Alonso-Monsalve, S., García Carballeira, F., Calderón, A. (2017). ComBos: a complete simulator of volunteer computing and desktop grids. *Simulation Modelling Practice and Theory*, 77, pp. 197-211.

DOI: 10.1016/j.simpat.2017.06.002

ComBoS: A Complete Simulator of Volunteer Computing and Desktop Grids

Saúl Alonso-Monsalve, Félix García-Carballeira, Alejandro Calderón

Computer Architecture Group

Computer Science and Engineering Department

Universidad Carlos III de Madrid, Leganés, Spain

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. In a similar way, Desktop Grid Computing is a form of distributed computing in which an organization uses its existing computers to handle its own long-running computational tasks. BOINC is the main middleware that provides a software platform for volunteer computing and desktop grid 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 paper we present a complete simulator of BOINC infrastructures, called ComBoS. Although there are other BOINC simulators, none of them allow us to simulate the complete infrastructure of BOINC. Our goal 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 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. The paper 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. The paper 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: Volunteer Computing, BOINC, Simulation, Throughput, Desktop Grids

1. Introduction

Volunteer Computing (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 [1]:

- Because of the huge number of computers 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 computers is in continuous growth.
- Volunteer Computing 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.

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

In a similar way, Desktop Grid Computing (DG) is a form of distributed computing in which an organization uses its existing computers to handle its own long-running computational task. This differs from VC in several ways: in DG, the computing resources can be trusted; there is no need for screensaver graphics, in fact it may be desirable to have the computation be completely invisible and out of the control of the PC user; and client deployment is typically automated. Both, VC and DG are disciplines of distributed computing that consider the whole Internet as a possible computing platform.

BOINC (Berkeley Open Infrastructure for Network Computing) [2] is an open-source software platform for Volunteer Computing and Desktop Grid Computing. It provides the most widely used middleware system for volunteer computing and desktop grids. Although the growth of desktop computers is stalled and currently the world is on laptops and moving to mobile fast, according to BOINCstats [3], currently there are 57 BOINC 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 100 on the TOP500 [4] list which is constituted by the 500 fastest supercomputers of the world. BOINC has evolved with the new mobile platforms, and currently it provides a version for Android devices [5]. In this kind of platforms BOINC only computes when the device is plugged into a power source (AC or USB) and the battery is charged 90% or more, so it won't significantly reduce the battery life or the recharge time. Moreover, BOINC transfers data only when the devices are connected to a WiFi network.

In this paper we present ComBoS[1] (Complete BOINC Simulator), a complete and open-source simulator for volunteer computing and desktop grids, based on the whole BOINC infrastructure. ComBoS simulates real VC and DG scenarios. These scenarios are defined by a large set of parameters specified in an 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 hosts) in just a few hours. Although there are other BOINC simulators, any of them allow to simulate the complete infrastructure of BOINC. Our goal was to create a complete simulator that 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 rest of the paper is organized as follows: Section 2 discusses related work; Section 3 presents and describes the simulator that we have developed; in Section 4 we validate ComBoS by comparing its statistical results with the real ones of three different BOINC projects; Section 5 analyzes the performance of the simulator; Section 6 presents some case studies; and finally, Section 7 concludes the paper and presents some future work.

2. Background and Related Work

2.1. BOINC

The computing resources that power Volunteer Computing (VC) and Desktop Grids (DG) are shared with the owners of the client machines. Because the resources are volunteered, utmost care is taken to ensure that the VC and DG tasks do not obstruct the activities of each machine's owner;

[1] `http://www.arcos.inf.uc3m.es/~combos/`

a VC or DG task is suspended or terminated whenever the machine is in use by another person. As a result, VC or DG resources are volatile in the sense that any number of factors can prevent the task of a VC or DG application from being completed. These factors include mouse or keyboard activity, the execution of other user applications, machine reboots, or hardware failures. Moreover, VC an DG 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.

BOINC [7] 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. BOINC is being used by several projects, including 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 [8] 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 [9, 10, 11], where each input file is different).

