full Professor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This first chapter briefly presents the project, including the key characteristics of a the project and its motivation (Section 1.1, *Background and Motivation*), the project objectives (Section 1.2, *Objectives*), and the entire structure of the document (Section 1.3, *Document Structure*).

1.1 Background and Motivation

Volunteer computing is a type of distributed computing in which ordinary people donate their idle computer time to science projects. Since the late 1990's [1], volunteer computing systems, such as SETI@home [2], have become the largest and most powerful distributed computing systems in the world, offering an abundance of computing power at a fraction of the cost of dedicated, custom-built supercomputers. Most of the existing volunteer computing systems have the same basic structure: a client program that runs on the volunteer's computer, periodically contacting project-operated servers over the Internet to request jobs and report the results of completed jobs.

The computing resources that power volunteer computing are shared with the owners of the machines. Because the resources are volunteered, utmost care is taken to ensure that the volunteer computing tasks do not obstruct the activities of each machine's owner; a volunteer computing task is suspended or terminated whenever the machine is in use by another person. As a result, volunteer computing resources are volatile in the sense that a number of factors can prevent the task of a volunteer computing application from being completed. These factors include mouse or keyboard activity, the execution of other user applications, machine reboots, or hardware failures. Moreover, volunteer computing resources are heterogeneous, in the sense that they differ in operating systems, CPU speeds, network bandwidth and memory and disk sizes. Consequently, the design of systems and applications that utilize this system is challenging [1].

Berkeley Open Infrastructure for Network Computing (BOINC) [3] is an open-source volunteer computing platform which consists of approximately 60 projects. It provides a complete middleware system for volunteer computing. In fact, BOINC is the most widely used middleware system. According to BOINCstats [4], there are currently 57 projects, with more than 13 million hosts participating in these projects. The number of active hosts is around 1 million, offering 190 PetaFLOPS of computation. One example of this is the Einstein@home project, in which users regularly contribute about 1,080 TeraFLOPS of computational power, which would rank Einstein@home among the top 20 on the TOP500 [5] list which is constituted by the 500 fastest supercomputers of the world.

BOINC became generalized as a platform for distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, environmental science, and astrophysics. Currently, organizations such as universities or research centers are reluctant to use volunteer computing in their projects because they do not have sufficient resources to know a priori the results they would get. This happens because there are not too many volunteer computing simulators, and those that exist do not provide a complete simulation of all components included in the BOINC infrastructure. This is the reason why we have developed a simulator to solve this problem.

In this bachelor thesis we present a complete volunteer computing simulator based on the whole BOINC infrastructure. The software we have developed is called Complete BOINC Simulator (ComBoS). ComBoS simulates real volunteer computing scenarios. These scenarios are defined by a large set of parameters specified in an Extensible Markup Language (XML) file, including the number of projects, the characteristics of each project and the network environment. The outputs of the simulations allow us to analyze a wide range of statistical results, such as the throughput of each project, the number of jobs executed by the clients, the total credit granted and the average occupation of the BOINC servers. ComBoS has been implemented in C programming language, with the help of the tools provided by the MSG API of Simgrid [6]. Thanks to this, we have managed to perform massive simulations (> 500,000 clients) in just a few hours.

1.2 Objectives

The main objective of this project is to **develop a simulator that, unlike existing ones, can simulate real-world volunteer computing scenarios taking into account the complete infrastructure of BOINC, and can guide the design of BOINC projects**. The secondary objectives are:

- Providing a complete specification of the simulation parameters in each execution.
- Optimizing the simulator to be efficient in terms of time.
- Allowing simulations with hundreds of thousands of volunteer clients.
- Breaking down the simulator structure into discrete modules that allow to easily add new functionalities in the future.
- Implementing an scheduler on the client side that is close to the actual planning of the BOINC client.
- Creating a generator that allows the user to compile and generate the files (executables, platform, and deployment) required for each simulation.

1.3 Document Structure

The document contains the following chapters:

- Chapter 1, *Introduction*, presents a brief description of the document contents. It also includes the motivation and the objectives of the project.
- Chapter 2, *State of the Art*, includes a description of the different types of current distributed computing and presents the related work.
- Chapter 3, *Analysis*, briefly describes the project, explains the chosen solution, sets the requirements, and presents the regulatory framework of the project.
- Chapter 4, *Design*, details the design of the system, including all of its components.
- Chapter 5, *Implementation and Deployment*, includes the implementation details of the main parts of the developed software and the necessary features for the application deployment.

- Chapter 6, *Verification, Validation and Evaluation*, details a complete verification and validation of the project. It also shows an evaluation of different test cases using the simulator.
- Chapter 7, *Planning and Budget*, presents the concepts related to the followed planning, breaks down all the project costs, and describes the socio-economic environment.
- Chapter 8, *Conclusions and Future Work*, includes the contributions of the project, explains the main conclusions of the project and presents future work.
- Appendix A, *User Manual*, includes a complete user manual for the application. It contains a tutorial for the installation of the tools used, and a number of practical and educational examples to learn how to perform simulations using the developed software.

Chapter 2

State of the Art

This chapter presents the state of the art, the latest and most advanced stage of the technologies related to our application. First, we discuss the different types of large-scale distributed systems (Section 2.1). After this, we present the current methods and tools for the simulation of distributed systems (Section 2.2). Finally, we deal with the existing volunteer computing simulators (Section 2.3).

2.1 Large-Scale Distributed Systems

A distributed system consists of multiple autonomous computers that communicate through a computer network. These autonomous computers interact with each other in order to achieve a common goal [7, 8].

Anecdotally, in 1996 Nancy A. Lynch [9] explained that the word *distributed* in terms such as “distributed system”, “distributed algorithm”, and “distributed programming” originally referred to computer networks where individual computers were physically distributed within some geographical area. The terms are nowadays used in a much wider sense, even referring to autonomous processes that run on the same physical computer and interact with each other by message passing, as clarified Gregory R. Andrews and Shlomi Dolev few years later [10, 11].

There are several valid definitions of *distributed system*. The most commonly used share the following properties: [12]:

- *“There are several autonomous computational entities, each of which has its own local memory”* [9, 10, 11, 12, 13].

- *"The entities communicate with each other by message passing"* [10, 12, 13].

A distributed system may have a common goal, such as solving a large computational problem [12, 13]. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users [12, 13].

Other typical properties of distributed systems are:

- *"The system has to tolerate failures in individual computers"* [9, 12, 13].
- *"The structure of the system (network topology, network latency, number of computers) is not known in advance, the system may consist of different kinds of computers and network links, and the system may change during the execution of a distributed program"* [9, 13].
- *"Each computer has only a limited, incomplete view of the system. Each computer may know only one part of the input"* [9, 12, 13]

Most current ideas of distributed computing are based on High-Performance Computing. High-Performance Computing (HPC) is the use of super computers and parallel processing techniques for solving complex computational problems that are too large for standard computers. According to SearchEnterpriseLinux [14], the term applies especially to systems that function above 10^{12} Floating-point Operations per Second (1 TeraFLOP). The term HPC is occasionally used as a synonym for supercomputing, although technically a supercomputer is a system that performs at or near the currently highest operational rate for computers. Currently, there are supercomputers that work at more than 10^{15} Floating-point Operations per Second (1 PetaFLOP).

Then, we present the main large-scale distributed systems.

2.1.1 World Wide Web

WWW stands for World Wide Web. Not everyone knows that it is not synonymous with the Internet. The World Wide Web (WWW) (or just "the Web"), as ordinary people call it, is a subset of the Internet. The Web consists of pages that can be accessed using a Web browser. Popular web browsers include Internet Explorer, Mozilla Firefox, Opera, Chrome, Safari, and Netscape. The Internet is the actual network of networks where all the information resides.

Things like Telnet, File Transfer Protocol (FTP), E-commerce, Internet gaming, Internet Relay Chat (IRC), and e-mail are all part of the Internet, but are not part of the World Wide Web. The Hypertext Transfer Protocol (HTTP) protocol is used to transfer Web pages to computers. With hypertext, a word or phrase can contain a link to another Web site. All Web pages are written in the Hyper-Text Markup Language (HTML) language, which works in conjunction with HTTP [15].

2.1.2 Grid

Ian Foster and Carl Kesselman [16, 17, 18] explained that the term "the Grid" was coined in the mid-1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. Much progress has since been made on the construction of such an infrastructure and on its extension and application to commercial computing problems. And while the term "Grid" has also been on occasion conflated to embrace everything from advanced networking and computing clusters to artificial intelligence, there has also emerged a good understanding of the problems that Grid technologies address, and at least a first set of applications for which they are suited [16, 17, 18].

Grid technologies and concepts were originally developed to enable resource sharing within scientific collaborations, first within early Gigabit/sec testbeds [19, 20] and then on increasingly larger scales [21, 22, 23, 24]. Applications in this context include the federation of diverse distributed datasets, distributed computing for computationally demanding data analyzes (pooling of compute power and storage), coupling of scientific instruments with remote computers and archives (increasing functionality as well as availability), and collaborative visualization of large scientific datasets (pooling of expertise).

A common theme underlying these different usage modalities is a need for coordinated resource sharing and problem solving in multi-institutional, dynamic virtual organizations. More recently, it has become clear that similar requirements arise in commercial settings, not only for technical and scientific computing applications but also for commercial distributed computing applications, including enterprise application integration and business-to-business partner collaboration over the Internet. Just as the Web began as a technology for scientific collaboration and was adopted for e-business, a similar trajectory for Grid technologies is seen [16].

Ian Foster and Carl Kesselman [16] thus argue that both science and industry can benefit from Grids. However, at the risk of stating the case too broadly, they make a more comprehensive statement. A primary purpose of infrastructure and information technology is to enable people to perform their daily tasks more effectively or efficiently. To the extent that these tasks are performed in collaboration with others, Grids are more than just a niche technology, but rather a direction in which our infrastructure must evolve if it is to support the way work gets done in our society and our social structures.

They also consider [16] that the success of the Grid to date owes much to the relatively early emergence of clean architectural principles, de facto standard software, aggressive early adopters with challenging application problems, and a vibrant international community of developers and users. This combination of factors led to a solid base of experience that has more recently driven the definition of the service-oriented Open Grid Services Architecture that today forms the basis for both open-source and commercial Grid products.

2.1.3 Volunteer Computing

In 2005, David P. Anderson [25] defined Volunteer Computing (VC) [26] as a paradigm in which large numbers of computers, volunteered by members of the general public, provide computing and storage resources. Early volunteer computing projects include the Great Internet Mersenne Prime Search [27], SETI@home [2], distributed.net [28] and Folding@home [29]. VC is being used in high-energy physics, molecular biology, medicine, astrophysics, climate study, and other areas. This kind of platforms have been used mainly for the execution of Bag-of-Tasks, which does not require any interaction between network participants [30].

VC is a type of distributed computing in which ordinary people donate processing and storage resources to one or more scientific projects. Most of the existing VC systems have the same basic structure: a client program runs on the volunteer's computer, periodically contacting project-operated servers over the Internet, to request jobs and report the results of completed jobs. VC is important for several reasons [31]:

- Because of the huge number (> 1 billion) of PCs in the world, VC can supply more computing power to science than any other type of computing. In addition, this advantage will increase over time, because the number of PCs is in continuous growth.

- VC power can not be bought; it must be earned. A research project that has limited funding but large public appeal can get remarkable computing power. In contrast, traditional supercomputers are extremely expensive, and are available only for applications or teams that can afford them.

- VC promotes public interest in science, and provides the public with a voice in determining the directions of scientific research.

Since the late 1990's [1], VC systems, such as SETI@home [2], have become the largest and most powerful distributed computing systems in the world, offering an abundance of computing power at a fraction of the cost of dedicated, custom-built supercomputers. Many applications from a wide range of scientific domains –including computational biology, climate prediction, particle physics, and astronomy - have utilized the computing power offered by Volunteer Computing systems. VC systems have allowed these applications to provide computing resources to projects at a huge scale, often resulting in major scientific discoveries that would not have otherwise been possible.

The computing resources that power VC are shared with the owners of the client machines. Because the resources are volunteered, utmost care is taken to ensure that the VC tasks do not obstruct the activities of each machine's owner; a VC task is suspended or terminated whenever the machine is in use by another person. As a result, VC resources are volatile in the sense that any number of factors can prevent the task of a VC application from being completed. These factors include mouse or keyboard activity, the execution of other user applications, machine reboots, or hardware failures. Moreover, VC resources are heterogeneous, in the sense that they differ in operating systems, CPU speeds, network bandwidth and memory and disk sizes. Consequently, the design of systems and applications that utilize this system is challenging [1].

BOINC

Berkeley Open Infrastructure for Network Computing (BOINC) [3] is an open-source Volunteer Computing (VC) platform which consists of approximately 60 projects. It provides a complete middleware system for VC. In fact, BOINC is the most widely used middleware system. According to BOINCstats [4], currently there are 57 projects, with more than 13 million hosts participating in these projects. The number of active hosts is around 1 million, offering 190 PetaFLOPS of computation. One example of this is the Einstein@home project, in which users regularly contribute about 1,080 TeraFLOPS of computational power, which would rank Einstein@home among the top 20 on the TOP500 [5] list which is constituted by the 500 fastest supercomputers of the world.

David P. Anderson, the founder of BOINC, explained in a paper of 2004 [32] that BOINC makes it easy for scientists to create and operate public-resource computing projects. It supports diverse applications, including those with large storage or communication requirements. PC owners can participate in multiple BOINC projects, and can specify how their resources are allocated among these projects. BOINC is being used by several projects, including the famous SETI@home, Climateprediction.net, LHC@home, Predictor@home, and Einstein@Home. Volunteers participate by running a BOINC client program on their computers. They can "attach" each computer to any set of projects, and can control the fraction of the resource that is devoted to each project.

Some projects require efficient data replication: Einstein@home [33] uses large (40 MB) input files, and any given input file may be sent to a large number of hosts (in contrast with projects like SETI@home [2, 34, 35], where each input file is different).

The BOINC architecture [25] allows data servers to be located anywhere; they are simply web servers, and do not access the BOINC database. Current BOINC-based projects that use large files (Einstein@Home [36] and Climateprediction.net [37]) use replicated and distributed data servers, located at partner institutions. The upload/download traffic is spread across the commodity Internet connections of those institutions.

BOINC-based projects are autonomous. Each project operates a server consisting of several components [25]:

- Web interfaces for account and team management, message boards, and other features.
- A task server that creates tasks, dispatches them to clients, and processes returned tasks.
- A data server from which BOINC clients download input files and executables, and to which output files are uploaded.
- BOINC clients that download input files and executables, and upload output files.

These components share data stored on disks, including relational databases and a file storage (see Figure 2-1) [25]. Data servers handle file uploads using a certificate-based mechanism to ensure that only legitimate files, with prescribed size limits, can be uploaded. File downloads are handled by plain HTTP [32]. BOINC provides a form of redundant computing in which each computation is performed on multiple clients [38], the results are compared, and are accepted only when a 'consensus' is reached. In some cases new results must be created and sent.

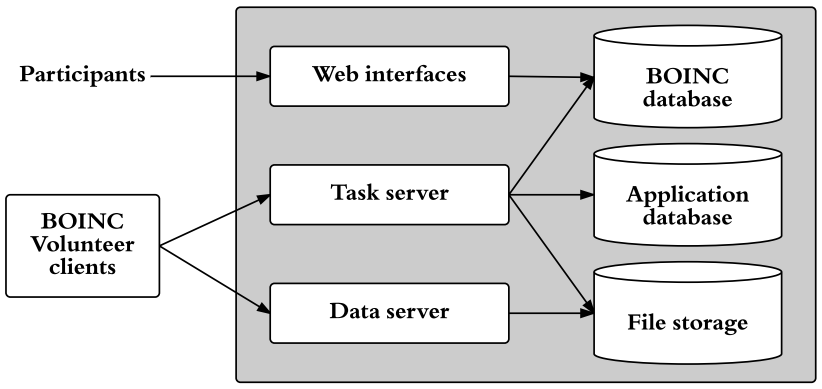


Figure 2-1: *A BOINC server consists of multiple components, sharing several forms of storage.*

Files (associated with application versions, workunits, or results) have project-wide unique names and are immutable. Files can be replicated: the description of a file includes a list of URLs from which it may be downloaded or uploaded [32]. Files can have associated attributes indicating, for example, that they should remain resident in a host after their initial use, that they must be validated with a digital signature, or that they must be compressed before network transfer [3].

The client downloads and uploads files and runs applications; it maximizes concurrency, using multiple CPUs when possible and overlapping communication and computation. BOINC's computational system also provides a distributed storage facility (of computational inputs or results, or of data not related to distributed computation) as a by-product. This storage facility is quite different from Peer-to-Peer (P2P) storage systems such as Gnutella, PAST [39] and Oceanstore [40]. In these systems, files can be created by any peer, and there is no central database of file locations. This leads to a set of technical problems (e.g. naming and file location) that are not present in the BOINC facility [32].

The BOINC architecture [41] is based on a strict master/worker model (see Figure 2-2), with a central server responsible for dividing applications into thousands of small independent tasks and then distributing the tasks to the worker nodes as they request the workunits. To simplify network communication and bypass any Network Address Translation (NAT) problems that might arise from bidirectional communication, the centralized server never initiates communication with worker nodes: all communication is initiated by the worker when more work is

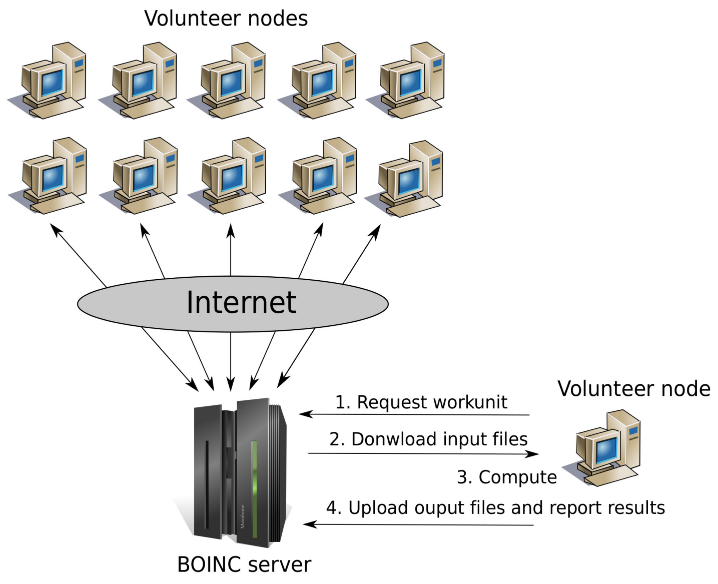


Figure 2-2: *Strict master/worker model in BOINC.*

needed or results are ready for submission [41].

2.1.4 Cloud

The best definition of cloud computing comes from an article in PC Mag written by Eric Griffith [42]: *"In the simplest terms, cloud computing means storing and accessing data and programs over the Internet instead of your computer's hard drive"* (see Figure 2-3). Cloud computing is provided for you as a service by another company and accessed over the Internet. Cloud computing covers a range of delivery and service models [43]. The common characteristic of these service models is an emphasis on pay-as-you-go and elasticity (which means the ability to quickly expand and collapse the utilized service as demand requires). Thus new approaches to data analysis and distributed computing have also emerged in conjunction with the growth of cloud computing. These include models like MapReduce and scalable key-value stores like Big Table [44, 45].

Cloud service models and computing technologies are attractive to scientific computing users due to the ability to control the software environment, as well as the ability to get on-demand access to resources to supplement or replace existing systems. Resource providers and scientific computing users servicing these users are considering the impact of these new models and technologies [46].

According to the NIST definition [47], cloud computing is composed of three service models

Figure 2-3: *Cloud computing model.*

and four deployment models, which are presented below.

Service Models

Cloud contributions are usually categorized as Infrastructure as a Service, Platform as a Service, and Software as a Service. Each of these models can play a significant role in scientific computing [43].

The distinction between the service models is based on the layer at which the service is abstracted to the end user (e.g., system software, hardware, etc.). The end user then has complete control over the software stack above the abstracted level.

- **Infrastructure as a Service (IaaS):** "*The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls)*" [47].

  Amazon Web Services [48] is the most currently used IaaS cloud computing platform. It provides a number of different levels of computational power and storage at various costs.

- **Platform as a Service (PaaS):** "*The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment*" [47].

  PaaS often includes facilities for application design, development, deployment and testing, and interfaces to manage scalability, security, state, storage, etc. Hadoop, Google App Engine, and Windows Azure are popular PaaS offerings in the commercial space.

- **Software as a Service (SaaS):** "*The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings*" [47].

  SaaS examples are: Cisco WebExm, Citrix GoToMeeting, Concur, Google Apps, Salesforce, and Workday.

Deployment Models

It is very important for businesses to understand their requirements before opting for various deployment models available on the cloud. Clouds can have one of the following deployment models, depending on how the cloud infrastructure is operated:

- **Private Cloud**: "*The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises*" [47].

- **Community Cloud**: "*The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by* *one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises"* [47].

- **Public Cloud**: *"The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider"* [47].
- **Hybrid Cloud**: *"The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds)"* [47].

2.1.5 Peer-to-Peer

Peer-to-Peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads among peers. Peers are equally privileged, equipotent participants in the application. They are said to form a P2P network of nodes (see Figure 2-4).