The BOINC architecture [12] 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 [13] and Climateprediction.net [14]) 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:

- 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 1). 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. BOINC provides a form of redundant computing in which each computation is performed on multiple clients [15], the results are compared, and are accepted only when a 'consensus' is reached. In some cases new results must be created and sent.

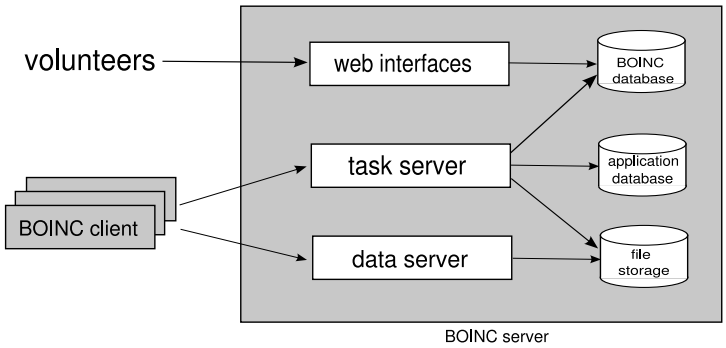


Figure 1: A BOINC server consists of multiple components, sharing several forms of storage.

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. 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.

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 storage systems such as Gnutella, PAST [16] and Oceanstore [17]. 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.

The BOINC architecture is based on a strict master/worker model (see Figure 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 NAT (Network Address Translation) 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 needed or results are ready for submission.

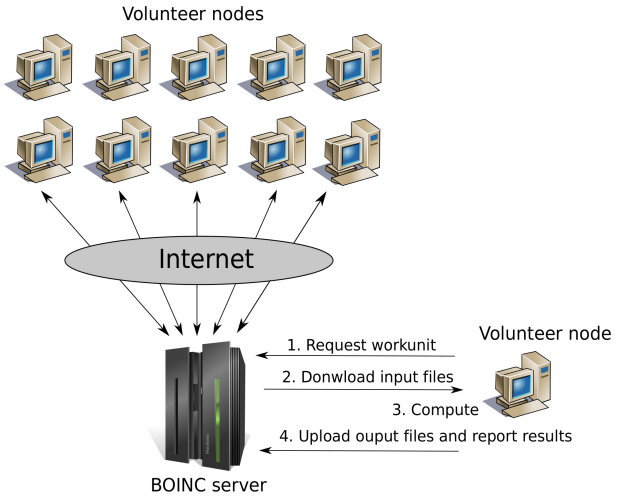


Figure 2: Strict master/worker model in BOINC.

2.2. BOINC Simulators

Desktop Grid Computing and Volunteer Computing are forms of distributing computing that seek to exploit existing resources [18]. However, there are several differences between them. 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, desktop grid computing resources belong to information technology professionals, users may or may not be anonymous, computing resources can be trusted, and performance is partially predictable.

There are not too many volunteer computing simulators, although most of them are focused on BOINC. As ComBoS, there are other BOINC simulators based on the SimGrid toolkit [6]. An example of this is SimBOINC, which simulates the BOINC client scheduler. SimBOINC [19] 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,

Table 1: Comparison of BOINC simulators.

Features	ComBoS	SimBOINC	SimBA	EmBOINC
Network	input	-	traces	traces
Number of client hosts	N	1	40,000	100,000
Hosts availability	input (statistical distribution) or traces	simulation	traces	traces
Hosts power	input (statistical distribution) 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	N	-	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

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 [20], but it is more efficient (about three or four times faster) and also has a source code of about 800 lines.

In contrast to the aforementioned simulators, SimBA [21] (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 [22] uses a population of volunteered clients and emulates the server component. EmBOINC does not have a client scheduler either. Table 1 compares the main features of the programs presented in this section (SimBOINC, SimBA and EmBOINC) and ComBoS.

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 section we describe our simulator in detail.

This simulator has been written in C programming language using SimGrid. SimGrid [6] is a simulation-based framework for evaluating cluster, grid and P2P algorithms and heuristics. The SimGrid simulation core [20] implements and provides interfaces to a number of simulation models that vary in sophistication, and can be used to simulate different types of resources, such as network resources and computational resources.

3. Simulator Description

ComBoS is a complete simulator of BOINC infrastructures that simulates the behavior of all componentes involved: projects, servers, network, redundant computing, and volunteer nodes. In this section we describe all the simulator components and present a significant use case. To create a simulation, ComBoS requires to specify all the parameters needed in a XML file, such as the simulation time in hours. All other parameters required by the XML file are detailed below.

3.1. Simulator Components

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. 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 [23]:

- A *workunit* describing the computation to be performed.
- 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 paper, we use both terms interchangeably.

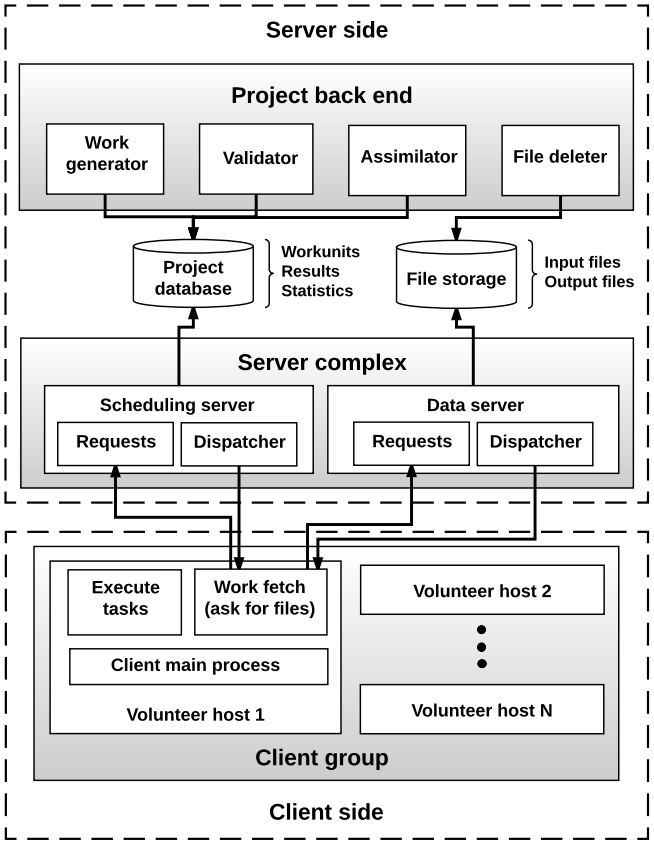


Figure 3: ComBoS architecture.

3.1.1. Server side

Servers are responsible for managing projects. The architecture of the server side is shown in Figure 3. The server side of a project consist of two parts [15]:

- 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; a *validator*, that examines sets of results and selects canonical results; an *assimilator*, that handles workunits that are completed; and a *file deleter*, that deletes input and output files that are no longer needed.

Table 2: Project parameters.

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.

- 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; and *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.

ComBoS allows for the definition of multiple projects. For each project, users must define the parameters described in Table 2.

3.1.2. Client side

In ComBoS, the client side is formed by groups of 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 [24]:

- 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.

Algorithm 1 Client main thread

```
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 thread
8:         WAIT scheduling_interval
9:     end while
10:    Return
11: end function
```

Algorithm 2 Work fetch thread

```
1: function WORK_FETCH( )
2:     project = null
3:     while time < max_time do
4:         for each project p in projects do
5:             if p meets the requirements then
6:                 project = p
7:             end if
8:         end for
9:         if project and not deadlines_missed then
10:            ASK_FOR_WORK(project)
11:        end if
12:        WAIT work_fetch_period
13:    end while
14:    SIGNAL Client main thread
15:    Return
16: end function
```