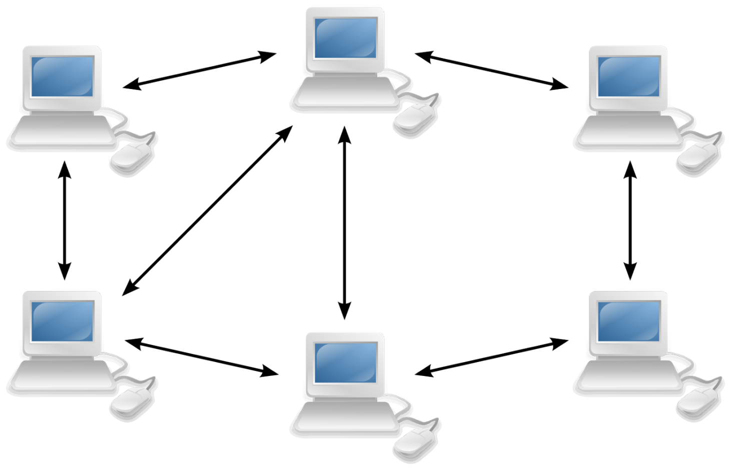


Figure 2-4: *A Peer-to-Peer basic network.*

Peers make a portion of their resources, such as network bandwidth, processing power, or disk storage, directly available to other network participants, without the need for central coordination by servers or stable hosts [49]. Peers are both consumers and suppliers of resources, in contrast to the traditional client-server model in which the supply and consumption of resources is divided.

While P2P systems had previously been used in many application domains [50], the architecture was popularized by the file sharing system Napster, originally released in 1999. The concept has inspired new philosophies and structures in many areas of human interaction. In such social contexts, P2P as a meme refers to the egalitarian social networking that has emerged throughout society, enabled by Internet technologies in general.

2.2 Simulation of Distributed Systems

This section presents the details related to the simulation of distributed systems and networks. In addition, we discuss the features of the SimGrid framework.

2.2.1 General Purpose Simulation Frameworks

Simulation frameworks are important to know the behavior of a system when it is not available under a given workload. In most cases, tests are unable to provide definitive answers, and maths are often not sufficient to fully understand these systems. For example, in a Formula 1 race, the mathematical predictions and the tests performed on cars are not enough to know the actual behavior of the cars in the race, because in the race a large number of unexpected events can occur: rain, crashes, etc. This is the main reason why simulations are of central importance. This is also the case for large-scale distributed systems. Simulations are vital in order to know the performance of these systems a priori. Examples of simulations are: simulating the behavior of a particular scheduling algorithm, or simulating the behavior of a distributed file system.

Different types of simulations are:

- **Emulation:** an emulator is a combination of hardware and software that recreates the operation and behavior of another system, and provides the service offered by the system that is being emulating.
- **Static simulation:** does not use the time as a parameter. The simulation runs until a certain equilibrium is achieved. A static simulation is managed by random number sequences. They are also called Monte Carlo simulations.
- **Discrete-event simulation:** models a system whose overall status changes with time. The overall status is updated whenever an event occurs.

2.2.2 Discrete-Event Simulations

As explained above, discrete-event simulations models systems whose overall status changes with time, and the overall status is updated whenever an event occurs. These systems are widely used for network simulations, specially in packet-level simulations. Pseudocode 2.1 shows the simulation algorithm.

Pseudocode 2.1 Discrete-event simulation algorithm.

```
1: Initiate state variables
2: $time$ = 0
3: Get the first event
4: while (there are events) and ($time < max\_time$) do
5:     Increase the time
6:     Get/remove the next event from the list
7:     Process the event:
8:         Update the global state
9:         Update the statistics of the simulation
10:        Generate new events
11: end while
12: Print results
```

The event generation is divided into two types:

- **Event tracing:** traces from real systems are recorded and these traces are used by the simulator.
- **Random distribution:** events and entries are generated from a determined statistic distribution function.

2.2.3 Network Simulations

Network simulations are divided into two main types: paquet-level simulation and flow-based simulation [51].

Paquet-Level Simulation

According to Pedro Velho and Arnaud Legrand [51], packet-level simulators use discrete-event simulation by which a flow over a network path can be represented as a sequence of events , such as packet arrivals and departures at end-points and routers. End-points and routers both implement full-fledge network protocols. Simulation time typically increases in proportion to the number of events [52]. Popular such simulators include Cnet [53, 54], SSFNet [55], ns-1 [56], ns-2 [57], ns-3 [58, 59], GTNetS [60], and INET [61]. The main problem with these simulators is that simulation time can be orders of magnitude larger than simulated time for simulations that involve realistic topologies with many flows. For instance, using GTNetS, which is known for good scalability, simulating 200 flows each transferring 100MB between two random end-points in a random 200-node topology for 125 sec of simulated time takes approximately 1500 sec on a 3.2GHz Xeon processor [62].

Although lower packet size leads to behavior presumably qualitatively closer to that of real networks, nothing in this simulator ensure that the behavior is quantitatively close to that of, for instance, TCP. Another simulator, GridSim [63] implements a protocol that includes some elements of UDP and allows for variable packet size. Like Bricks, GridSim requires small packet size to hope to gain accuracy close to that of true packet-level simulators on realistic network topologies, but then suffers from high simulation costs. Many other "grid" simulators exist, such as OptorSim [64], GangSim [65], Neko [66], or HyperSim [67] (readers interested in depth details are invited to consult [68]]). All implement some network model, but to the best of our knowledge (i.e., based on publications and/or on inspection of source codes), these simulators either use packet-level simulation or do not attempt to implement a model that realistically tracks the behavior of TCP networks [51].

Flow-Based Simulation

To increase the speed of network simulation one approach is to use theoretical models to compute the throughput of each flow in a network topology at a given time [51]. Models have been proposed [69, 70, 71], that model the throughput of a TCP flow as a function of packet loss and round trip delay, as well as some parameters of the network and of the TCP protocol. Unfortunately, some of these parameters are difficult to measure and instantiate for the purpose of grid simulations. Furthermore, it is not clear how the model can be applied to arbitrary network topologies with many simulated flows competing for network resources. Instead, one desires reasonable models that capture the bandwidth sharing behavior induced by TCP among flows on arbitrary topologies and that are defined by a few simple parameters, TCP congestion window size, and namely link physical bandwidths. This definition of macroscopic models of bandwidth sharing is challenging [72].

2.2.4 SimGrid

SimGrid [6, 73, 74] is a scientific instrument to study the behavior of large-scale distributed systems such as Grids, Clouds, HPC or P2P systems. The SimGrid framework [6, 73, 74] is a simulation-based framework for evaluating cluster, grid and P2P algorithms and heuristics. SimGrid was conceived as a scientific instrument, thus the validity of its analytical models was thoughtfully studied [75], ensuring their realism. The key features of SimGrid are:

- "*A scalable and extensible simulation engine that implements several validated simulation models, and that makes it possible to simulate arbitrary network topologies, dynamic compute and network resource availabilities, as well as resource failures*" [73].
- "*High-level user interfaces for distributed computing researchers to quickly prototype simulations either in C or in Java*" [73].
- "*APIs for distributed computing developers to develop distributed applications that canseamlesslyrun in simulation mode "or in real-world mode"*" [73].

SimGrid offers three user interfaces: MSG, SMPI and SimDag (see Figure 2-5).

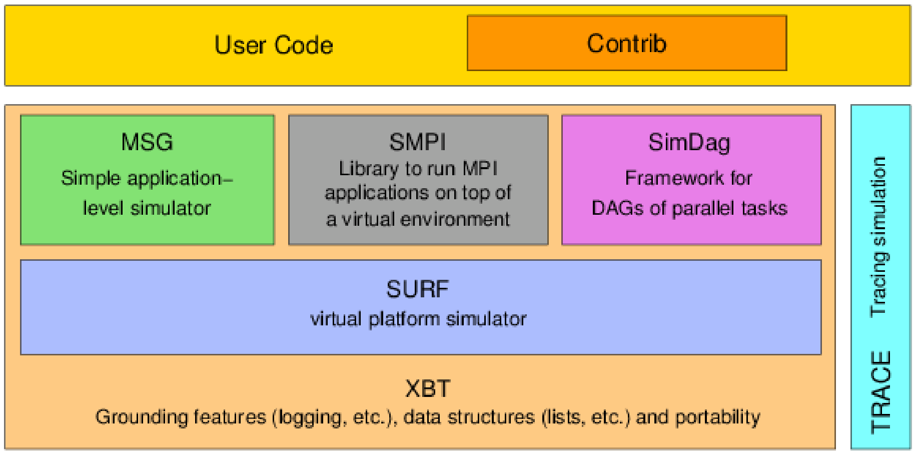

User Code
Contrib
MSG
Simple application-level simulator
SMPI
Library to run MPI applications on top of a virtual environment
SimDag
Framework for DAGs of parallel tasks
SURF
virtual platform simulator
XBT
Grounding features (logging, etc.), data structures (lists, etc.) and portability
TRACE
Tracing simulation

Figure 2-5: *SimGrid components.*

Currently (stable version 3.12) [73], **SimDag** is the descendant of SimGrid v1 and is designed for the investigation of scheduling heuristics for applications as task graphs. **MSG** is the interface introduced in SimGrid v2 to study CSPs and allows to use SimGrid as a development lab for real distributed applications. **SMPI** enables the direct simulation of Message Passing Interface (MPI) applications.

XBT is a "toolbox" module used throughout the software, which is written in ANSI C for performance. It implements classical data containers, logging and exception mechanisms, and support for configuration and portability. SURF is the code-name of the simulation engine. **SimIX** is an internal module (between MSG, SMPI, SimDag and SURF) that provides a POSIX-like API on top of SURF, thus easing the development of simulation APIs that implement the abstraction of multiple concurrent processes. For instance, it would allow the development of openMP- or BSP-like user interfaces [73].

SURF is the SimGrid simulation kernel. It was designed with two main goals in mind. First, it must be highly modular to allow the implementation (and comparison) of several resource models. In addition, as it constitutes the basis of the whole SimGrid framework, SURF must be carefully optimized so as not to hinder simulation speed [73].

MSG

MetaSimGrid [76], or MSG for short, is one of the three user interfaces of the SimGrid simulation framework. This interface was added into SimGrid v2 to allow the study of CSP applications. While initially intended for studying scheduling algorithms, it proved perfectly usable in other contexts, such as desktop grids and in time became the most widely used SimGrid API. For this reason, this interface is frozen: existing API functions will not changed (but new functions are added to fulfill new needs). This is to ensure that code written with MSG remains functional with subsequent releases of SimGrid. Version 3.3 introduces Java bindings to the MSG API (called jMSG), allowing user reluctant to program in C to still use SimGrid. MSG was the first distributed programming environment provided within SimGrid. While almost realistic, it remains quite simple. Users should use the MSG module if they want to study some heuristics for a given problem they do not really want to implement. The following notions are essential [76]:

- **Process**. Users need to simulate many independent scheduling decisions, so the concept of process is at the heart of the simulator. A process may be defined as a code, with some private data, executing in a host.
- **Host**. A host (or location) is any possible place where a process may run. Thus it may be

represented as a physical resource with computing capabilities, some mailboxes to enable running processes to communicate with remote ones, and some private data that can be only accessed by local processes.

- **Task**. Since most scheduling algorithms rely on a concept of task that can be either computed locally or transferred on another processor, it seems to be the right level of abstraction for users purposes. A task may then be defined by a computing amount, a message size, and some private data.
- **Link**. Like in real-life environments, hosts are connected through network links. Then, a link represents the physical notion of a network link that connects two hosts, or a host with a switch. A link may then be defined by a latency, and a bandwidth.
- **Mailbox**. For convenience, the simulator provides the notion of channel that is close to the TCP port notion.
  - Each mailbox is independent of the position of the network.
  - Messages are sent to a mailbox and received from a mailbox.
  - They are identified as strings.
  - The functioning of sending and receiving messages can be synchronous or asynchronous.

Figure 2-6 [76] shows a simple platform to simulate using MSG. The platform consists of a master and three workers. The processors on which the master and the workers are executing are all connected with the same compound link. To simulate this scenario using MSG, it is only necessary to specify the components (hosts, links, and router) of the simulation in a platform file (normally an XML file); write the master and worker processes in a C programming language file; and finally associate these processes with the corresponding hosts via a deployment file.

To sum up, using the above entities a simulator should be described only in terms of processes, running on hosts, and interacting by sending, receiving, and processing tasks. Algorithms implemented on top of SimGrid should not have direct access to links, but rather should be implemented as processes that send tasks to hosts using mailboxes. In fact, a host may have many associated mailboxes, and a mailbox is identified simply by a string. So, sending a task to a host using a mailbox consists of transfer the task through a particular link, and the task is received by a mailbox located on a host.

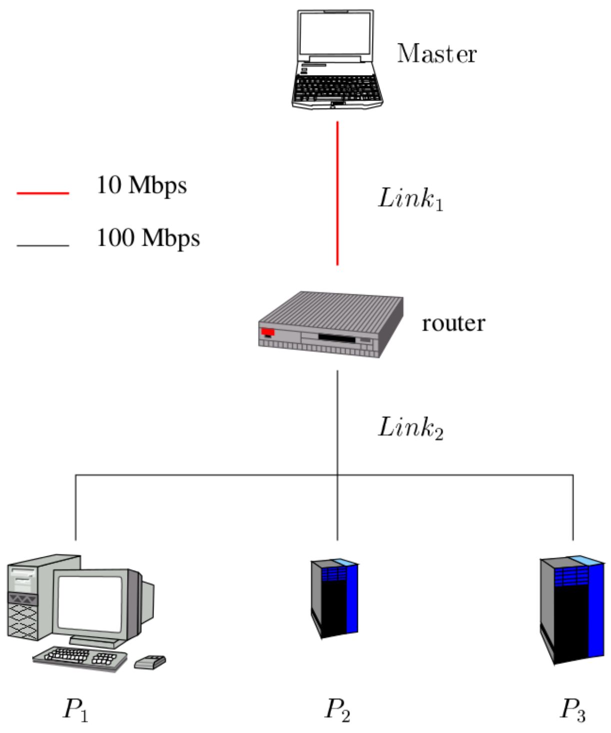


Figure 2-6: *A very simple MSG platform.*

SMPI

This module allows the simulation of unmodified MPI application [77] by intercepting MPI primitives in a manner similar to the MicroGrid approach [78]. SMPI [74] permits to study existing MPI application by emulating them on top of the SimGrid simulator. In other words, it will constitute an emulation solution for parallel codes.

SimDag

SimDag [74] allows to prototype and simulate scheduling heuristics for applications structured as task graphs of (possibly parallel) tasks. With this API one can create tasks, add dependencies between tasks, retrieve information about the platform, schedule tasks for execution on particular resources, and compute the DAG execution time. SimDag provides some functionnalities to simulate parallel task scheduling with DAGs models (Direct Acyclic Graphs). The old versions of SimGrid were based on DAGs. But the DAG part (named SG) was removed in SimGrid 3 because the new kernel (SURF) was implemented. SURF was much faster and more flexible than SG and did not use DAGs. SimDag is a new implementation of DAGs handling and it is built on top of SURF.

2.3 Volunteer Computing Simulators

Is volunteer computing a type of grid computing? David Anderson, director of SETI@home and founder of BOINC says no [79]. Grid computing and volunteer computing are forms of distributing computing that seek to exploit existing resources. However, there are several differences between them. They use different software and have different capabilities. On the one hand, in volunteer computing resources are owned and managed by ordinary people, users are anonymous, and project performance is not predictable. On the other hand, grid computing resources belong to information technology professionals, users may or may not be anonymous, and performance is partially predictable. This explains why we are not going to compare ComBoS, our simulator, with other grid computing simulators.

There are not too many volunteer computing simulators, although most of them are focused on BOINC. As ComBoS, our simulator, there are other BOINC simulators based on the SimGrid toolkit [6]. An example of this is SimBOINC, which simulates the BOINC client scheduler. SimBOINC [80] uses almost exactly the BOINC client's CPU scheduler source code. This is why SimBOINC simulations are almost perfect. SimBOINC code is public. Nevertheless, even though the results are optimal, this simulator is quite extensive (> 20,000 lines of source code) and not overly efficient in terms of time. Furthermore, like other simulators, SimBOINC is focused on the client side, leaving aside the other parts of the system. In 2010, a simulator with similar results to SimBOINC's was created [81], but it is more efficient (about three or four times faster) and has a source code of about 800 lines.

In contrast to the aforementioned simulators, SimBA [82] (Simulator of BOINC Applications) is a simulator that reproduces the creation, characterization and termination of workers by using trace files obtained from real BOINC projects. It simulates the BOINC server scheduler and its interaction with a large number of simulated hosts. Some weaknesses are that it is not highly scalable (< 50,000 hosts, while some projects have more than 100,000 active hosts at the moment), each project must be simulated individually, and there is no client scheduler.

Finally, although it is an emulator rather than a simulator, we feel the need to mention EmBOINC. EmBOINC [83] uses a population of volunteered clients and emulates the server component. EmBOINC does not have a client scheduler either. Table 2.1 compares the main features of the programs presented in this section (SimBOINC, SimBA and EmBOINC).

Features	SimBOINC	SimBA	EmBOINC
Network	-	traces	traces
Number of client hosts	1	40,000	100,000
Hosts availability	simulation	traces	traces
Hosts power	input	traces	traces
Hosts scheduler	simulation	-	-
Hosts reliability	-	traces	traces
Hosts execution	simulation	traces	traces
Hosts organization	-	individual hosts	individual hosts
Disk access	-	-	-
Number of tasks	-	200,000	350,000
Workunit validation	-	simulation	emulation
Workunit details	traces	input	traces
Project details	traces	-	-
Number of task servers	N	1	1
Number of data servers	-	-	-
Server scheduler	-	simulation	emulation

Table 2.1: *Comparison of main BOINC simulators.*

Our intention was to create a simulator that, unlike the existing ones, could simulate realistic scenarios taking into account the whole BOINC infrastructure. The result of our work is ComBoS, a simulator that executes complex simulations based on BOINC environments, creating the simulation platform from a complete XML input file and generating a set of statistical results, such as the throughput of each project, the number of jobs executed by the clients or the average occupation of the BOINC servers. We can manage multiple projects and hundreds of thousands of hosts in the same simulation. In the next chapters we describe our simulator in detail.

Chapter 3

Analysis

The main purpose of this chapter is to describe the project by obtaining and specifying the requirements for the simulator, which can provide enough information for a detailed analysis that, therefore, can serve to further design and implementation (Chapters 4, *Design*; and 5, *Implementation and Deployment*) of a software that meets those requirements.

In order to obtain the requirements of the system, the supervisor has played the role of the customer in different meetings, while the student has played the roles of analyst, designer, programmer, and tester.

Section 3.1 briefly summarizes the project description. Section 3.2 discusses the chosen solution and compares it to the alternatives considered. Section 3.3 specifies the system requirements, starting with the user requirements, and ending with the functional and non-functional requirements. Finally, Section 3.4 indicates the set of laws and regulations for the management of the software.

3.1 Project Description

As explained in Chapter 2, *State of the Art*, volunteer computing is a form of distributed computing in which volunteers donate processing and storage resources to computing projects. Usually, every job represents a portion of a larger problem whose computation is divided into smaller chunks and addressed in parallel. BOINC [32] is a middleware system for volunteer computing that makes it easy for scientists to create and operate public-resource computing projects. It supports diverse applications, including those with large storage or communication requirements. PC owners can participate in multiple BOINC projects, and can specify how their resources are allocated among these projects.

Figure 3-1 shows the BOINC architecture in a basic way, in which different client volunteers participate in a single project.

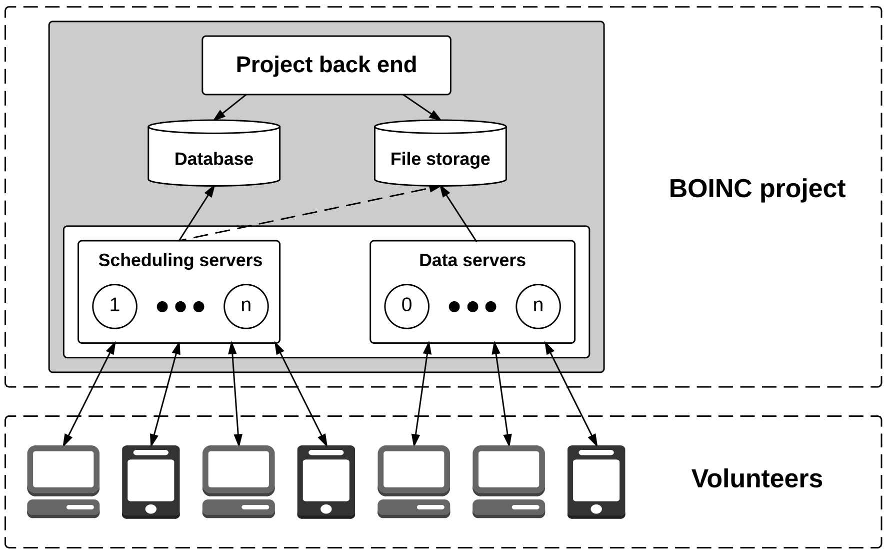


Figure 3-1: *Basic BOINC architecture.*

A BOINC project consists of a project back-end, which manages the whole project; scheduling servers, that communicate with participant hosts; and data servers, that distribute input files and collect output files. Volunteers contact with scheduling servers for jobs and download the corresponding files from data servers. After that, volunteers perform the computation and upon completion, they send the results to scheduling servers and upload output files to data servers.

The main objective of this project is to develop a simulator that, unlike existing ones, can simulate real-world volunteer computing scenarios and can guide the design of BOINC projects. For this bachelor thesis, we have developed a complete simulator of the BOINC infrastructure, called ComBoS.