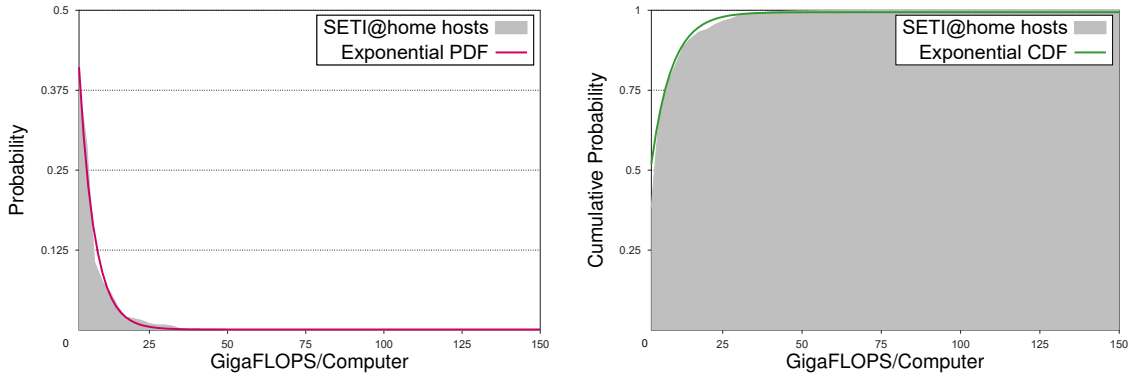
The scheduling is based on a round-robin between projects, weighted according to their resource share. This scheduling is described in detail in [24]. In addition, we have relied on the client scheduler code implemented in [20]. In ComBoS, each client is implemented with at least three different threads: the client main thread (Algorithm 1), which updates the client parameters every scheduling interval; the work fetch thread (Algorithm 2), which selects the project to ask for work; and the execution threads, one per attached project, that execute the tasks. Meanwhile, our simulator is complemented with the most important features of the real scheduler (deadline scheduling, long term debt, fair sharing and exponential back-off).

Table 3: VN group parameters.

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.

Apart from that, ComBoS allows for the definition of multiple VN groups. The power and the availability of each host of the group is obtained from a traces file. Alternatively, the power and the availability can be modelled with input statistical distributions. For each group, users must define the parameters described in Table 3. To simulate VN groups using SimGrid, we have used the *cluster* entity. Like real clusters, each cluster contains many hosts interconnected by a dedicated network. SimGrid does not allow us to fix the power and availability of individual hosts within a cluster, so we have implemented the necessary functionality in order to solve the problem. As indicated in Table 3 (VN group parameters), users can define the power and availability of the VN 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 4, which has a mean of 5.871 GigaFLOPS per host.



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

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

Figure 4: CPU performance modeling for SETI@home VN.

3.2. Use Case

Figure 5 shows an example of a potential simulation that can be carried out by ComBoS. The figure shows a simplified environment with two BOINC projects and 350,000 clients. The first project is represented by two scheduling servers (S0 and S1) and two data servers (DS0 and DS1). The second project consists of a single scheduling server (S2) and three data servers (DS2, DS3 and DS4). Clients are grouped in 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.

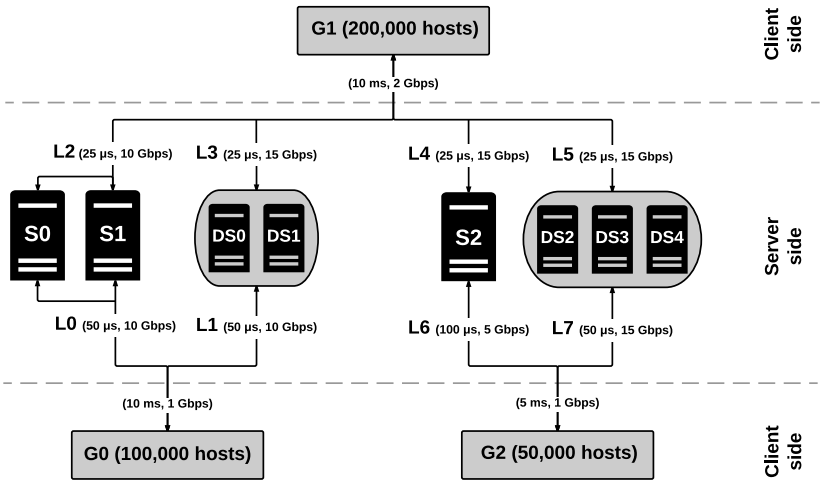


Figure 5: Simulation environment example.