3.2 Solution Selection

As explained in Chapter 1, *Introduction*, in this document we present ComBoS, a complete simulator of volunteer computing platforms. As discussed in Section 2.3, *Volunteer Computing Simulators* (contained in Chapter 2, *State of the Art*), the current existing volunteer computing simulators simulate only part of the system, and not the whole BOINC infrastructure. Some simulators are focused exclusively on the client side, and others on the server side. Others are not scalable enough with respect to the number of hosts, and other simulators do not allow for the simulation of several projects at the same time. The simulator developed during this project aims to combine all the features of the BOINC infrastructure in a single program (see Table 2.1).

To develop the simulator, we studied different tools/frameworks, which are presented in Table 3.1. To the best of our knowledge, the simulation frameworks that are best suited for the purpose of modelling and simulating volunteer computing environments are SimGrid [74], PVMsim [84], Virtual-GEMS [85], MDCSim [86] and SPECI-2 [87]. The features and models analyzed in Table 3.1 are:

Features	SimGrid	PVMsim	Virtual-GEMS	MDCSim	SPECI-2
Languages	C/C++/Java/Ruby	C	C/C++/Ruby	C++/Java	Java
Open Source	✓	✓	✓	✗	✓
Models					
Communication	✓	✗	✗	✓	✓
Energy	✓	✗	✗	✓	✗
Hardware	✓	✓	✓	✓	✓
Scheduling	✓	✓	✗	✗	✓
Users	✓	✗	✗	✓	✗

Table 3.1: *Comparison of simulation frameworks for distributed computing systems.*

- **Language:** programming languages compatible with the framework.
- **Open Source:** indicates whether the code and documentation is available to the public at no cost.
- **Communication:** indicates whether it is possible to design the communication between hosts.
- **Energy:** indicates whether it is possible to measure the energy consumption of the hosts.
- **Hardware:** indicates whether it is possible to indicate the hardware-level details of the hosts and networks involved in the simulations.
- **Scheduling:** indicates whether it is possible to implement different scheduling policies.

- **Users:** indicates whether it is possible to simulate the behavior of real users.

After analyzing the pros and cons of each one, we have chosen to use SimGrid, a simulation framework also presented in the previous chapter. SimGrid [74] is the simulation environment of distributed computing most widely used worldwide, and it is the only framework that covers all the models analyzed (Communication, Energy, Hardware, Scheduling, and Users). Furthermore, it is open-source, so we were able to use SimGrid without any problems.

As shown in Table 3.1, SimGrid allows the development of simulations using three different programming languages: C, Java and Ruby. Ruby is an interpreted programming language, so we have dismissed its use because the interpreted programming languages are not good in terms of efficiency. Figure 3-2 [74] shows the performance of the C and Java bindings provided by SimGrid. The best performance is achieved with the "raw" (without threads) C programming language, so we have decided to use it in order to develop the simulator.

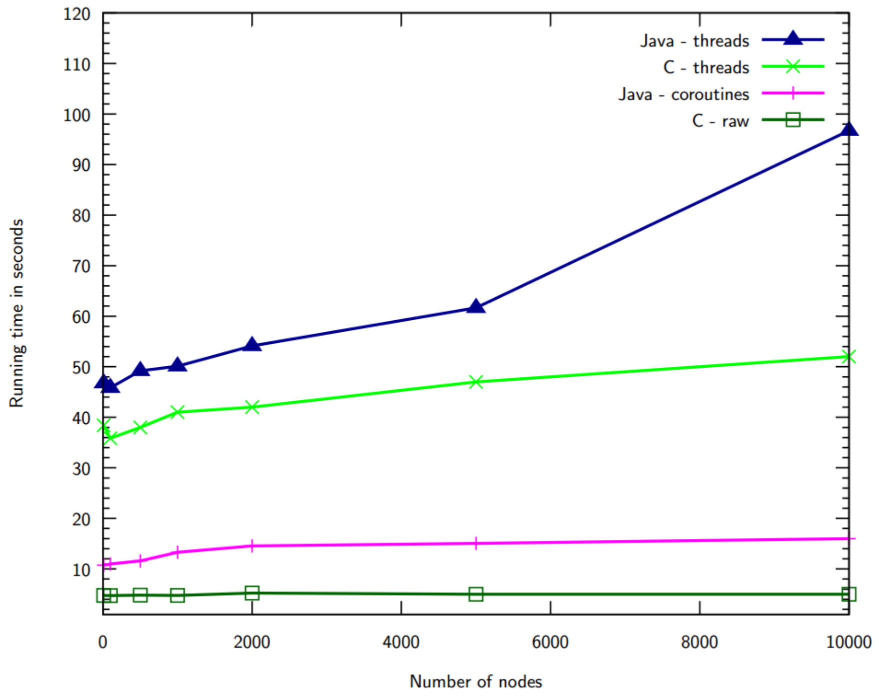


Figure 3-2: *SimGrid bindings performance.*

To sum up, the final project has involved the design, development and implementation of a simulator of the whole BOINC infraestructure, implemented using the C programming language and using the tools provided by SimGrid.

3.3 Requirements

This section provides a detailed description of the application requirements. For the requirement specification task, the IEEE recommended practices [88] were followed. According to these practices, a good specification must address the software functionality, performance issues, the external interfaces, other non-functional features and design or implementation constraints. Moreover, the requirements specification must be:

- **Complete:** the document reflects all significant software requirements.
- **Consistent:** requirements must not generate conflicts with each other.
- **Correct:** every requirement is one that the software shall meet according to the user needs.
- **Modifiable:** the structure of the specification allows changes to the requirements in a simple, complete and consistent way.
- **Ranked based on importance and stability:** every requirement must indicate its importance and its stability.
- **Traceable:** the origin of every requirement is clear and it can be easily referenced in further stages.
- **Unambiguous:** every requirement has a single interpretation.
- **Verifiable:** every requirement must be verifiable, that is, there exists some process to verify that the software complies with every single requirement.

Starting from the user requirements, which constitute an informal reference to the product performance that the client expects, we derived the software requirements (in this case, functional requirements and non-functional requirements) that guided the design process with specific information on the functionality of the system and other characteristics. The retrieved requirements were structured according with the following schema:

1. **User Requirements**
   (a) **Capacity:** the requirement describes the expected system functionality as in use cases.

   (b) **Restriction:** the requirement specifies constraints or conditions the system must fulfil.

2. **Software Requirements**

   (a) **Functional**

      i. **Functional:** the requirement describes the basic system functionality and purpose while minimizing ambiguity.

      ii. **Inverse:** the requirement limits the functionality of the application to clarify its scope.

   (b) **Non-Functional**

      i. **Performance:** the requirement is related to the minimum required performance of the resulting system.

      ii. **Interface:** the requirement is related to the user interface of the application.

      iii. **Scalability:** the requirement is related to the ability of the system to adapt to increasing workloads.

      iv. **Platform:** the requirement specifies the underlying software and hardware platforms in which the system will operate.

Table 3.2 provides the template used for requirements specification. Note that for user requirements, the ID format will be UR-XYY, where X indicates the requirement subtype: capacity requirements (C), or restrictions (R). YY corresponds to the requirement number under its subcategory. For software requirements, the ID format SR-X-YZZ will be used, where X indicates if it is a functional (F) or non-functional (NF) requirement, and Y represents its subcategory: functional (F), inverse (I), performance (P), interface (UI), scalability (S), or platform (PL). ZZ corresponds to the requirement number under its subcategory.

3.3.1 User Requirements

This subsection specifies the user requirements.

ID	Requirement ID.
Name	Requirement name.
Type	Indicates the category in which the requirement would be placed according to the previously described schema.
Origin	Constitutes the requirement source. It might be the user, another requirement or other stakeholders involved in the project.
Priority	Indicates the requirement priority according to its importance. A requirement can be identified either as *essential*, *conditional* or *optional*.
Stability	Indicates the requirement variability through the development process, defined as *stable* or *unstable*.
Description	Detailed explanation of the requirement.

Table 3.2: *Template for requirements specification.*

ID	UR-C01
Name	BOINC projects simulation
Type	Capacity
Origin	User
Priority	Essential
Stability	Stable
Description	The application shall simulate real BOINC projects.

Table 3.3: *User requirement UR-C01.*

ID	UR-C02
Name	Client scheduling
Type	Capacity
Origin	User
Priority	Essential
Stability	Stable
Description	The client scheduler of the simulator shall follow the actual BOINC client scheduling.

Table 3.4: *User requirement UR-C02.*

ID	UR-C03
Name	Simulation components
Type	Capacity
Origin	User
Priority	Essential
Stability	Stable
Description	The simulations shall cover all the elements present in the BOINC infrastructure.

Table 3.5: *User requirement UR-C03.*

ID	UR-R01
Name	Linux as underlying OS
Type	Restriction
Origin	User
Priority	Essential
Stability	Stable
Description	The simulator shall be designed for Linux operating systems.

Table 3.6: *User requirement UR-R01.*

ID	UR-R02
Name	SimGrid toolkit
Type	Restriction
Origin	User
Priority	Essential
Stability	Stable
Description	The application shall use the SimGrid toolkit in order to implement the distributed computing functionalities.

Table 3.7: *User requirement UR-R02.*

ID	UR-R03
Name	Scalability
Type	Restriction
Origin	User
Priority	Essential
Stability	Stable
Description	The simulator shall be scalable (carry out executions by simulating a large number of client hosts).

Table 3.8: *User requirement UR-R03.*

3.3.2 Functional Requirements

This subsection specifies the functional requirements.

ID	SR-F-F01
Name	Credit calculation
Type	Functional
Origin	UR-C01
Priority	Essential
Stability	Stable
Description	The simulator shall calculate the number of credits granted to each volunteer client analogously to actual BOINC projects.

Table 3.9: *Functional requirement SR-F-F01.*

ID	SR-F-F02
Name	Collection of statistics
Type	Functional
Origin	UR-C01
Priority	Essential
Stability	Stable
Description	The simulator shall collect, for each project, the same statistics that actual BOINC projects (published in BOINCstats [4]).

Table 3.10: *Functional requirement SR-F-F02.*

ID	SR-F-F03
Name	Almost identical outputs
Type	Functional
Origin	UR-C01
Priority	Essential
Stability	Stable
Description	The outputs of the simulator for existing projects should be almost identical to those published in BOINCstats [4].

Table 3.11: *Functional requirement SR-F-F03.*

ID	SR-F-F04
Name	Multiple BOINC projects
Type	Functional
Origin	UR-C01
Priority	Essential
Stability	Stable
Description	The simulator shall allow the simulation of different projects simultaneously.

Table 3.12: *Functional requirement SR-F-F04.*

ID	SR-F-F05
Name	Client scheduler
Type	Functional
Origin	UR-C02
Priority	Essential
Stability	Stable
Description	The client scheduler shall follow the actual BOINC client scheduling (described in [89]).

Table 3.13: *Functional requirement SR-F-F05.*

ID	SR-F-F06
Name	Realistic simulation elements
Type	Functional
Origin	UR-C03
Priority	Essential
Stability	Stable
Description	All simulations shall include the following elements: tasks, volunteer hosts, servers, data servers, networks, and hosts availability.

Table 3.14: *Functional requirement SR-F-F06.*

3.3.3 Non-Functional Requirements

This subsection specifies the non-functional requirements.

ID	SR-NF-PL01
Name	Ubuntu 14.04
Type	Platform
Origin	UR-R01
Priority	Essential
Stability	Stable
Description	The simulator shall work on the Ubuntu Linux distribution, version 14.04.

Table 3.15: *Non-functional requirement SR-NF-PL01.*

ID	SR-NF-PL02
Name	SimGrid MSG API
Type	Platform
Origin	UR-R02
Priority	Essential
Stability	Stable
Description	The implementation, setup and control of the simulations shall be carried out using the MSG API of the SimGrid toolkit.

Table 3.16: *Non-functional requirement SR-NF-PL02.*

ID	SR-NF-PL03
Name	C programming language
Type	Platform
Origin	UR-R03
Priority	Essential
Stability	Stable
Description	The simulator shall be written in the C programming language.

Table 3.17: *Non-functional requirement SR-NF-PL03.*

ID	SR-NF-S01
Name	Large simulations
Type	Scalability
Origin	UR-R03
Priority	Essential
Stability	Stable
Description	The application must be able to perform simulations with more than 100,000 hosts in a machine with at least 8 GB of Random-Access Memory (RAM).

Table 3.18: *Non-functional requirement SR-NF-S01.*

ID	SR-NF-P01
Name	Linear-time execution
Type	Performance
Origin	UR-R03
Priority	Conditional
Stability	Stable
Description	Runtime of the simulator must be linear (approximately) in the number of hosts.

Table 3.19: *Non-functional requirement SR-NF-P01.*

ID	SR-NF-UI01
Name	Simulation parameters
Type	Interface
Origin	Analyst
Priority	Essential
Stability	Stable
Description	To perform simulations, users only need to specify the simulation parameters in an XML file.

Table 3.20: *Non-functional requirement SR-NF-UI01.*

ID	SR-NF-UI02
Name	Progress bar
Type	Interface
Origin	Analyst
Priority	Conditional
Stability	Stable
Description	The simulations should include a progress bar.

Table 3.21: *Non-functional requirement SR-NF-UI02.*

3.4 Regulatory Framework

This section discusses the necessary constraints taking into account the regulatory framework. Specifically, the legal restrictions applicable to the simulator are specified.

3.4.1 Legal Constraints

In the real BOINC system, users must be registered, and BOINC databases handle confidential information from users, so it is necessary to ensure that third parties can not access that information. One solution is to encrypt the information transmitted following some cryptographic protocol. In Spain, this requirement is specified in the article 104 of the RD 1720/2007 [90], which deals with the Spanish Data Protection Law.

In contrast, the developed application does not use private data from users, and neither transmits any confidential information to third-parties, because it is just a simulator that does not even require Internet access.

On the other hand, it is crucial that our simulator be available as an open-source software. We want it to be such that anyone can redistribute the code or modify it by the terms of the GNU Lesser General Public License (LGPL) [91]. To do this, our simulator is available on the following website: `https://www.arcos.inf.uc3m.es/~combos/`.

Chapter 4

Design

This chapter provides a complete description of the developed simulator, including the internal architecture and the different software components. ComBoS is a complete simulator of BOINC infrastructures that simulates the behavior of all componentes involved: projects, servers, network, scheduling, redundant computing, and volunteer nodes. In this chapter we describe all the simulator components (Section 4.1, *Simulator Components*) and present the policies of the client scheduler (Section 4.2, *Local Scheduling Policies*).

4.1 Simulator Components

In order to understand the architecture of the simulator in the simplest way possible, we have divided all components of the simulator into two groups: the server side and the client side. This component division makes that in the future we can easily add new modules to the system. The specification of the networks that connect both groups is detailed in the client side. In the server side, jobs are created and distributed to the clients. A BOINC job has two parts:

- A *workunit* describing the computation to be performed. Each workunit has asocciated a list of input files: their names, and the names by which the application refers to them. Typically these input files are downloaded from a data server [92].

- One or more *results*, each of which describes an instance of a computation, either unstarted, in progress, or completed. The BOINC client software refers to results as *tasks*. In this document, we use both terms interchangeably. When a volunteer has completed a task, it sends the result to a scheduling server and uploads the corresponding output files to a data server [92].

The aim of this section is to explain the complete architecture of the simulator, as can be seen in Figure 4-1. Figure 4-1 shows all the components of the simulator, distinguishing the client side and the server side. The network consists of links that connect the client side with the server side. Throughout this section, each part of the complete architecture of the simulator is explained.

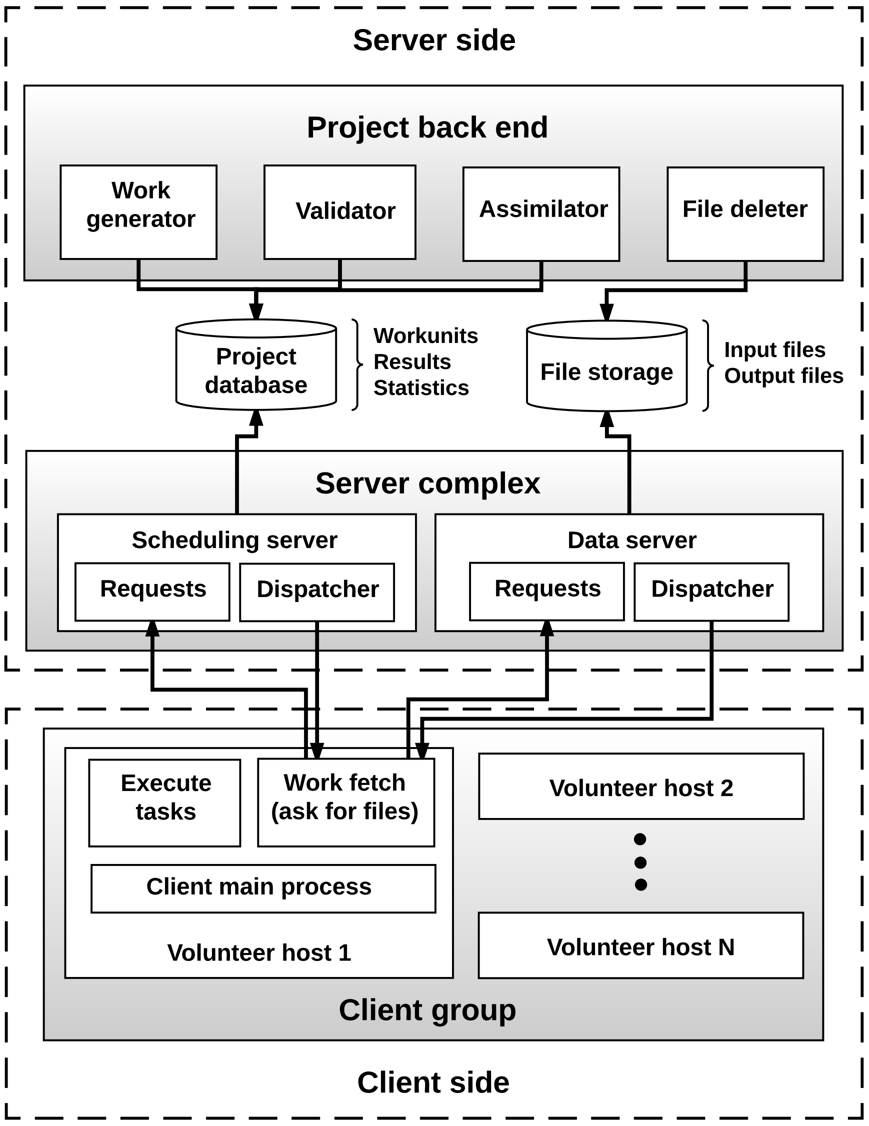


Figure 4-1: *Architecture of the simulator.*

4.1.1 Server Side

Servers are responsible for managing projects. The architecture of the server side is shown in Figure 4-2. The server side of a project consist of two parts [38]:

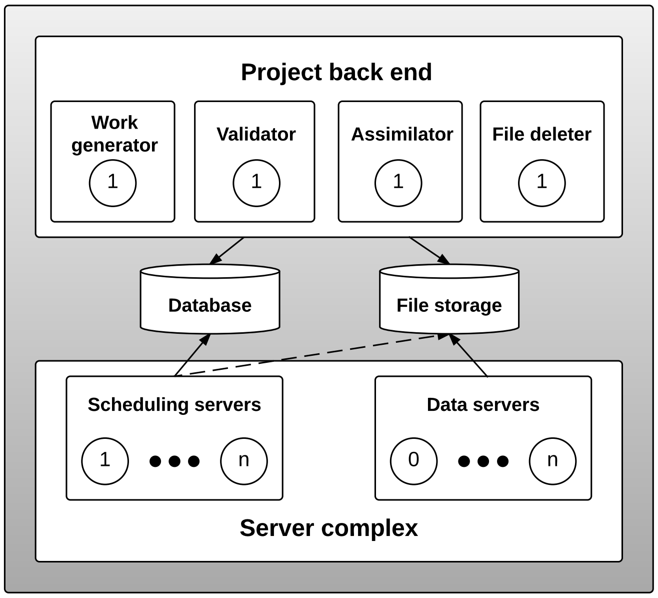


Figure 4-2: *Server side architecture of ComBoS.*

- A *project back end* that supplies applications and workunits, and that handles the computational results. It includes:
  - A *Work generator*, that creates workunits and the corresponding input files. There is one work generator per application. It works as a daemon program that maintains a given supply of work. It stores the workunits in the Database, and the input files in the File Storage (see Figure 4-2).
  - A *Validator*, that compares redundant results and selects a canonical result representing the correct output, and a canonical credit granted to users and hosts that return the correct output. There is one validator per application.
  - An *Assimilator*, that handles workunits that are ‘completed’: that is, that have a canonical result or for which an error condition has occurred. There is one assimilator per application. Handling a successfully completed result might involve recording results in a database and perhaps generating more work.
  - A *file deleter*, that deletes input and output files that are no longer needed.

- A *BOINC server complex* that manages data distribution and collection. It includes:
  - One or more *scheduling servers* (sometimes called *task servers*), that communicate with participant hosts. Volunteers contact with scheduling servers to ask for work and scheduling servers distribute work among volunteer clients. In addition, scheduling servers collect results from volunteer hosts.
  - *Data servers*, that distribute input files and collect output files. For small projects, if there are no data servers, scheduling servers also operate as data servers. These servers need not be owned or operated by the project. A project might, for example, recruit other organizations to donate network bandwidth by hosting data servers; data could be moved on tape between the project back end and the data servers.

ComBoS allows for the definition of multiple projects. For each project, users must define the parameters described in Table 4.1.

Projects grant credit to users for the amount of computational work they have contributed to the system. Credit is an extremely important incentive for users, and provides a mechanism for ranking users and teams. The source code of the BOINC client is open-source, and volunteers are untrusted and anonymous. That is the reason why malicious volunteers can easily hack the client code to report erroneous results, or to claim large amounts of credit. BOINC supports redundant computing in order to increase the likelihood that only correct results are accepted and that credit is granted fairly. The procedure described in [89]:

"This works as follows. Two or more instances of each job are created and dispatched to clients. A completed result specifies an amount of claimed credit (typically based on CPU time and CPU benchmarks). If both results are returned before their deadline, and the results agree (according to project-specified criteria [18]) then the result is considered correct, and both users receive a granted credit equal to the minimum of the claimed credits. This reduces the payoff for claiming more credit than is deserved.

If the results don't agree, or if one of the results is not reported by its deadline, the server generates an additional instance of the job, and sends it to a third host. This is repeated until a quorum of matching results is found or a limit on the number of instances is reached. Then, at some later point, the job's input and output files are deleted from the server, and eventually its database record is deleted."

Parameter	Description
name	Project name.
nscheduling_servers	Number of scheduling servers of the project.
ndata_servers	Number of data servers. If zero, scheduling servers also operate as data servers.
server_pw	CPU power of each server, in FLOPS.
disk_bw	Hard disk drive performance for each server, in bytes/s.
ifgl_percentage	Percentage of input files that must be generated locally on the client.
ifcd_percentage	Percentage of times a client must download new input files (due to locality scheduling).
input_file_size	Average amount of data that clients should download per workunit, in bytes.
output_file_size	Average amount of data that clients should upload per workunit, in bytes.
replication	Number of replicas of each file in the system.
task_fpops	Average task duration, in number of floating point operations needed to compute each task.
delay_bound	The time by which the result must be completed by the clients.
min_quorum	Minimum number of successful results required for the validator. If a strict majority agree, a consensus has been reached and the workunit is considered correct (there is a canonical result).
target_nresults	Number of results to create initially per workunit.
max_error_results	If the number of client error results exceed this, the workunit is declared to have an error.
max_total_results	If the number of results for this workunit exceeds this, the workunit is declared to be in error.
max_success_results	If the number of success results for this workunit exceeds this, and a consensus has not been reached, the workunit is declared to be in error.
success_percentage	Percentage of success results (when completed).
canonical_percentage	Percentage of success results that make up a consensus.

Table 4.1: *Project parameters.*

Credit is just a measure of how much work your computers have done. It does not have monetary value. 1-GigaFLOP machine (a machine that performs 10^9 Floating-point operations per second), running full time, produces 200 units of credit in 1 day [93]. Therefore, a 1-GigaFLOP machine, produces $0.0023\overline{148}$ units of credit in 1 second (Equation 4.1), or, in other words, the computation of 10^9 Floating-point operations produces $0.0023\overline{148}$ units of credit.

$$\frac{1\ (day)}{1\ (second)} = \frac{200\ (credits)}{x\ (credits)}\ \rightarrow\ x = \frac{200\ (seconds \cdot credits)}{86,400\ (seconds)} = 0.0023\overline{148}\ (credits) \tag{4.1}$$

Therefore, the execution of a task of N-Giga Floating-point operations ($N \times 10^9$ Floating-point operations) produces credits following Equation 4.2.

$$credits = N \cdot 0.0023\overline{148} \tag{4.2}$$

For instance, the SETI@home project [2] jobs need an average execution of 7,560 Giga Floating-point operations ($7.56 \cdot 10^{12}$ Floating-point operations) to complete. Applying Equation 4.2, the completed jobs should have an average of 18 credits each.

4.1.2 Client Side

In ComBoS, the client side is formed by groups of clients/Volunteer Nodes (VN). VN are used by the participants who join a BOINC-based project. Each VN group in ComBoS can be attached to any set of projects, and the client performs CPU scheduling among all runnable jobs. A VN is responsible for asking a project for more work, and scheduling the jobs of the different projects.

The BOINC client implements two related scheduling policies [89]:

- CPU scheduling: of the currently runnable jobs, which to run. Of the preempted jobs, which to keep in memory.
- Work fetch: when to ask a project for more work, which project to ask, and how much work to ask for.

These two policies have a considerable impact on the performance of BOINC-based projects. The scheduling is based on a round-robin between projects, weighted according to their resource share. This scheduling is described in detail in [89]. In this project, we have followed the same scheduling as the actual BOINC client. This scheduling policy includes the following preferences [89]:

- A *resource share* for each project. The fraction of a bottleneck resource R devoted to a project P should be roughly equal to P's resource share divided by the sum of resource shares of projects contending for R.
- *ConnectionInterval*: the typical time between periods of network activity. This lets users provide a "hint" about how often they connect, and it lets modem users tell BOINC how often they want it to automatically connect.

- *SchedulingInterval*: the "time slice" of the BOINC client CPU scheduler (the default is one hour).

The scheduling also depends on the physical characteristics of each host. In ComBoS, the power of each host is defined by the number of Floating-point Operations per Second (FLOPS). To simulate the power and availability of the hosts, ComBoS allows the use of statistical distribution functions. In addition, for even more realistic simulations, ComBoS allows the use of power and availability traces from an input file. Finally, each job in ComBoS has a number of associated parameters (as part of project parameters), including estimates of its number of floating-point operations, a list of input files, a deadline by which it should be reported, etc. The description of these parameters is found in Tables 4.1 and 4.2.

Parameter	Description
nclients	Number of VN of the group.
connection_interval	The typical time between periods of network activity.
scheduling_interval	The "time slice" of the BOINC client CPU scheduler (the default is one hour).
gbw	Bandwidth between each VN and the network backbone.
glatency	Latency between each VN and the network backbone.
traces_file	File with the VN power and availability traces (optional).
pv_distri	VN power fit distribution: Weibull, Gamma, Lognormal, Normal, Hyperexponential, Exponential (in case there is not a traces file).
max_power	Maximum power a VN might have using a random distribution.
min_power	Minimum power a VN might have using a random distribution.
av_distri	VN availability fit distribution: Weibull, Gamma, Lognormal, Normal, Hyperexponential, Exponential (in case there is not a traces file).
nv_distri	VN non-availability fit distribution: Weibull, Gamma, Lognormal, Normal, Hyperexponential, Exponential (in case there is not a traces file).
att_projs	Number of projects attached for each VN.
For each project:	
priority	Priority of the project, used by the client scheduler.
lsbw	Network bandwidth between the VN group and the scheduling servers of the project.
lslatency	Network latency between the VN group and the scheduling servers of the project.
ldbw	Network bandwidth between the VN group and the data servers of the project.
ldlatency	Network latency between the VN group and the data servers of the project.

Table 4.2: *VN group parameters.*

4.2 Local Scheduling Policies

In this section we describe the local scheduling policies used in ComBoS client. These scheduling policies solve the issues previously discussed. We have relied on the BOINC client scheduling described in [89].

4.2.1 Terminology

As we wanted ComBoS to simulate the real BOINC client scheduler, we also used the same terminology described in [89]:

A job J is nearly runnable if neither J nor its project is suspended, and J has not finished computing. A job J is runnable if, in addition, its input files have been downloaded. A project P is runnable if P has at least one runnable job; similarly for nearly runnable.

A project is contactable if the client is allowed to ask it for work (projects may be non-contactable for various reasons; for example, the client uses exponential back-off in response to Remote Procedure Call (RPC) failures). A project is potentially runnable if it is contactable or nearly runnable. A project's potentially runnable resource share is its resource share relative to the set of potentially runnable projects.

The wall CPU time of a process is the amount of wall-clock time it has been running at the OS level (the human perception of the passage of time). The CPU time may be significantly less, for example if the process does I/O or paging, or if CPU-intensive non-BOINC processes run at the same time.

4.2.2 Job Completion Time Estimation

In ComBoS, we estimate the remaining time of an unstarted job by simply dividing the number of floating-point operations (*task_fpops* parameter) of the job by the power of the volunteer host (in FLOPS).

4.2.3 Debt

The notion of debt is an important concept in the BOINC local scheduling policies. The debt to a project is the amount of wall CPU time owed to it, relative to other projects. The BOINC client scheduler uses two types of debt [89]:

- *Short-term debt*: it is the debt used by the CPU scheduler. It is varied over the set of runnable projects, and is bounded so that the maximum short-term debt is no greater than 86,400 seconds (one day).
- *Long-term debt*: it is used by the work-fetch policy. It is varied over the set of potentially runnable projects.

4.2.4 Simulating Weighted Round-Robin Scheduling

The BOINC client scheduling is based on a round-robin between projects, weighted according to their resource share. It produces the following outputs for each job J and project P [89]:

- *DeadlineMissed(J)*: whether J misses its deadline (*delay_bound* parameter).
- *DeadlinesMissed(P)*: the number of jobs J of P for which DeadlineMissed(J).
- *TotalShortfall*: the amount of additional work (measured in execution time) needed to keep the CPU busy for the next ConnectionInterval seconds.
- *Shortfall(P)*: the additional work (measured in CPU time) for project P needed to keep it from running short of work in the next ConnectionInterval seconds.

4.2.5 CPU Scheduling Policy

CPU scheduling consists of two policies [89]:

- *Job selection policy*: chooses the jobs and creates the run list. It uses Earliest Deadline First (EDF) scheduling, which allows the client to meet the deadlines that would otherwise be missed.
- *Enforcement policy*: attempts to run the jobs of the run list, while possibly postponing the preemption of jobs that haven't checkpointed recently (to avoid wasted CPU time). It selects the job to run. Let X be the set of jobs in the run list that are not currently running, and let Y be the set of running jobs that are not in the run list. The enforcement policy is as follows:

    1. If DeadlineMissed(J) for some job J in X, then preempt a job in Y, and run J (preempt the job with the least wall CPU time since checkpoint). Repeat as needed.

2. If there is a job J in Y that has checkpointed since the last call to this function, preempt J and run a job in X.

4.2.6 Work-Fetch Policy

The work fetch policy function runs periodically. It either decides to not fetch work, or selects a project to ask for work (Algorithm 5.2) [89]. This policy is included as a process within the client (see Figure 4-1).

―
Chapter 5

Implementation and Deployment

This chapter deals with the implementation and deployment of the software. Regarding the implementation of the system, the more complicated parts of the code are explained (Section 5.1, *Implementation*). On the other hand, we explain the steps required to deploy the final system (Section 5.2, *Deployment*).

5.1 Implementation

As we explained in Chapter 3, *Analysis*, we have implemented the simulator using the C programming language and the tools provided by SimGrid. The SimGrid core is responsible for planning the different processes, but the developer is the one that must use synchronization mechanisms to avoid race conditions. He have used several mutexes and condition variables in order to solve the problem of reading-writing in shared structures.

Furthermore, we have worked so that the project servers simulated in ComBoS have a realistic behavior. To do this, we have divided the functioning of each server (both scheduling and data servers) into two main processes: requests and dispatcher (see Figure 5-1).

The *requests* process is in charge of receiving all client requests through the corresponding mailbox. It receives messages asynchronously; that is, the process does not wait to finish receiving a message from a client in order to start receiving the next one. Each time a request is received, it is inserted into a queue that is shared with the *dispatcher* process. The dispatcher process is responsible for dealing with requests and answering to the volunteer clients if necessary. Response messages are sent asynchronously as well.

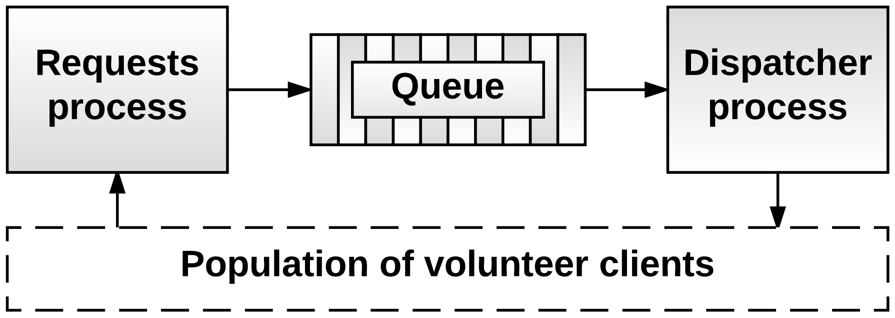


Figure 5-1: *Server main processes.*

On the other hand, each client is implemented with at least three different processes: the client main process (Pseudocode 5.1), which updates the client parameters every scheduling interval; the work fetch process (Pseudocode 5.2), which selects the project to ask for work; and the execution processes, one per attached project, that execute the tasks. We have not included the pseudocode of the execution process because it only dequeues tasks and executes them. Our simulator is complemented with the most important features of the real scheduler (deadline scheduling, long term debt, fair sharing and exponential back-off) as explained in Section 4.2, *Local Scheduling Policies* (Chapter 4, *Design*).

Pseudocode 5.1 Client main process

```
1: function CLIENT_MAIN( )
2:     while time < max_time do
3:         increase wall_cpu_time to the running project
4:         UPDATE_DEBT
5:         UPDATE_DEADLINE_MISSED
6:         CPU_SCHEDULING
7:         SIGNAL Work fetch process
8:         WAIT scheduling_interval
9:     end while
10:    Return
11: end function
```

Pseudocode 5.2 Work fetch process

```
function WORK_FETCH( )
    project = null
    while time < max_time do
        for each project $p$ in projects do
            if $p$ meets the requirements then
                project = $p$
            end if
        end for
        if project and not deadlines_missed then
            ASK_FOR_WORK(project)
        end if
        WAIT work_fetch_period
    end while
    SIGNAL Client main process
    Return
end function
```

5.2 Deployment

This section presents the deployment of the system. The technical specifications recommended for the final user to obtain the best experience from the application are:

- **Operating System**: Ubuntu 14.04.4 LTS (Linux distribution) or higher.
- **Processor**: Intel(R) Core(TM) i7 CPU 920 @2.67GHz or higher.
- **RAM**: 8 GB or higher.
- **Storage**: 1 GB of free space in the Hard Disk Drive.
- **Network**: Internet connection is not required.
- **Software**: The following software must be installed in order to run the application:

  1. GCC (GNU Compiler) 5.1 or higher.
  2. SimGrid toolkit 3.10 or higher.

To make life easier for the end user, we have organized the application files in the simplest way possible. Figure 5-2 shows the initial structure of the application files.

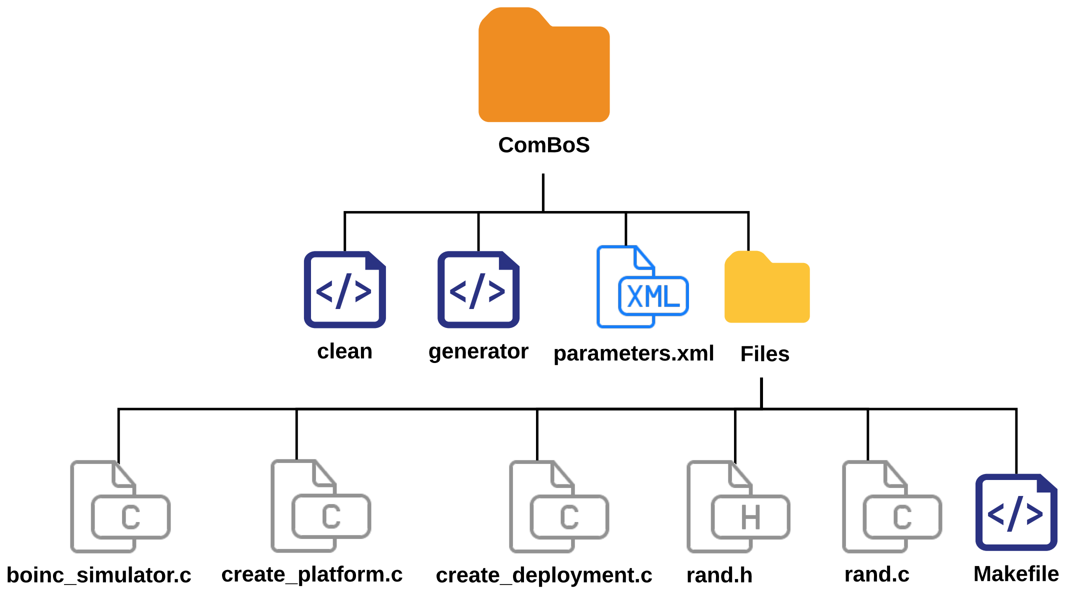


Figure 5-2: *Initial folder structure.*

The files located inside the ComBoS main folder are detailed below:

- **clean:** it is a script that removes the files created by the generator script.
- **generator:** it is a script that generates all the files needed for the simulation based on the parameters specified in the parameters.xml file. The files created by this script are:
  - **execute:** it is a script that executes the simulation (runs the file boinc_simulator).
  - **Files/create_platform:** it is the executable file that results from compiling the create_platform.c file.
  - **Files/create_deployment:** it is the executable file that results from compiling the create_deployment.c file.
  - **Files/platform.xml:** platform file created by create_platform.
  - **Files/deployment.xml:** deployment file created by create_deployment.
  - **Files/boinc_simulator:** it is the executable file that results from compiling the boinc_simulator.c and rand.c files.

- **parameters.xml:** to create a simulation, ComBoS requires the specification of all the parameters needed in an XML file, such as the simulation time in hours. All other parameters required by the XML file are detailed in Tables 4.1 and 4.2 (presented in Chapter 4, *Design*)
- **Files:** folder that includes:
  - **boinc_simulator.c:** simulator main source code.
  - **create_platform.c:** source code that contains the generation of the platform file.
  - **create_deployment.c:** source code that contains the generation of the deployment file.
  - **rand.h:** header file for random functions.
  - **rand.c:** source code that contains the random functions.
  - **Makefile:** script that compiles and links the code files.

The Appendix A presents a complete user manual of the simulator. It includes a tutorial for the installation of the SimGrid toolkit, and a number of practical and educational examples to learn how to perform simulations. In a basic way, in order to deploy the application, the user only needs to follow the following steps:

- Download the main folder of the ComBoS application (of which the structure is presented in Figure 5-2).
- Indicate the simulation parameters in the parameters XML file. This file must include all the parameters specified in Tables 4.1 and 4.2 (presented in Chapter 4, *Design*).
- Generate all the simulation files required, by just running the generator script.
- Run the execution script.

Chapter 6

Verification, Validation and Evaluation

This chapter details the verification, validation and evaluation of the project. First, we present the verification and validation of the simulator (Section 6.1, *Verification and Validation*), and we detail a series of tests that allowed us to verify that we had met all the requirements set in Chapter 3 (*Analysis*). After this, we show the validation of the outputs of the simulations, demonstrating that the simulator performs accurate and realistic simulations. We also display a study of the performance of the simulator (Section 6.2, *Performance Study*), in which we show that ComBoS is efficient and scalable. Finally, we present several case studies of the simulator usage (Section 6.3, *Case Studies*), with the corresponding analysis and evaluation of the results.

We have used the drand48 Linux functions [94] as random number generator in our simulations. These functions generate pseudo-random numbers using the linear congruential algorithm and 48-bit integer arithmetic. Each simulation result presented in this chapter is based on the average of 20 runs. For a 95% interval, the error is less than $\pm$ 3% for all values.

6.1 Verification and Validation

The main objective of this section is to verify that all the requirements set out in Chapter 3 (*Analysis*) have been fulfilled. In addition, we validate the results provided by ComBoS, comparing them to the results of SimBOINC and to the statistical results of the official BOINC webpage.

In software engineering, verification and validation are the processes of checking that a software system meets specifications and that it fulfills its intended purpose. As explained in Chapter 3 (*Analysis*), the customer initially sets the requirements desired for the final product (user requirements). From there, analysts specify software requirements (functional and non-functional requirements). In order to verify that the project requirements are met, verification and validation processes are needed (see Figure 6-1).

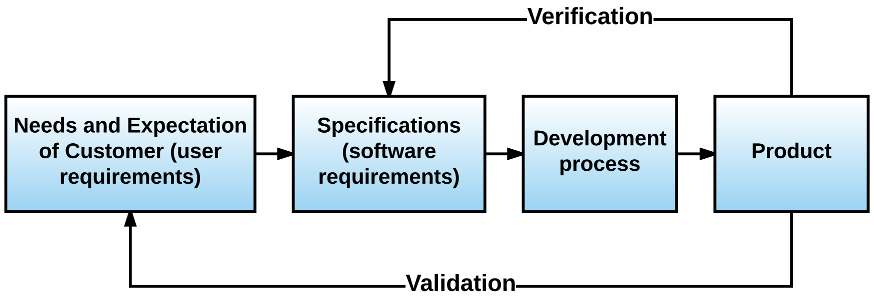

Verification
Needs and Expectation of Customer (user requirements)
Specifications (software requirements)
Development process
Product
Validation

Figure 6-1: *Software verification and validation.*

Software verification is the process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase (the software requirements). *Software validation* is the process of evaluating the final product at the end of the development process to determine whether it satisfies the requirements specified by the user at the beginning of the project [95].

6.1.1 Verification Tests

In order to perform the verification tests, we have followed a dynamic process during the development phase of the software. With these tests we wanted to answer the question: “Are we building the product right?”. Table 6.1 provides the template used for the verification tests. Note that the ID format is VET-XX, where XX indicates the verification test number.

ID	Test ID.
Name	Test name.
Requirements	Software requirements fulfilled with this test.
Description	Test description.
Preconditions	Predicates that must always be true before performing the test.
Procedure	A fixed, step-by-step sequence of activities performed by the test.
Postconditions	Predicates that must always be true just after performing the test.
Evaluation	*Passed* or *Failed*.

Table 6.1: *Template for verification tests.*

Then, we specify the verification tests.

ID	VET-01
Name	Platform.
Requirements	SR-NF-PL01, SR-NF-PL02, SR-NF-PL03, SR-NF-UI02.
Description	Verify that the software can be used on the platform and is developed with the tools specified in the requirements.
Preconditions	1. Use a machine with Ubuntu 14.04 operating system. 2. GCC (GNU Compiler) 5.1 or higher must be installed on the machine. 3. The user must be located in the main directory of the ComBoS application.
Procedure	1. Check the code of the simulator source files (all these files are inside the /Files folder). 2. Run the generator script with the default parameters to create the simulation files. 3. Run the simulation.
Postconditions	1. All source files must be written in C programming language. 2. The implementation, setup and control of the simulations must be carried out using the MSG API of the SimGrid toolkit. 3. Simulations run while a progress bar indicates the percentage of execution. 4. The simulator must successfully finish its execution in the specified operating system.
Evaluation	Passed

Table 6.2: *Verification test VET-01.*

ID	VET-02
Name	Realistic BOINC elements in simulations.
Requirements	SR-F-F06, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the simulator allows to simulate all the BOINC actual elements.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the following elements in the simulation parameters: tasks, volunteer hosts, servers, data servers, networks, and hosts availability. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution.
Evaluation	Passed

Table 6.3: *Verification test VET-02.*

ID	VET-03
Name	Statistics of BOINC projects.
Requirements	SR-F-F02, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the outputs of the simulator are the same as those published by BOINCstats [4]. The outputs are: credits, hosts, active hosts, and FLOPS.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution. 3. The outputs of the simulator contain at least: credits, hosts, active hosts, and FLOPS (The same as those published by BOINCstats [4]).
Evaluation	Passed

Table 6.4: *Verification test VET-03.*

ID	VET-04
Name	Multiple BOINC proyects simultaneously.
Requirements	SR-F-F04, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the simulator allows multiple project simulations simultaneously.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters of three different projects (for example, the SETI@home, Einstein@home, and LHC@home projects). 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution.
Evaluation	Passed

Table 6.5: *Verification test VET-04.*

ID	VET-05
Name	BOINC client scheduler.
Requirements	SR-F-F05, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the client scheduler implemented produces the same results as the actual BOINC scheduler.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the client side simulation parameters of three different projects: SETI@home, Einstein@home, and LHC@home. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution. 3. The outputs of the simulation are the same as the real BOINC client scheduler (it is detailed in 6.1.3, *Validation of the Client Scheduler*).
Evaluation	Passed

Table 6.6: *Verification test VET-05.*

ID	VET-06
Name	Accurate simulations of BOINC projects.
Requirements	SR-F-F01, SR-F-F03, SR-NF-UI01, SR-NF-UI02.
Description	Verify that the outputs of the simulator for existing projects (SETI@home, Einstein@home, and LHC@home) should be almost identical to those published in BOINCstats [4].
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters of three different projects: SETI@home, Einstein@home, and LHC@home. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution. 3. The outputs of the simulation are the same as the actual BOINC projects, in terms of FLOPS and credit (it is detailed in 6.1.4, *Validation of Whole Simulator*).
Evaluation	Passed

Table 6.7: *Verification test VET-06.*

ID	VET-07
Name	Large simulations.
Requirements	SR-NF-S01, SR-NF-UI01, SR-NF-UI02.
Description	Verify the application is able to perform simulations with more than 100,000 hosts in a machine with at least 8 GB of RAM.
Preconditions	1. Use a machine with at least 8GB or RAM. 2. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters with more than 100,000 hosts. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution.
Evaluation	Passed

Table 6.8: *Verification test VET-07.*

ID	VET-08
Name	Execution time.
Requirements	SR-NF-P01, SR-NF-UI01, SR-NF-UI02.
Description	Check that simulations follow a linear execution time.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters with different workloads. 2. Run the generator script to create the simulation files. 3. Run the simulation. 4. Go to 2 specifying different simulation parameters.
Postconditions	1. Simulations run while a progress bar indicates the percentage of execution. 2. The simulator must successfully finish its execution. 3. Check that executions follow a linear execution time when increasing the workload (it is detailed in 6.2, *Performance Study*).
Evaluation	Passed

Table 6.9: *Verification test VET-08.*

The verification test traceability matrix (Table 6.10) determines that all the software requirements have been verified during the development phase of the project.

Requirements	**VET-01**	**VET-02**	**VET-03**	**VET-04**	**VET-05**	**VET-06**	**VET-07**	**VET-08**
SR-F-F01						✓		
SR-F-F02			✓					
SR-F-F03						✓		
SR-F-F04				✓				
SR-F-F05					✓			
SR-F-F06		✓						
SR-NF-PL01	✓							
SR-NF-PL02	✓							
SR-NF-PL03	✓							
SR-NF-S01							✓	
SR-NF-P01								✓
SR-NF-UI01		✓	✓	✓	✓	✓	✓	✓
SR-NF-UI02	✓	✓	✓	✓	✓	✓	✓	✓

Table 6.10: *Verification test traceability matrix.*

6.1.2 Validation Tests

To perform the validation tests, we have checked the final software, comparing it with the user needs specified in Chapter 3 (*Analysis*). With these tests we want to answer the question: "Have we built the right product?". Table 6.11 provides the template used for the validation tests. Note that the ID format is VAT-XX, where XX indicates the validation test number.

ID	Test ID.
Name	Test name.
Requirements	User requirements fulfilled with this test.
Verification tests	Verification tests that help us to validate this test.
Description	Test description.
Preconditions	Predicates that must always be true before performing the test.
Procedure	A fixed, step-by-step sequence of activities performed by the test.
Postconditions	Predicates that must always be true just after performing the test.
Evaluation	*Passed* or *Failed*.

Table 6.11: *Template for validation tests.*

Then, we specify the validation tests.

ID	VAT-01
Name	BOINC projects simulation.
Requirements	UR-C01.
Verification tests	VET-03, VET-06.
Description	Validate that the simulator is able to simulate the behavior of BOINC projects.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters of three different BOINC projects: SETI@home, Einstein@home, and LHC@home. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution. 2. The outputs of the simulation are the same as the actual BOINC projects, in terms of FLOPS and credit (it is detailed in 6.1.4, *Validation of Whole Simulator*).
Evaluation	Passed.

Table 6.12: *Validation test VAT-01.*

ID	VAT-02
Name	Client scheduling.
Requirements	UR-C02.
Verification tests	VET-05.
Description	Validate that the client scheduler implemented produces the same results as the actual BOINC scheduler.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the client side simulation parameters of three different projects: SETI@home, Einstein@home, and LHC@home. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution. 2. The outputs of the simulation are the same as the real BOINC client scheduler (it is detailed in 6.1.3, *Validation of the Client Scheduler*).
Evaluation	Passed.

Table 6.13: *Validation test VAT-02.*

ID	VAT-03
Name	Simulation components.
Requirements	UR-C03.
Verification tests	VET-02.
Description	Validate that the simulations cover all the elements of the BOINC infrastructure.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the following elements in the simulation parameters: tasks, volunteer hosts, servers, data servers, networks, and hosts availability. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution.
Evaluation	Passed.

Table 6.14: *Validation test VAT-03.*

ID	VAT-04
Name	Platform.
Requirements	UR-R01, UR-R02.
Verification tests	VET-01.
Description	Validate that the software can be used on the platform and is developed with the tools specified in the requirements.
Preconditions	1. Use a machine with a Linux operating system. 2. The user must be located in the main directory of the ComBoS application.
Procedure	1. Check the code of the simulator source files (all these files are inside the /Files folder). 2. Run the generator script with the default parameters to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution. 2. The implementation, setup and control of the simulations must be carried out using the MSG API of the SimGrid toolkit.
Evaluation	Passed.

Table 6.15: *Validation test VAT-04.*

ID	VAT-05
Name	Scalability.
Requirements	UR-R03.
Verification tests	VET-07.
Description	Validate that the application is able to perform large simulations.
Preconditions	1. The user must be located in the main directory of the ComBoS application.
Procedure	1. Specify the simulation parameters with more than 100,000 hosts. 2. Run the generator script to create the simulation files. 3. Run the simulation.
Postconditions	1. The simulator must successfully finish its execution.
Evaluation	Passed.

Table 6.16: *Validation test VAT-05.*

The validation test traceability matrix (Table 6.17) determines that all the user needs have been validated in the final product.

Requirements	**VAT-01**	**VAT-02**	**VAT-03**	**VAT-04**	**VAT-05**
UR-C01	✓				
UR-C02		✓			
UR-C03			✓		
UR-R01				✓	
UR-R02				✓	
UR-R03					✓

Table 6.17: *Validation test traceability matrix.*

6.1.3 Validation of the Client Scheduler

To validate the client scheduler of ComBoS, we have compared the results of different executions of the simulator with the equivalent SimBOINC simulations. Of course, as we only want to validate the client scheduler (individually), we have simulated scenarios with no delay caused by network or servers.

All scenarios considered are based on a single client host with three associated projects (Einstein@home, SETI@home and LHC@home). Through the different tests we have varied the priorities of the projects and the time of each simulation. When using hosts with the same power, our goal is to compare the number of tasks executed in each simulator.

As explained in Section 2.3, *Volunteer Computing Simulators* (Chapter 2, *State of the Art*), SimBOINC simulates the BOINC client scheduler and its simulations are highly accurate, because it uses almost exactly the BOINC client's CPU scheduler source code. Tables 6.18, 6.19 and 6.20 show different test cases:

- Table 6.18 presents the number of tasks executed by a client host of $1.4 \cdot 10^9$ FLOPS on simulations of 100, 500, 1,000, 5,000, and 10,000 hours. The priorities of the three projects are the same, so that each project uses the same runtime (33% CPU). The results of ComBoS and SimBOINC are almost identical.

	SimBOINC			*ComBoS*		
Time in hours	Einstein@home (33%)	SETI@home (33%)	LHC@home (33%)	Einstein@home (33%)	SETI@home (33%)	LHC@home (33%)
100	1	21	33	1	22	28
500	7	108	166	7	112	163
1,000	14	220	331	13	223	333
5,000	70	1,103	1,652	70	1,106	1,659
10,000	139	2,214	3,319	139	2,221	3,331

Table 6.18: *Executed tasks (three projects running on a single host of 1.4 GigaFLOPS).*

- Table 6.19 proposes a case similar to the previous test. In this case, the host has a power of $5.5 \cdot 10^9$ FLOPS and the priorities of the projects differ. The tasks of the LHC@home project consume 50% of CPU usage, while the tasks of the Einstein@home and SETI@home projects consume 25% of the CPU usage each. As in the previous case, the number of tasks executed in ComBoS is practically the same as in the case of SimBOINC.

	SimBOINC			*ComBoS*		
Time in hours	Einstein@home (25%)	SETI@home (25%)	LHC@home (50%)	Einstein@home (25%)	SETI@home (25%)	LHC@home (50%)
100	4	67	181	4	64	182
500	21	332	975	21	333	975
1,000	42	662	1,955	40	662	1,981
5,000	208	3,297	9,831	206	3,297	9,889
10,000	416	6,581	19,637	413	6,593	19,784

Table 6.19: *Executed tasks (three projects running on a single host of 5.5 GigaFLOPS).*

- Table 6.20 includes three different test cases. In each test case, a host of FLOPS $5.5 \cdot 10^9$ runs a unique project (100% of the CPU time). In the first case, the host performs tasks of Einstein@home project and the results are exactly the same in both simulators. In the case of SETI@home and LHC@home projects the results vary minimally.

	Einstein@home(100%)		*SETI@home*(100%)		*LHC@home*(100%)	
Time in hours	SimBOINC	ComBoS	SimBOINC	ComBoS	SimBOINC	ComBoS
100	16	16	263	263	395	394
500	82	82	1,318	1,319	1,978	1,972
1,000	164	164	2,637	2,639	3,956	3,945
5,000	824	824	13,177	13,195	19,780	19,728
10,000	1,649	1,649	26,315	26,390	39,473	39,457

Table 6.20: *Executed tasks (single project running on a single host of 5.5 GigaFLOPS).*

If we consider only the client scheduler, ComBoS results match those of SimBOINC, demonstrating the proper functioning of the simulator in this regard.

6.1.4 Validation of Whole Simulator

To validate the complete simulator, we have relied on data from the BOINCstats website [4], which provides official statistical results of BOINC projects. In this section, we analyze the behavior of ComBoS considering the simulation results of the SETI@home, Einstein@home and LHC@home projects.

We have used the CPU power traces of the client hosts that make up the VN of each project [96, 97, 98]. We have not used any other traces. In order to model the availability and unavailability of the hosts, we used the results obtained in [99]. This research analyzed about 230,000 hosts' availability traces obtained from the SETI@home project. According to this paper, 21% of the hosts exhibit truly random availability intervals, and it also measured the goodness of fit of the resulting distributions using standard probability-probability (PP) plots. For availability, the authors saw that in most cases the Weibull distribution is a good fit. For unavailability, the distribution that offers the best fit is the log-normal. The parameters used for the Weibull distribution are $shape = 0.393$ and $scale = 2.964$. For the log-normal, the parameters obtained and used in ComBoS are a distribution with mean $\mu = -0.586$ and standard deviation $\sigma = 2.844$. All these parameters were obtained from [99] too. For the network parameters, we have used the bandwidth and latency values of current ADSL networks, and 10 Gbps for the network backbone. SimGrid's models allow us to adjust this network values. We have obtained all the other parameters of the simulations from the official websites of the SETI@home, Einstein@home, and LHC@home projects.

			BOINCstats		*ComBoS*	
Project	Total hosts	Active hosts	*GigaFLOPS*	*Credit/day*	*GigaFLOPS*	*Credit/day*
SETI@home	3,970,427	175,220	864,711	171,785,234	865,001	168,057,478
Einstein@home	1,496,566	68,338	1,044,515	208,902,921	1,028,172	205,634,486
LHC@home	356,942	15,814	7,521	1,504,214	7,392	1,393,931

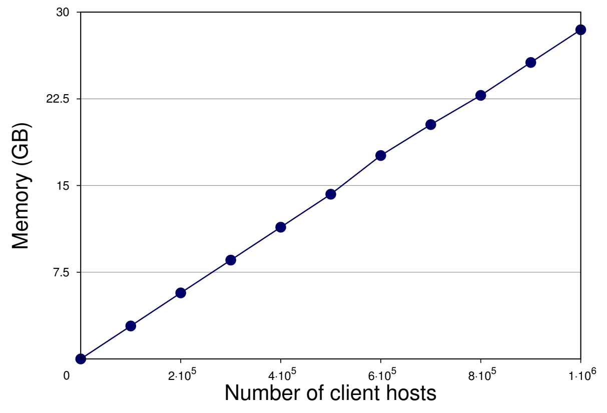
Table 6.21: *Validation of the whole simulator.*

Table 6.21 compares the actual results of the SETI@home, Einstein@home and LHC@home projects with those obtained with ComBoS in terms of GigaFLOPS and credits. The error obtained is 2.2% for credit/day and 0.03% for GigaFLOPS compared to the SETI@home project; 1.6% for credit/day and for GigaFLOPS compared to the Einstein@home project; and 7.3% for credit/day and 1.7% for GigaFLOPS compared to the LHC@home project. We consider that these results allow us to validate the whole simulator.

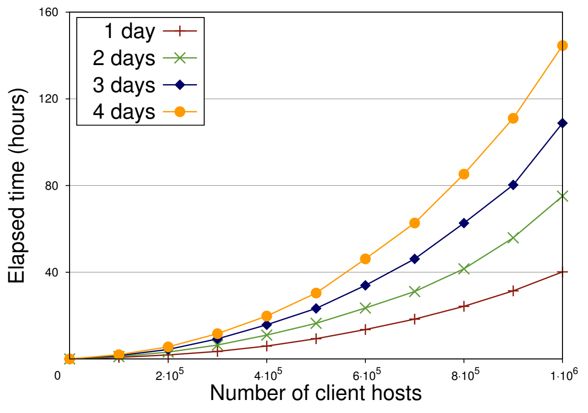
6.2 Performance Study

In this section we analyze the performance of the simulator in terms of memory usage and execution time. All measurements were made on a computer with 32 GB of RAM and 8 Intel Core i7 processors running at 2.67 Ghz each. The server runs the Linux 3.13.0-85-generic kernel. It runs the 3.10 version of the SimGrid toolkit. In spite of the computer has eight cores, each simulation was performed individually in a single core.

Figure 6-2a shows the memory usage of the simulator and Figure 6-2b shows the execution time by increasing the number of client hosts in each simulation. Note that the tests have been carried out up to 1 million hosts, the same number of active hosts of all BOINC projects together. Figure 6-2a shows a linear ($O(n)$) memory footprint. Figure 6-2b shows the execution time of the simulator for four different simulation times: 1 day, 2 days, 3 days and 4 days. Both metrics demonstrate that the simulator is highly scalable. This has been possible due to the high performance [100] of the SimGrid toolkit.



(a) *Memory usage.*



(b) *Execution time.*

Figure 6-2: *Performance study.*

6.3 Case Studies

In this section we will present different case studies using ComBoS. Our goal is to show the performance that would result when volunteer computing platforms are used to process big amounts of data. We are especially interested in analyzing bottlenecks and limits that an architecture like BOINC presents. We show a few practical examples of the simulator usage, with the subsequent analysis of the execution results.

Figure 6-3 represents the scenario used in the evaluation, which consists of a single project (one scheduling server (Server), and different data servers (Data Servers)) and three groups of Volunteer Nodes (Group 1, Group 2 and Group 3). Networks that connect each group of VN with the servers are fixed in all simulations. In each test case, we specify the number of VN per group, the number of data servers, the size of the input files, and the duration of the tasks.

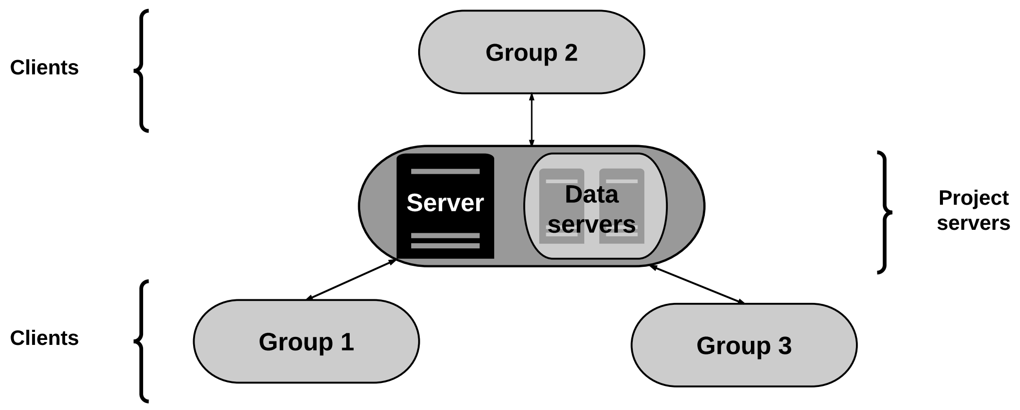
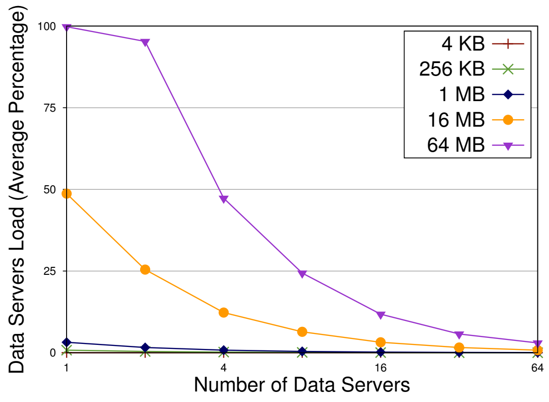


Figure 6-3: *Simplified scenario used in the case studies.*

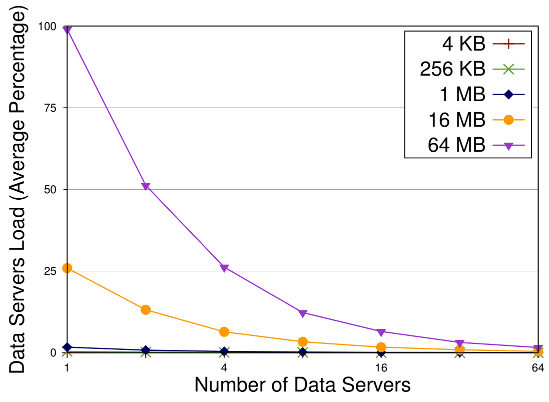
All other parameters are fixed in all simulations. The size of the output files has not been taken into account in the simulations. Every execution in this section has simulated 100 hours.

6.3.1 Data Servers Load

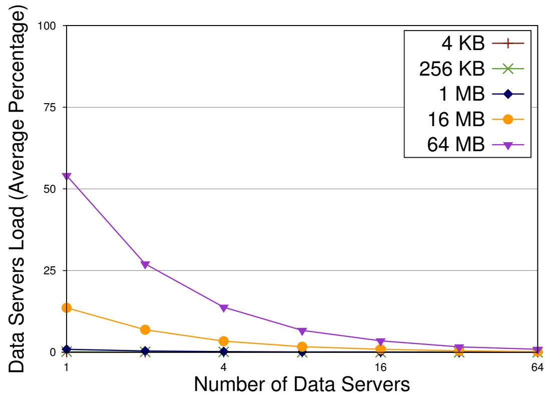
We have divided our tests into two groups: the first one (Figure 6-4), in which we consider 3,000 VN (1,000 per group); and the second one (Figure 6-5), in which we consider 30,000 VN (10,000 per group).



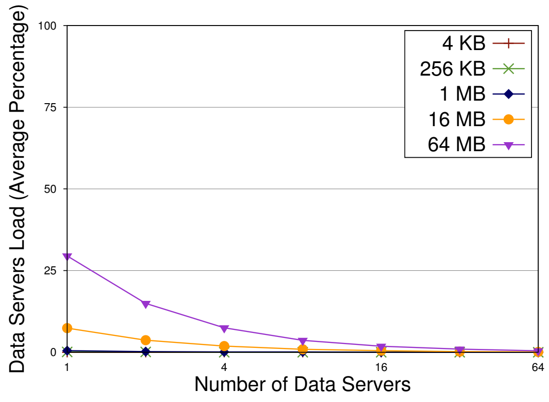
(a) *Task duration: 3.6* $\cdot 10^{12}$ *fpops.*



(b) *Task duration: 7.2* $\cdot 10^{12}$ *fpops.*



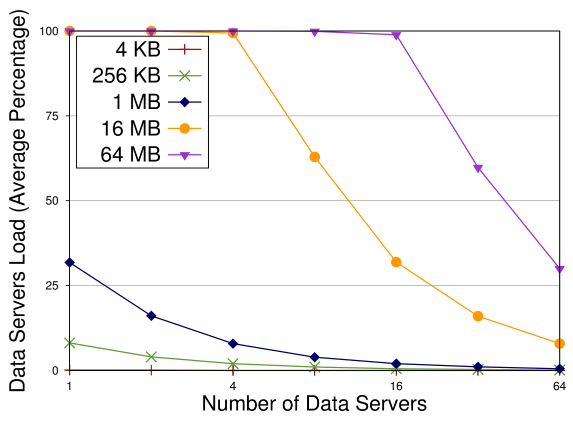
(c) *Task duration: 1.4* $\cdot 10^{13}$ *fpops.*



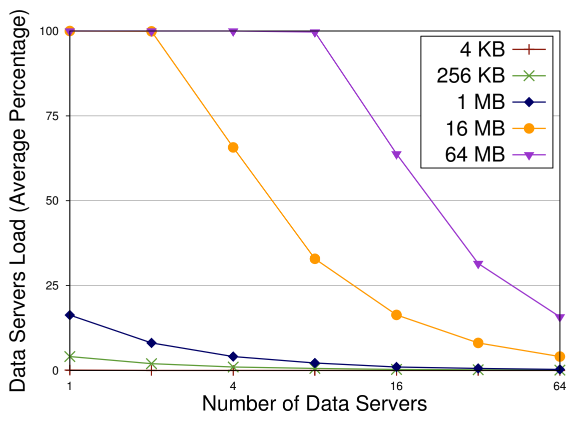
(d) *Task duration: 2.9* $\cdot 10^{13}$ *fpops.*

Figure 6-4: *Data servers load with 3,000 VN (1,000 VN per group).*

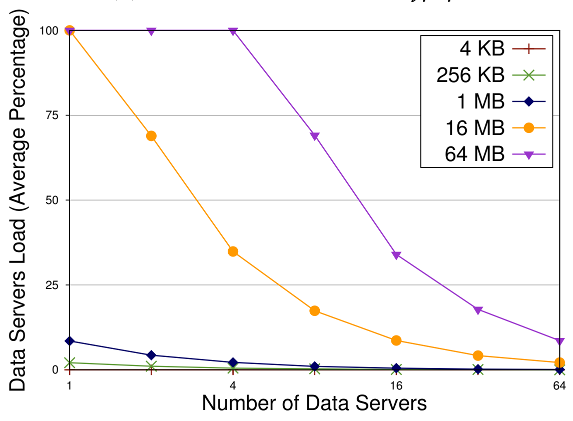
In these experiments, our aim was to analyze the average load of the data servers using a different number of data servers in each simulation (from 1 to 64). We collected results from five input file sizes: 4 KB, 256 KB, 1 MB, 16 MB and 64 MB. The average number of floating-point operations required to complete the computation of each task (fpops) in Figures 6-4a and 6-5a are $3.6 \cdot 10^{12}$ (1 hour for a 1 GigaFLOP machine), $7.2 \cdot 10^{12}$ in Figures 6-4b and 6-5b (2 hours for a 1 GigaFLOP machine), $1.4 \cdot 10^{13}$ in Figures 6-4c and 6-5c (4 hours for a 1 GigaFLOP machine), and $2.9 \cdot 10^{13}$ in Figures 6-4d and 6-5d (8 hours for a 1 GigaFLOP machine). The work tends to be distributed among all servers and the load of the scheduling servers has not been remarkable in any simulation.



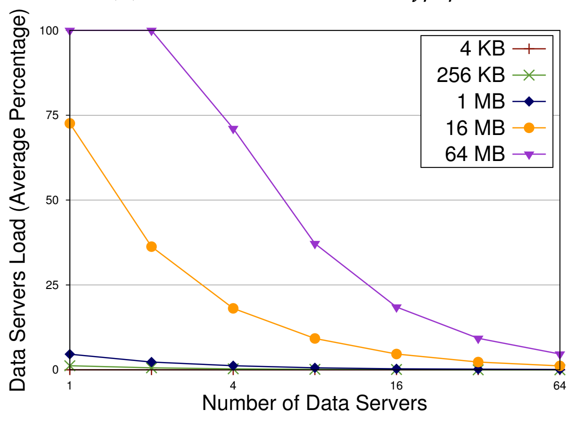
(a) *Task duration: 3.6 ·10^{12} fpops.*



(b) *Task duration: 7.2 ·10^{12} fpops.*



(c) *Task duration: 1.4 ·10^{13} fpops.*



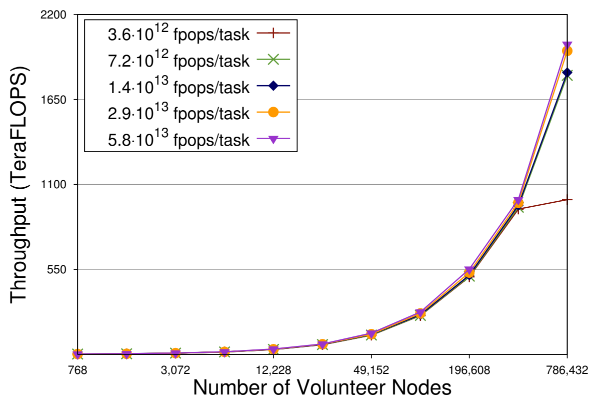
(d) *Task duration: 2.9 ·10^{13} fpops.*

Figure 6-5: *Data servers load with 30,000 VN (10,000 VN per group).*

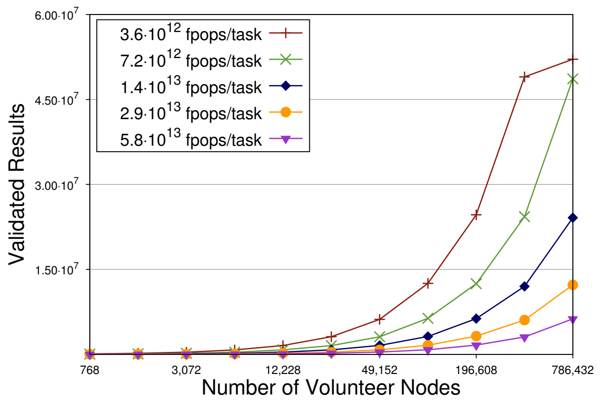
Using these results, we can estimate the average load of data servers in real scenarios. The larger the size of each input file and the lower the task duration, the greater the data servers' load becomes. Thus, in some simulations data servers turn into the bottleneck of the system. This type of simulations would help designers to verify the feasibility of BOINC projects. In this way, ComBoS can guide the design of BOINC projects.

6.3.2 Combined Results

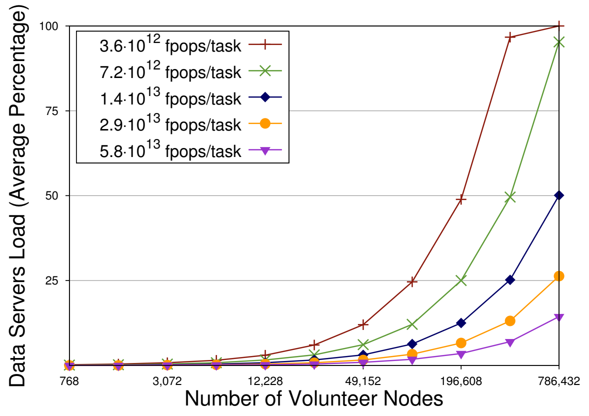
In the previous experiment we studied the scalability of the simulated environment by varying the duration of the tasks, the size of the input files, and the number of data servers. Now, we want to analyze the performance of the same infrastructure by increasing the number of VN, and setting the number of data servers to 4 and the input file size to 1 MB. The results of these simulations are shown in Figure 6-6.



(a) *Throughput.*



(b) *Validated results.*



(c) *Data servers load.*

Figure 6-6: *Combined results.*

The results obtained show that we can estimate the system's throughput (Figure 6-6a, constant for the same number of floating point operations executed) and the number of validated results (Figure 6-6b, inversely proportional to the task duration). In addition, as in the previous experiment, we can find out the load of the data servers (Figure 6-6c). As shown in Figure 6-6c, for $3.6 \cdot 10^{12}$ fpops per task, data servers get saturated when there are about 200,000 VN, causing a severe deceleration in throughput (Figure 6-6a) and validated results (Figure 6-6b).

Chapter 7

Planning and Budget

This chapter presents a detailed planning of the project (Section 7.1, *Planning*). Then, we explain the project costs (Section 7.2, *Budget*). At the end of the chapter, we comment on the socio-economic environment of the project (Section 7.3, *Socio-Economic Environment*).

7.1 Planning

This section includes the complete project planning. First, we describe the software development methodology used. After that, we detail the time duration of each phase of the project, collecting all times in a Gantt chart.

7.1.1 Justification of the Methodology

Due to its characteristics, we have divided our project into three iterations:

- **Basic functionality:** the first iteration has been to achieve the simulation of a simple distributed computing system. The aim of this phase has been to simulate client machines that exchange messages with a server through the a network.
- **Client side:** this phase has been to incorporate all the necessary functionality on the client side (described in Chapter 4, *Design*).
- **Server side:** this phase has been to incorporate all the necessary functionality on the server side (described in Chapter 4, *Design*).

It was necessary to have an iterative methodology used to develop each of the phases independently to join all together in the last stage and obtain the final product. For this purpose, we have analyzed three different software development methodologies: Software prototyping [101], the Waterfall model [102] and the Spiral model [103]. Software prototyping did not fit well because it requires building a prototype of the software in a short time. The Waterfall model is a sequential design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through different phases. The problem with this methodology is that it does not allow iterations within the software development. Finally, the Spiral model allowed fragmenting the project into different iterations. The model combines the strengths of the other two models (simplicity and flexibility), and uses an iterative process. Although this model is slower than the other two, it allowed us to apply different iterations so we decided to apply it to the whole process.

7.1.2 Life Cycle

The life cycle development process of the project has followed the Spiral lifecycle model [103]. Figure 7-1 shows the Spiral model using a scheme.

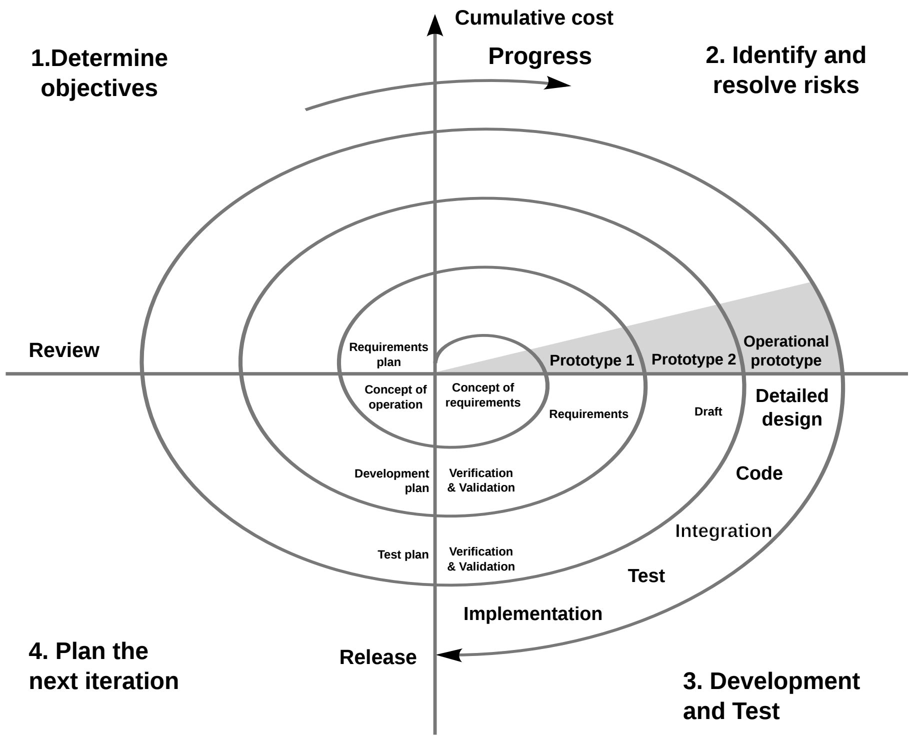


Figure 7-1: *Spiral model (Boehm, 2000).*

The Spiral model has four phases, which are repeated during the different iterations of the model. These phases are:

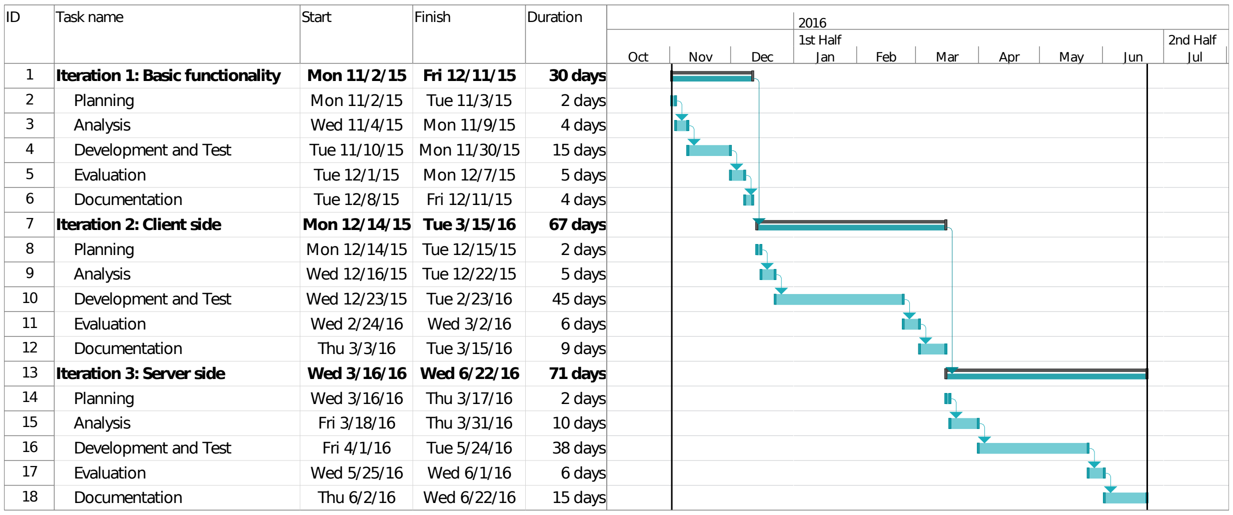
- **Planning** (Determine objectives in Figure 7-1): the user requirements are gathered, a feasibility study of the system is performed, and the iteration objectives are determined.
- **Analysis** (Identify and resolve risks in Figure 7-1): a full analysis of requirements is done and the potential risks are identified. This phase ends with a basic design.
- **Development and Test**: Code implementation is done. Test cases and test results are performed.
- **Evaluation** (Plan the next iteration in Figure 7-1): Customers evaluate the software and provide their feedback. In this case, the student tries to get the supervisor's approval. This is the *critical task* of the life cycle, since we can only move on to the next iteration of the Spiral lifecycle model if this task is approved.

Each phase starts with a design goal and ends with the customer (the supervisor) reviewing the progress so far. As previously explained, we have divided the software development into three iterations: basic functionality, client side, and server side. In the last iteration, the complete software must undergo extensive testing in order to validate the simulator.

7.1.3 Time Estimation

The Gantt chart (Figure 7-2) shows all the tasks carried out during the project development. This project has been developed within a Collaboration in University Departments Scholarship [104], funded by the Spanish Ministry of Education, Culture, and Sport. The project began on November 2st, 2015, and ended on June 22, 2016, making a total of almost eight months of work. During this time, I have worked from Monday to Friday, four hours a day.

The Gantt chart shows all the tasks performed in each iteration of the spiral lifecycle model. Recall that the three iterations were: Basic functionality, Client side and Server side. In addition to the tasks (phases) mentioned above (Planning, Analysis, Development and Test, and Evaluation), we have included the Documentation task at the end of each iteration. The Documentation task has consisted mainly in drafting this bachelor thesis.



ID	Task name	Start	Finish	Duration
1	**Iteration 1: Basic functionality**	**Mon 11/2/15**	**Fri 12/11/15**	**30 days**
2	Planning	Mon 11/2/15	Tue 11/3/15	2 days
3	Analysis	Wed 11/4/15	Mon 11/9/15	4 days
4	Development and Test	Tue 11/10/15	Mon 11/30/15	15 days
5	Evaluation	Tue 12/1/15	Mon 12/7/15	5 days
6	Documentation	Tue 12/8/15	Fri 12/11/15	4 days
7	**Iteration 2: Client side**	**Mon 12/14/15**	**Tue 3/15/16**	**67 days**
8	Planning	Mon 12/14/15	Tue 12/15/15	2 days
9	Analysis	Wed 12/16/15	Tue 12/22/15	5 days
10	Development and Test	Wed 12/23/15	Tue 2/23/16	45 days
11	Evaluation	Wed 2/24/16	Wed 3/2/16	6 days
12	Documentation	Thu 3/3/16	Tue 3/15/16	9 days
13	**Iteration 3: Server side**	**Wed 3/16/16**	**Wed 6/22/16**	**71 days**
14	Planning	Wed 3/16/16	Thu 3/17/16	2 days
15	Analysis	Fri 3/18/16	Thu 3/31/16	10 days
16	Development and Test	Fri 4/1/16	Tue 5/24/16	38 days
17	Evaluation	Wed 5/25/16	Wed 6/1/16	6 days
18	Documentation	Thu 6/2/16	Wed 6/22/16	15 days

Figure 7-2: *Gantt chart.*

7.2 Budget

This section details the overall project budget. On the one hand, we present the project costs and, on the other hand, we disclose the offer presented to the customer.

7.2.1 Project Costs

Table 7.1 summarizes the main features of the project including the total budget.

Project Information	
Title	A Complete Simulator for Volunteer Computing Environments
Author	Saúl Alonso Monsalve
Department	Computer Science and Engineering Department
Start date	2nd of November of 2015
End date	22nd of June of 2016
Duration	8 months
Indirect costs ratio	20 %
Total budget	30,526.49

Table 7.1: *Project Information.*

Then the total budget of the project is broken down below.

Direct Costs

In this part, the direct costs of the project are presented. Table 7.2 shows the direct costs caused by personnel costs, based on the planning presented in the previous section. The supervisor and the student have played the following roles:

- **Supervisor:** Project manager.
- **Student:** Analyst, Developer, Tester.

Category	**Cost per hour (€)**	**Hours**	**Total (€)**
Project manager	60	56	3,360
Analyst	35	188	6,580
Developer	35	316	11,060
Tester	25	112	2,800
Total			**23,800.00**

Table 7.2: *Human resources costs.*

Table 7.3 shows the direct costs caused by equipment acquisition and usage. The chargeable cost, C, is calculated using the following formula:

$$C = \frac{d \cdot c \cdot u}{D} \tag{7.1}$$

Where:

- **C:** Chargeable cost. It is equivalent to the depreciated value.
- **d:** Time the equipment has been used.
- **c:** Equipment cost.
- **u:** Project dedication. Percentage of time the equipment has been used.
- **D:** Equipment depreciation period.

Concept	**Cost, c (€)**	**Dedication, u (%)**	**Dedication, d (months)**	**Depreciation, D (months)**	**Chargeable cost, C (€)**
Desktop PC	799.99	100	8	36	177.78
Laptop	529.99	25	8	36	29.44
ARCOS Tucan	89,501.60	10	6	60	895.02
ARCOS Mirlo	2,469.99	70	6	60	172.90
Printer	399.24	5	3	60	1.00
Total					**1,276.14**

Table 7.3: *Equipment costs.*

Furthermore, the equipment presented in Table 7.3 is detailed below:

- **Desktop PC:** All in One - Asus Z220ICUK, 21.5”, i5-6400T, 8GB, 1TB)
- **Laptop:** Toshiba L50D-C-19D, A10-8700P, 8GB RAM and 1TB.
- **ARCOS Tucan:** Cluster used by the research group ARCOS.
- **ARCOS Mirlo:** Server used by the research group ARCOS. 32GB RAM and eight i7 processors of 2.67GHz each.
- **Printer:** HP LaserJet Enterprise P3015.

Other direct costs are shown in Table 7.4. These costs consist of office material, a toner for the printer, and the monthly travel pass. Office material includes: pencils, pens, notebooks, paper, tipex, and markers.

Concept	**Cost (€)**
Office material	112.98
Toner (x1)	89.62
Monthly travel pass (x8)	160
Total	**362.60**

Table 7.4: *Other direct costs.*

Costs summary

Table 7.5 shows the complete summary of the project costs. Indirect costs (20% of direct costs) consist of the electricity and water bills, telephone, Internet access, etc.

Costs summary	
Human resources	23,800.00
Equipment	1,276.14
Other direct costs	362.60
Indirect costs	5,087.75
Total budget	**30,526.49**

Table 7.5: *Costs summary.*

The total budget for this project amounts to **30,526.49 € (thirty thousand five hundred twenty-six euro and forty-nine cent)**.

7.2.2 Project Offer Proposal

Table 7.6 shows a detailed offer proposal. This offer includes the estimated risks (20%), the expected benefits (15%), and the Value Added Tax (Spanish Impuesto Sobre el Valor Añadido (IVA)), which corresponds to 21% [105]. After applying all theses concepts, the final amount for this project in case of sale to a third-party client is **50,973.14 € (fifty thousand nine hundred seventy-three euro and fourteen cent).**

Offer proposal			
Concept	**Increment (%)**	**Partial value (€)**	**Aggregated cost (€)**
Project costs	-	30,526.49	30,526.49
Risk	20	6,105.30	36,631.79
Benefits	15	5,494.77	42,126.56
IVA	21	8,846.58	50,973.14
Total			**50,973.14**

Table 7.6: *Offer proposal.*

7.3 Socio-Economic Environment

As commented in previous chapters, ComBoS can guide the design of BOINC projects. This means that BOINC project designers can perform accurate simulations using ComBoS before deploying the system. Thanks to this, designers can save money and resources, because they will know the performance of the system before deploying it. In addition, it can also save energy because designers will not need to perform tests using the original infrastructure, as they will only need to use ComBoS in order to analyze the functioning of different alternatives.

Moreover, BOINC operates as a platform for distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, environmental science, and astrophysics. On the one hand, there are projects that help the scientific community, such as the SETI@home project [2], of which the purpose is to analyze radio signals, searching for signs of extraterrestrial intelligence; or the Citizen Science Grid project [106], which is dedicated to supporting a wide range of research and educational projects. On the other hand, there are projects dedicated to the environmental care, such as the Climateprediction.net project [37], which studies climate. Therefore, our simulator indirectly contributes to both science and the environment.

Chapter 8

Conclusions and Future Work

In this chapter we discuss the main contributions of our work. In addition, we present the conclusions of the work, revise the objectives set at the beginning of this document, and include some personal conclusions. Finally, we discuss future work.

8.1 Contributions

The project carried out during this bachelor thesis fits with many of the subjects studied in the Degree of Computer Science and Engineering of the Carlos III University of Madrid, highlighting the following subjects in particular:

- **Computer Architecture** (compulsory subject, Course 3): in which an introduction to the use of parallelism in computers is done.
- **Computer Networks** (compulsory subject, Course 3): in which the basic principles of networks and communication services are introduced.
- **Distributed Systems** (compulsory subject of the minor in Computer Engineering, Course 3): in which the design and programming of applications for distributed environments is studied.
- **Operating Systems** and **Operating systems design** (compulsory subjects, Courses 2 and 3, respectively): in which the fundamentals and the operating system design are studied.

This bachelor thesis has made an important contribution to the volunteer computing field. The main contributions of this work have been published in two articles:

- **"Analyzing the Performance of Volunteer Computing for Data Intensive Applications"**, Saúl Alonso Monsalve, Félix García Carballeira, Alejandro Calderón, *The 2016 International Conference on High Performance Computing & Simulation (HPCS 2016).* The 14th Annual Meeting, Innsbruck, Austria, July, 2016, conference.

- **"Estudio del rendimiento de plataformas de computación voluntaria para aplicaciones intensivas en datos"**, Saúl Alonso Monsalve, Félix García Carballeira, Alejandro Calderón, *XXVII Jornadas de Paralelismo (JP2016).* Salamanca, Spain, September, 2016, To Appear, conference.

Besides, at the time of delivery of this document we also have two other articles sent awaiting acceptance.

8.2 Conclusions

In this work we have described the design of ComBoS, a Complete simulator of the BOINC infrastructure. The aim of this project was to develop a simulator that, unlike other BOINC simulators, could simulate realistic scenarios taking into account the whole BOINC infrastructure: projects, servers, network, redundant computing, and volunteer nodes. Table 8.1 compares again the features of the existing BOINC simulators (SimBOINC, SimBA, and EmBOINC, presented in Chapter 2, *State of the Art*), but this time he have also included ComBoS, our simulator. Table 8.1 shows that ComBoS is the only simulator that gathers all BOINC components. In addition, it is the simulator that can perform the largest simulations.

ComBoS has been implemented in C programming language using SimGrid, a simulation-based framework for evaluating cluster, grid and P2P algorithms and heuristics. This document describes everything related to the development of ComBoS, taking into account the state of the art, the analysis, the design and the implementation of software. We have validated our simulator with the results obtained in three famous BOINC projects (Einstein@home, SETI@home and LHC@home), in terms of credits and GigaFLOPS obtained. We have shown the performance of the simulator in terms of memory usage and execution time, and we have demonstrated that ComBoS is highly scalable. Finally, we have used the simulator with different data intensive workloads and platforms in order to analyze possible bottlenecks and limits that an architecture like BOINC presents.

Features	ComBoS	SimBOINC	SimBA	EmBOINC
Network	input	-	traces	traces
Number of client hosts	500,000	1	40,000	100,000
Hosts availability	input (statist. distrib.) or traces	simulation	traces	traces
Hosts power	input (statist. distrib.) or traces	input	traces	traces
Hosts scheduler	simulation	simulation	-	-
Hosts reliability	input + simulation	-	traces	traces
Hosts execution	simulation	simulation	traces	traces
Hosts organization	individual hosts + clusters	-	individual hosts	individual hosts
Disk access	input	-	-	-
Number of tasks	100,000,000	-	200,000	350,000
Workunit validation	input + simulation	-	simulation	emulation
Workunit details	input	traces	input	traces
Project details	input	traces	-	-
Number of task servers	N	N	1	1
Number of data servers	N	-	-	-
Server scheduler	simulation	-	simulation	emulation

Table 8.1: *Comparison of main BOINC simulators (Including ComBoS).*

The main objective of this project was to develop a simulator that could guide the design of BOINC projects, and we have shown throughout this document that we have achieved our goal. We have also met all the other goals presented in the introduction of the document:

- To perform a simulation, users must specify all necessary parameters in an XML file (see Appendix A, *User Manual*).
- The simulator has linear-time complexity in the number of simulated hosts (see Section 6.2, *Performance Study*).
- The simulator allows to perform simulations of platforms with 1 million hosts (see Sections 6.2, *Performance Study*, and 6.3, *Case Studies*).
- We have developed the simulator by dividing its structure in different modules that allow us to easily add new functionalities in the future (see Chapter 4, *Design*).
- We have implemented the BOINC client scheduler (see Chapters 4, *Design*, and 5, *Implementation and Deployment*).

- We have created a generator that allows the user to compile and generate the files (executables, platform, and deployment) required for each simulation taking into account the parameters indicated in the XML file (see Chapter 5, *Implementation and Deployment*, and Appendix A, *User Manual*).

At a personal level, this work has helped me to get started in the world of scientific research. I have managed to use a lot of knowledge that I learned throughout the degree. Besides, I have learned all the key concepts of volunteer computing and how to use the most important simulation tools. I am very happy with the overall result, as I believe I have managed to overcome all the problems that have arisen.

8.3 Future Work

The aim of volunteer computing is that organizations be able to attain large computing power thanks to the participation of volunteer clients instead of a high investment in infrastructure. There are projects, like the ATLAS@home project, in which the number of running jobs has reached a plateau, due to a high load on data servers caused by file transfer. For future work, we want to use the simulator in order to design new models and architectures for volunteer computing platforms. In particular, we are interested in designing alternatives to the current system to solve the aforementioned problem. We also want to perform more case studies and use the simulator in order to analyze the energy consumption of the machines involved in a volunteer computing project.

Appendix A

User Manual

This appendix presents a detailed user manual of ComBoS. First we indicate the basic requirements to deploy the application and a detailed tutorial for the installation of SimGrid. Finally, we present an example of the simulator usage.

A.1 Basic Requirements

The technical specifications recommended for the final user to obtain the best experience from the application are:

- **Operating System**: Ubuntu 14.04.4 LTS (Linux distribution) or higher.
- **Processor**: Intel(R) Core(TM) i7 CPU 920 @2.67GHz or higher.
- **RAM**: 8 GB or higher.
- **Storage**: 1 GB of free space in the Hard Disk Drive.
- **Network**: Internet connection is not required.
- **Software**: The following software must be installed in order to run the application:

    1. GCC (GNU Compiler) 5.1 or higher.
    2. SimGrid toolkit 3.10 or higher.

A.2 SimGrid Installation

We will present a tutorial for the installation of the SimGrid toolkit (version 3.10). First, you have to download the official binary package from the *download page* (`http://simgrid.gforge.inria.fr/download.php`). In this case you will download the file *SimGrid-3.10.tar.gz*.

Then, you have to recompile de archive. This should be done in a few lines:

```
$ tar xf SimGrid-3.10.tar.gz
$ cd SimGrid-3.10
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/simgrid $HOME
$ make
$ make install
```

After following these steps, you will have the SimGrid toolkit installed in your computer.

A.3 Usage Example

In order to use ComBoS, you must download the corresponding files from the following website: `https://www.arcos.inf.uc3m.es/~combos/`. After unzipping the downloaded file, the unzipped files will follow the folder structure presented in Figure 5-2 (Section 5.2, *Deployment* (Chapter 5, *Implementation and Deployment*)). To perform simulations using ComBoS, it is necessary to model the platform to be simulated. Once you know the environment to simulate, you must specify all simulation parameters in the parameters XML file.

Figure A-1 shows an example of a potential simulation that can be carried out by ComBoS. The figure shows a simplified platform with two BOINC projects and 350,000 clients. The first project is represented by two scheduling servers (SS0 and SS1) and two data servers (DS0 and DS1). The second project consists of a single scheduling server (SS2) and three data servers (DS2, DS3 and DS4). Clients are grouped into three sets. The first group (G0) consists of 100,000 hosts and has a route to the first project. The second group (G1), has 200,000 hosts and a route to both projects. The third group (G2) consists of 50,000 computers and has route to the second project. The rest of the figure shows the links among the elements of the environment (from L0 to L7). In each of the links, latency and bandwidth are indicated.

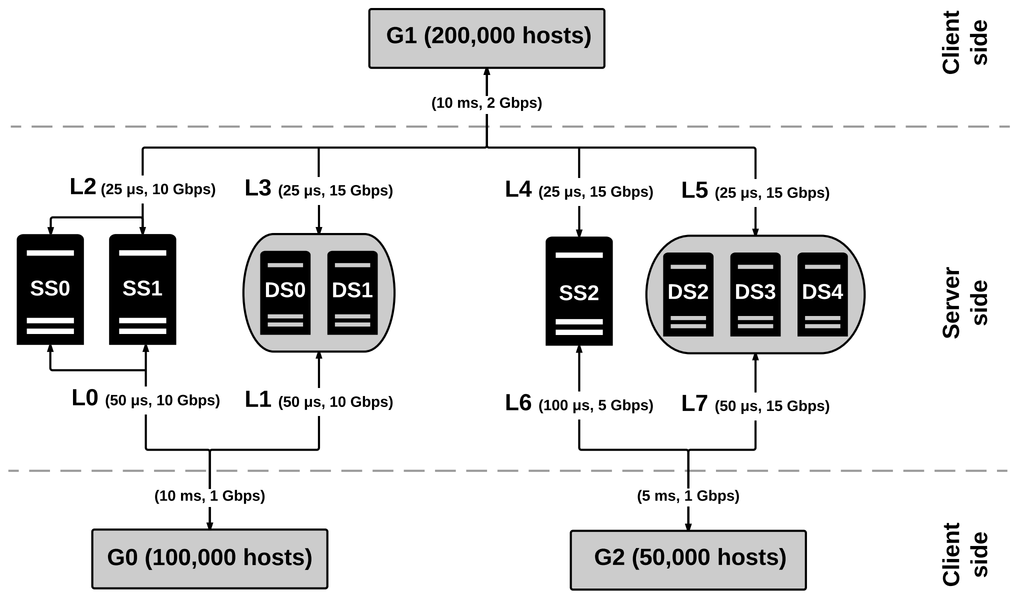
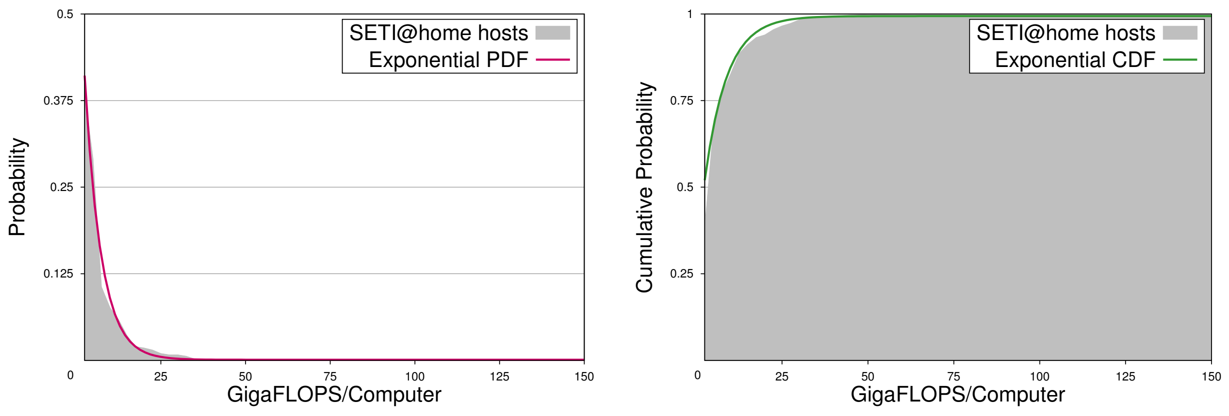


Figure A-1: *Simulator platform example.*

To create a simulation, ComBoS requires to specify all the parameters described in Tables 4.1 and 4.2 in the parameters.xml file (see Listing A.1). Users can define the power and availability of the volunteer hosts via either a traces file or distribution functions. For example, in the case of the SETI@home project, we have analyzed the 3,900,000 hosts that participate in this project. The CPU performance of the hosts can be modeled according to an exponential function, as shown Figure A-2, which has a mean of 5.871 GigaFLOPS per host.



(a) *Probability density function of SETI@home hosts power.*

(b) *Cumulative distribution function of SETI@home hosts power.*

Figure A-2: *CPU performance modeling for SETI@home hosts.*

Our software first processes the XML file, creating the necessary deployment and platform files for subsequent simulations. In ComBoS, all this is transparent to the user. The user only has to specify all the parameters of the simulation in the XML file (Listing A.1), and run the generator script using the following command:

```
$ ./generator
```

The above command generates the platform and deployment files. The platform file contemplates all the necessary elements in the simulation: hosts, clusters, links, etc. The deployment file indicates the processes that should be created during the simulation. In addition, the generator script also compiles all source files needed for the simulation and generates the executable file. Finally, to run the simulation you just need to run the execution script:

```
$ ./execute
```

The execution results are composed by multiple statistical results (see Listings A.2): the execution time, the memory usage of the simulator, the load of the scheduling and data servers, the total number of work requests received in the scheduling servers, the job statistics (number of jobs created, sent, received, analyzed, success, fail, too late, etc), the credit granted to the clients, the number of FLOPS, the average power of the volunteer nodes, and the percentage of time the volunteer nodes were available during the simulation.

```
<simulation_time>100</simulation_time>

<!-- Server side -->
<server_side>
    <n_projects>2</n_projects>
    <sproject>
        <snumber>0</snumber>
        <name>PROJECT1</name>
        <nscheduling_servers>2</nscheduling_servers>
        <ndata_servers>2</ndata_servers>
        <server_pw>12000000000.0</server_pw>
        <disk_bw>80000000</disk_bw>
        <ifgl_percentage>100<ifgl_percentage>
        <ifcd_percentage>100<ifcd_percentage>
        <input_file_size>368640</input_file_size>
        <task_fpops>7560000000000</task_fpops>
        <output_file_size>65536</output_file_size>
        <min_quorum>2</min_quorum>
        <target_nresults>2</target_nresults>
        <max_error_results>2</max_error_results>
        <max_total_results>4</max_total_results>
        <max_success_results>3</max_success_results>
        <delay_bound>100000000</delay_bound>
        <success_percentage>95</success_percentage>
        <canonical_percentage>95</canonical_percentage>
        <replication>2</replication>
    <sproject/>
    <sproject>
        <snumber>1</snumber>
        <name>PROJECT2</name>
        <nscheduling_servers>1</nscheduling_servers>
        <ndata_servers>3</ndata_servers>
        <server_pw>12000000000.0</server_pw>
        <disk_bw>60000000</disk_bw>
        <ifgl_percentage>100<ifgl_percentage>
        <ifcd_percentage>100<ifcd_percentage>
        <input_file_size>52428800</input_file_size>
        <task_fpops>20800000000000</task_fpops>
        <output_file_size>16777216</output_file_size>
        <min_quorum>2</min_quorum>
        <target_nresults>2</target_nresults>
        <max_error_results>2</max_error_results>
        <max_total_results>4</max_total_results>
        <max_success_results>3</max_success_results>
        <delay_bound>100000000</delay_bound>
        <success_percentage>95</success_percentage>
        <canonical_percentage>95</canonical_percentage>
        <replication>2</replication>
    <sproject/>
</server_side>

<!-- Client side -->
<client_side>
    <n_groups>3</n_groups>
    <group>
        <n_clients>100000</n_clients>
        <connection_interval>1</connection_interval>
        <scheduling_interval>3600</connection_interval>
        <gbw>1Gbps</gbw>
        <glatency>10ms</glatency>
        <traces_file>NULL</traces_file>
        <max_speed>117.71</max_speed>
        <min_speed>0.07</min_speed>
        <pv_distri>5</pv_distri>
        <pa_param>0.1734</pa_param>
        <pb_param>-1</pb_param>
        <av_distri>0</av_distri>
        <aa_param>0.393</aa_param>
        <ab_param>2.964</ab_param>
        <nv_distri>2</nv_distri>
        <na_param>2.844</na_param>
        <nb_param>-0.586</nb_param>
        <att_projs>1</att_projs>
        <gproject>
            <pnumber>0</pnumber>
            <priority>1</priority>
            <lsbw>10Gbps</lsbw>
            <lslatency>50us</lslatency>
            <ldbw>10Gbps</ldbw>
            <ldlatency>50us</latency>
        </gproject>
    </group>
    <group>
        <n_clients>200000</n_clients>
        <connection_interval>1</connection_interval>
        <scheduling_interval>3600</connection_interval>
        <gbw>2Gbps</gbw>
        <glatency>10ms</glatency>
        <traces_file>NULL</traces_file>
        <max_speed>117.71</max_speed>
        <min_speed>0.07</min_speed>
        <pv_distri>5</pv_distri>
        <pa_param>0.1734</pa_param>
        <pb_param>−1</pb_param>
        <av_distri>0</av_distri>
        <aa_param>0.393</aa_param>
        <ab_param>2.964</ab_param>
        <nv_distri>2</nv_distri>
        <na_param>2.844</na_param>
        <nb_param>−0.586</nb_param>
        <att_projs>2</att_projs>
        <gproject>
            <pnumber>0</pnumber>
            <priority>1</priority>
            <lsbw>10Gbps</lsbw>
            <lslatency>25us</lslatency>
            <ldbw>15Gbps</ldbw>
            <ldlatency>25us</latency>
        </gproject>
        <gproject>
            <pnumber>1</pnumber>
            <priority>1</priority>
            <lsbw>15Gbps</lsbw>
            <lslatency>25us</lslatency>
            <ldbw>15Gbps</ldbw>
            <ldlatency>25us</latency>
        </gproject>
    </group>
    <group>
        <n_clients>50000</n_clients>
        <connection_interval>1</connection_interval>
        <scheduling_interval>3600</connection_interval>
        <gbw>1Gbps</gbw>
        <glatency>5ms</glatency>
        <traces_file>NULL</traces_file>
        <max_speed>117.71</max_speed>
        <min_speed>0.07</min_speed>
        <pv_distri>5</pv_distri>
        <pa_param>0.1734</pa_param>
        <pb_param>−1</pb_param>
        <av_distri>0</av_distri>
        <aa_param>0.393</aa_param>
        <ab_param>2.964</ab_param>
        <nv_distri>2</nv_distri>
        <na_param>2.844</na_param>
        <nb_param>−0.586</nb_param>
        <att_projs>1</att_projs>
        <gproject>
            <pnumber>1</pnumber>
            <priority>1</priority>
            <lsbw>5Gbps</lsbw>
            <lslatency>100us</lslatency>
            <ldbw>15Gbps</ldbw>
            <ldlatency>50us</latency>
        </gproject>
    </group>
</client_side>
```

Listing A.1: *parameters.xml file filled with the parameters of the example.*

```
Memory usage: 11,570,812 KB

Total number of clients: 350,000

################### PROJECT1 ###################

Simulation ends in 100 h (360,000 sec)

Scheduling server 0: Busy: 13.4%
Scheduling server 1: Busy: 13.4%
Data server 0:  Busy: 12.2%
Data server 1:  Busy: 12.2%

 Number of clients: 300,000
 Messages received:   32,227,795 (work requests received + results received)
 Work requests received:  16,179,688
 Results created:   16,179,689 (100.0%)
 Results sent:   16,179,688 (100.0%)
 Results received:   16,048,107 (99.2%)
 Results analyzed:   16,048,107 (100.0%)
 Results success:   15,245,637 (95.0%)
 Results failed:   802,470 (5.0%)
 Results too late:   0 (0.0%)
 Results valid:   13,616,636 (84.8%)
 Workunits total:   7,716,639
 Workunits completed:   6,863,142 (88.9%)
 Workunits not completed:  853,497 (11.1%)
 Workunits valid:   6,808,318 (88.2%)
 Workunits error:   54,824 (0.7%)
 Throughput:    89.5 mens/s
 Credit granted:   231,482,812 credits
 FLOPS average:   285,949 GFLOPS

################### PROJECT2 ###################

Simulation ends in 100 h (360,000 sec)

Scheduling server 0: Busy: 1.7%
Data server 0:  Busy: 100.0%
Data server 1:  Busy: 100.0%
Data server 2:  Busy: 100.0%

 Number of clients: 250,000
 Messages received:   2,070,120 (work requests received + results received)
 Work requests received:  1,161,080
 Results created:   1,161,081 (100.0%)
 Results sent:   1,161,080 (100.0%)
 Results received:   909,040 (78.3%)
 Results analyzed:   909,040 (100.0%)
 Results success:   863,710 (95.0%)
 Results failed:   45,330 (5.0%)
 Results too late:   0 (0.0%)
 Results valid:   743,688 (81.8%)
 Workunits total:   559,384
 Workunits completed:   374,713 (67.0%)
 Workunits not completed:  184,671 (33.0%)
 Workunits valid:   371,844 (66.5%)
 Workunits error:   2,869 (0.5%)
 Throughput:    5.8 mens/s
 Credit granted:   35,697,024 credits
 FLOPS average:   42,968 GFLOPS

Group 0. Average speed: 6.704465 GFLOPS. Available: 62.1% Not available 37.9%
Group 1. Average speed: 5.455870 GFLOPS. Available: 55.3% Not available 44.7%
Group 2. Average speed: 6.110686 GFLOPS. Available: 57.0% Not available 43.0%

Clients. Average speed: 5.906157 GFLOPS. Available: 57.5% Not available 42.5%

Execution time:
0 days 7 hours 51 min 49 s
```

Listing A.2: *Simulation execution results.*

Glossary

dispatcher is a computing process that is responsible for the act of sending off, as to a specific destination. 51

framework is a real or conceptual structure intended to serve as a support or guide for the building of something that expands the structure into something useful. ix, x, xv, 3, 16, 19, 20, 27, 28, 39, 86

idle is the total time a computer or device has been powered on, but has not been used. 1

message passing is a form of communication used in parallel programming and object-oriented programming. Communications are completed by the sending of messages (functions, signals and data packets) to recipients. 5, 6

middleware is a general term for software that serves to "glue together" separate, often complex and already existing, programs. Some software components that are frequently connected with middleware include enterprise applications and Web services. vii, 2, 9, 25

networking refers to a telecommunications network which allows computers to exchange data. 7, 15, 16

open-source computer source code, that is made freely available to the general public by its creators. 2, 8, 9, 28, 39, 44

preemption is the act of temporarily interrupting a task being carried out by a computer system, without requiring its cooperation, and with the intention of resuming the task at a later time. 46, 49, 50

protocol is the special set of rules that end points in a telecommunication connection use when they communicate. Protocols specify interactions between the communicating entities. 18, 39

scheduling is the method by which work specified by some means is assigned to resources that complete the work. vii, x, 19–22, 26–28, 31, 35, 41, 42, 44–49, 51, 52, 66, 73, 74, 90, 92

throughput is a measure of how many units of information a system can process in a given amount of time. vii, 2, 18, 24, 76

Acronyms

ARCOS Grupo de Aquitectura de Computadores, Universidad Carlos III de Madrid. v, 82

BOINC Berkeley Open Infrastructure for Network Computing. vii, 2, 3, 9–11, 23–27, 39, 44, 46–49, 57, 69, 70, 72, 73, 75, 84, 86, 87, 90

ComBoS Complete BOINC Simulator. vii, 2, 23, 24, 26, 41, 44, 46–48, 54, 55, 57, 59–63, 66, 67, 69–71, 73, 75, 84, 86, 89–92

EDF Earliest Deadline First. 49

FLOPS Floating-point Operations per Second. xvi, 2, 6, 9, 45, 47, 48, 60, 62, 66, 69–71, 86, 91, 92

FTP File Transfer Protocol. 7

HPC High-Performance Computing. 6, 19

HTML Hyper-Text Markup Language. 7

HTTP Hypertext Transfer Protocol. 7, 10

IaaS Infrastructure as a Service. 13

IRC Internet Relay Chat. 7

IVA Impuesto Sobre el Valor Añadido. 83

MPI Message Passing Interface. 20, 22

NAT Network Address Translation. 11

P2P Peer-to-Peer. ix, xiii, 11, 15, 16, 19, 86

PaaS Platform as a Service. 13, 14

PC Personal Computer. 8, 10, 25, 82

RAM Random-Access Memory. 37, 53, 62, 82, 89

RPC Remote Procedure Call. 48

SaaS Software as a Service. 13, 14

WWW World Wide Web. 6, 7

XML Extensible Markup Language. 2, 21, 24, 38, 55, 87, 88, 90, 92

Bibliography

[1] INRIA/IN2P3, "XtremWeb: the open source platform for desktop grids." `http://www.xtremweb.net/`, Last visited, 2016.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[3] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, 2004.

[4] "BOINCstats." `http://boincstats.com/en/stats`, Last visited, Feb. 2016.

[5] "Top 500 Supercomputer list." `http://www.top500.org/`, Last visited, Apr. 2016.

[6] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2899–2917, June 2014.

[7] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems (4th ed.): concepts and design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[8] A. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms (2nd ed.)*. Prentice Hall, 2007.

[9] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[10] G. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 1999.

[11] S. Dolev, *Self-Stabilization*. MIT Press, 2000.

[12] S. Ghosh, *Distributed Systems: An Algorithmic Approach*. Chapman & Hall/CRC, 2006.

[13] D. Peleg, *Distributed computing: a locality-sensitive approach*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[14] "High-Performance Computing." `http://searchenterpriselinux.techtarget.com/definition/high-performance-computing`, Last visited, May 2016.

[15] TechTerms.com, "WWW." `http://www.techterms.com/definition/www`, Last visited, Apr. 2016.

[16] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 200–222, August 2001.

[17] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, pp. 37–46, June 2002.

[18] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. The Morgan Kaufmann Series in Computer Architecture and Design Series, Elsevier, 2004.

[19] C. Catlett, "In search of gigabit applications," *Communications Magazine, IEEE*, vol. 30, pp. 42 –51, april 1992.

[20] L. Smarr and C. E. Catlett, "Metacomputing," *Commun. ACM*, vol. 35, pp. 44–52, June 1992.

[21] J. I. Beiriger, H. P. Bivens, S. L. Humphreys, W. R. Johnson, and R. E. Rhea, "Constructing the asci computational grid," in *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, HPDC '00, (Washington, DC, USA), pp. 193–, IEEE Computer Society, 2000.

[22] S. Brunett, K. Czajkowski, S. Fitzgerald, I. Foster, A. Johnson, C. Kesselman, J. Leigh, and S. Tuecke, "Application experiences with the globus toolkit," 1998.

[23] W. E. Johnston, D. Gannon, and B. Nitzberg, "Grids as production computing environments: The engineering aspects of nasa's information power grid," in *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, HPDC '99, (Washington, DC, USA), pp. 34–, IEEE Computer Society, 1999.

[24] R. Stevens, P. Woodward, T. DeFanti, and C. Catlett, "From the i-way to the national technology grid," *Commun. ACM*, vol. 40, pp. 50–60, November 1997.

[25] D. Anderson, E. Korpela, and R. Walton, "High-performance task distribution for volunteer computing," in *e-Science and Grid Computing, 2005. First International Conference on*, pp. 8 pp.–203, July 2005.

[26] S. Choi, R. Buyya, H. Kim, E. Byun, M. Baik, J. Gil, and C. Park, "A taxonomy of desktop grids and its mapping to state-of-the-art systems," Tech. Rep. GRIDS-TR-2008-3, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Feb. 2008.

[27] Mersenne Research, Inc., "Great Internet Mersenne Prime Search." `http://www.mersenne.org/`, Last visited, Apr. 2016.

[28] distributed.net, "distributed.net." `http://distributed.net`, Last visited, Apr. 2016.

[29] S. M. Larson, C. D. Snow, M. R. Shirts, and V. S. Pande, "Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology," in *Computational Genomics: Theory and Application* (R. P. Grant, ed.), Horizon Bioscience, 2004.

[30] F. Costa, J. Silva, L. Veiga, and P. Ferreria, "Large-scale volunteer computing over the internet," *Journal of Internet Services and Applications*, vol. 3, pp. 329–346, 2012.

[31] "Volunteer computing." http://boinc.berkeley.edu/trac/wiki/VolunteerComputing, Last visited, Dec. 2015.

[32] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, (Washington, DC, USA), pp. 4–10, IEEE Computer Society, 2004.

[33] LIGO Scientific Collaboration and D. P. Anderson, "Einstein@Home search for periodic gravitational waves in early S5 LIGO data," *Physical Review D*, vol. 80, p. 042003, 2009.

[34] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela, "SETI@HOME—massively distributed computing for SETI," *Computing in Science and Engineering*, vol. 3, no. 1, pp. 78–83, 2001.

[35] P. Paul, "SETI@home project and its website," *Crossroads*, vol. 8, no. 3, pp. 3–5, 2002.

[36] A. P. Society, "Einstein@home." http://www.einsteinathome.org, Last visited, Apr. 2016.

[37] Oxford University, "Climateprediction.net." http://climateprediction.net, Last visited, Apr. 2016.

[38] "Creating BOINC Projects." https://boinc.berkeley.edu/boinc.pdf, Last visited, Jan. 2016.

[39] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 188–201, 2001.

[40] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," *SIGARCH Comput. Archit. News*, vol. 28, no. 5, pp. 190–201, 2000.

[41] F. Costa, L. Silva, I. Kelley, and I. Taylor, "Peer-To-Peer Techniques for Data Distribution in Desktop Grid Computing Platforms," *Proceedings of the CoreGRID Workshop on Programming Models Grid and P2P System Architecture Grid Systems, Tools and Environments 12-13 June 2007, Heraklion, Crete, Greece*, pp. 377–391, 2007.

[42] E. Griffith, "What Is Cloud Computing?." http://www.pcmag.com/article2/0,2817,2372163,00.asp, Last visited, June 2016.

[43] U.S. Department of Energy, *The Magellan Report On Cloud Computing for Science*. Office of Advance Scientific Computing Research (ASCR), 2011.

[44] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, (Berkeley, CA, USA), pp. 15–15, USENIX Association, 2006.

[45] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, pp. 4:1–4:26, June 2008.

[46] L. Grandinetti, O. Pisacane, and M. Sheikhalishahi, *Pervasive Cloud Computing Technologies: Future Outlooks and Interdisciplinary Perspectives*. 2014.

[47] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *Recommendations of the National Institute of Standards and Technology, Special Publication 800-145*, 2011.

[48] "Amazon Web Services." `http://aws.amazon.com/`, Last visited, June 2016.

[49] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pp. 101 –102, aug 2001.

[50] D. Barkai, "Peer-to-Peer Computing," *Intel Press*, 2002.

[51] P. Velho and A. Legrand, "Accuracy study and improvement of network simulation in the SimGrid framework," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, (ICST, Brussels, Belgium, Belgium), pp. 13:1–13:10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[52] B. Liu, D. Figueiredo, Y. Guo, J. Kurose, and D. Towsley, "A study of networks simulation efficiency: fluid simulation vs. packet-level simulation," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1244–1253 vol.3, 2001.

[53] C. McDonald, "A network specification language and execution environment for undergraduate teaching," *ACM SIGCSE Bulletin*, vol. 23, pp. 25–34, Mar. 1991.

[54] C. McDonald, "A network specification language and execution environment for undergraduate teaching," in *Proceedings of the twenty-second SIGCSE technical symposium on Computer science education*, SIGCSE '91, (New York, NY, USA), pp. 25–34, ACM, 1991.

[55] J. H. Cowie, D. M. Nicol, and A. T. Ogielski, "Modeling the global internet," *Computing in Science & Engineering*, vol. 1, pp. 42–50, Jan. 1999.

[56] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zapalla, "Improving simulation for network research," Tech. Rep. 99-702, University of Southern California, Computer Science Department, Mar. 1999.

[57] The University of Southern California, "The network simulator - ns-2." Web page, Nov. 2011.

[58] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 project goals," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, (New York, NY, USA), ACM, 2006.

[59] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. B. Kopena, "Network simulations with the ns-3 simulator," in *Demonstrations of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM'08, (Seattle, Washington, USA), ACM, Aug. 2008.

[60] G. F. Riley, "The Georgia Tech Network Simulator," in *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, MoMeTools '03, (New York, NY, USA), pp. 5–12, ACM, 2003.

[61] A. Varga, "The INET framework." `http://inet.omnetpp.org/`, Last visited, Apr. 2016.

[62] K. Fujiwara and H. Casanova, "Speed and accuracy of network simulation in the SimGrid framework," in *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, ValueTools '07, (ICST, Brussels, Belgium, Belgium), pp. 12:1–12:10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Oct. 2007.

[63] A. Sulistio, G. Poduval, R. Buyya, and C.-K. Tham, "On incorporating differentiated levels of network service into gridsim," *Future Gener. Comput. Syst.*, vol. 23, pp. 606–615, May 2007.

[64] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini, "OptorSim: A simulation tool for scheduling and replica optimisation in data grids," in *Proceedings of the Computing in High Energy and Nuclear Physics (CHEP) conference*, 2004.

[65] C. L. Dumitrescu and I. Foster, "GangSim: A simulator for grid scheduling studies," in *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2 - Volume 02*, CCGRID '05, (Washington, DC, USA), pp. 1151–1158, IEEE Computer Society, 2005.

[66] P. Urbán, X. Défago, and A. Schiper, "Neko: A Single Environment to Simulate and Prototype Distributed Algorithms," *Journal of Information Science and Engineering*, vol. 18, no. 6, pp. 981–997, 2002.

[67] S. Phatanapherom, P. Uthayopas, and V. Kachitvichyanukul, "Fast simulation model for grid scheduling using hypersim," in *Proceedings of the 2003 Winter Simulation Conference*, vol. 2, pp. 1494–1500 vol.2, 2003.

[68] B. Quetier and F. Cappello, "A survey of grid research tools: simulators, emulators and real life platforms," in *Proceedings of the 17th IMACS World Congress (IMACS)*, July 2005.

[69] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," *ACM SIGCOMM Computer Communication Review*, vol. 28, pp. 303–314, Oct. 1998.

[70] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, pp. 67–82, July 1997.

[71] T. Ott, J. Kemperman, and M. Mathis, "Window size behavior in TCP/IP with constant loss probability," in *Proceedings of 4th IEEE Workshop on High-Performance Communication Systems (HPCS)*, June 1997.

[72] L. Massoulié and J. Roberts, "Bandwidth sharing: objectives and algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, pp. 320–328, June 2002.

[73] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a generic framework for large-scale distributed experiments," in *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, UKSIM '08, (Washington, DC, USA), pp. 126–131, IEEE Computer Society, Mar. 2008.

[74] The SimGrid Team, "Simgrid." `http://simgrid.gforge.inria.fr/`, Last visited, Jan. 2016.

[75] "GridSim Project Team." `http://www.buyya.com/gridsim/`, Last visited, May 2016.

[76] A. Legrand and J. Lerouge, "MetaSimGrid: Towards realistic scheduling simulation of distributed applications," Tech. Rep. 2002-28, Laboratoire de l'Informatique du Parallélisme (LIP), École Normale Supérieure de Lyon, July 2002.

[77] M. Snir, S. Otto, S. Huss-Lederman, and J. Dongarra, "Mpi the complete reference," *The MIT Press*, 1996.

[78] H. Xia, H. Dail, H. Casanova, and A. Chien, "The micrograd: Using online simulation to predict application performance in diverse grid network environments," *In Second International Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*, 2004.

[79] iSGTW Opinion - Volunteer computing: grid or not grid?, "BOINCstats." `https://sciencenode.org/feature/isgtw-opinion-volunteer-computing-grid-or-not-grid.php`, Last visited, Nov. 2015.

[80] D. Kondo, *Scheduling Task Parallel Applications for Rapid Turnaround on Enterprise Desktop Grids*. PhD thesis, University of California at San Diego La Jolla, 2005.

[81] B. Donassolo, H. Casanova, A. Legrand, and P. Velho, "Fast and scalable simulation of volunteer computing systems using SimGrid," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, (New York, NY, USA), pp. 605–612, ACM, 2010.

[82] Trilce Estrada, David A. Flores, Michela Taufer, Patricia J. Teller, Andre Kerstens, David P. Anderson, "The Effectiveness of Threshold-Based Scheduling Policies in BOINC Projects," in *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference*, p. 88, 2006.

[83] Trilce Estrada, Michela Taufer, Kevin Reed, David P. Anderson, "EmBOINC: An emulator for performance analysis of BOINC projects," in *Parallel and Distributed Processing Symposium, International*, pp. 1–8, 2009.

[84] J. Wu, X. Zhao, X. Sui, and X. Yang, "PVMsim: A parallel simulation platform to evaluate virtual machines," *Future Computer and Communication, ICFCC'2010.*, 2010.

[85] A. García-Guirado, R. Fernández-Pascual, and J. M. García, "Virtual-GEMS: An Infrastructure To Simulate Virtual Machines," *Modelling, benchmarking and simulation*, 2009.

[86] S. Lim, B. Sharma, G. Nam, E. Kim, and C. Das, "MDCSim: A multi-tier data center simulation platform," *Cluster Computing and Workshops, CLUSTER'2009*, 2009.

[87] I. Sriram and D. Clif, "SPECI-2 - An Open-source Framework for Predictive Simulation of Cloud-scale Data-centres," *Simulation and Modeling Methodologies, Technologies and Applications, SIMULTECH'2011*, 2011.

[88] Institute for Electrical and Electronics Engineers, "IEEE Recommended Practice for Software Requirements Specifications," *IEEE*, pp. 830–1998, 1998.

[89] D. P. Anderson, "Local scheduling for volunteer computing," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–8, IEEE, 2007.

[90] BOE, 19 de enero de 2008, "Real Decreto 1720/2007, de 21 de diciembre, por el que se aprueba el Reglamento de desarrollo de la Ley Orgánica 15/1999, de 12 de diciembre, de protección de datos de carácter personal.," *Boletín Oficial del Estado, 17:4103-4136*.

[91] "The GNU Lesser General Public License." `http://www.gnu.org/licenses/licenses.html`, Last visited, June 2016.

[92] "BOINC Jobs." `https://boinc.berkeley.edu/trac/wiki/JobIn`, Last visited, Jan. 2016.

[93] "Computation credit." `http://boinc.berkeley.edu/wiki/Computation_credit`, Last visited, June 2016.

[94] UNIX, "drand48." `http://pubs.opengroup.org/onlinepubs/7908799/xsh/drand48.html`, Last visited, June 2016.

[95] Software Testing Fundamentals, "Verification vs Validation." `http://softwaretestingfundamentals.com/verification-vs-validation/`, Last visited, Mar. 2016.

[96] SETI@home, "CPU performance of SETI@home volunteer computers." `http://setiathome.berkeley.edu/cpu_list.php`, Last visited, Mar. 2016.

[97] EINSTEIN@home, "CPU performance of EINSTEIN@home volunteer computers." `https://www.einsteinathome.org/cpu_list.php`, Last visited, Mar. 2016.

[98] LHC@home classic, "CPU performance of LHC@home volunteer computers." `http://lhcathomeclassic.cern.ch/sixtrack/cpu_list.php`, Last visited, Mar. 2016.

[99] Bahman Javadi, Derrick Kondo, Jean-Marc Vincent, David P. Anderson, "Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home," *Parallel and Distributed Systems, IEEE Transactions*, vol. 22, pp. 1896–1903, 2011.

[100] Arnaud Legrand, "Scheduling for Large Scale Distributed Computing Systems: Approaches and Performance Evaluation Issues," tech. rep., Université Grenoble Alpes, 2015.

[101] Accelerated Technologies, Inc, "The Human Condition: A Justification for Rapid Prototyping," *Time Compression Technologies, vol. 3 no. 3*, p. 1, May 1998.

[102] Benington, Herbert D., "Production of Large Computer Programs," *IEEE Annals of the History of Computing (IEEE Educational Activities Department)*, p. 350–361, Oct. 1983.

[103] B. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer 21, 5*, pp. 61–72, 1988.

[104] BOE, 30 de junio de 2015, "Resolución de 17 de junio de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan becas de colaboración de estudiantes en departamentos universitarios para el curso académico 2015-2016," *Boletín Oficial del Estado, 155:53607-53616*.

[105] BOE, 6 de agosto de 2012, "Resolución de 2 de agosto de 2012, de la Dirección General de Tributos, sobre el tipo impositivo aplicable a determinadas entregas de bienes y prestaciones de servicios en el Impuesto sobre el Valor Añadido," *Boletín Oficial del Estado, 187:56055-56060*.

[106] "Citizen Science Grid." http://csgrid.org/csg/, Last visited, Feb. 2016.