To create a simulation, ComBoS requires to specify all the parameters described in Tables 2 and 3 in a XML file. Our software first processes the XML file, creating the necessary deployment

and platform files for subsequent simulations. 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 ComBoS, all this is transparent to the user. The user only has to specify all the parameters of the simulation in the XML file, run the generator script and finally run the execution script.

The execution results are composed by multiple statistical results: 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, and the VN average power and the percentage of time the VN were available during the simulation.

4. Validation of the Simulator

In this section we validate the results provided by ComBoS, comparing them to the results of SimBOINC and to the statistical results of the official BOINC webpage.

All measurements shown in this section and in sections 5 and 6 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. We have used the same host availability and unavailability parameters as those used in Section 4.2. Each simulation result presented in this section and sections 5 and 6 is based on the average of 20 runs. For a 95% confidence interval, the error is less than $\pm$ 2% for all values.

4.1. 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. We compare our simulator with SimBOINC, because is the only one that takes into account the client scheduling.

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 (Background and Related Work), 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 4, 5 and 6 show different test cases:

Table 4: Executed tasks (three projects running on a single host), $1.4 \cdot 10^9$ FLOPS host power.

	SimBOINC			*ComBoS*		
Time in hours	*Einstein@home*(25%)	*SETI@home*(25%)	*LHC@home*(50%)	*Einstein@home*(25%)	*SETI@home*(25%)	*LHC@home*(50%)
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 4 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.

Table 5: Executed tasks (three projects running on a single host), $5.5 \cdot 10^9$ FLOPS host power.

	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: Executed tasks (single project running on a single host), $5.5 \cdot 10^9$ FLOPS host power.

	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 5 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.
- Table 6 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.

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

4.2. Validation of Whole Simulator

To validate the complete simulator, we have relied on data from the BOINCstats website [3], 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 [25, 26, 27]. We have not used any other traces. In order to model the availability and unavailability of the hosts, we used the results obtained in [28]. 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 [28] too.

Table 7 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 7.9% for

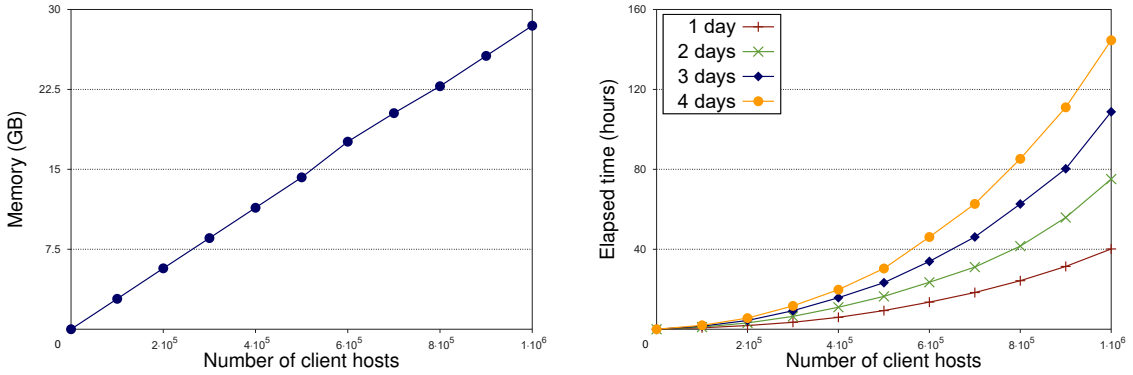
Table 7: Validation of the whole simulator.

			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

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.

5. Performance Study

In this section we analyze the performance of the simulator in terms of memory usage and execution time. We do not compare ComBoS with the current BOINC simulators described in Section 2 in terms of performance because our simulator is the only one that can manage simulations that take into account the whole BOINC infrastructure.



(a) Memory usage of the simulator.

(b) Execution time of the simulator.

Figure 6: Performance study.

Figure 6a shows the memory usage of the simulator and Figure 6b 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 6a shows a linear ($O(n)$) memory footprint. Figure 6b 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 [29] of the SimGrid toolkit. Notice that we have managed to run simulations with 1 million hosts, the same number of active machines of all BOINC projects combined.

6. Case Studies

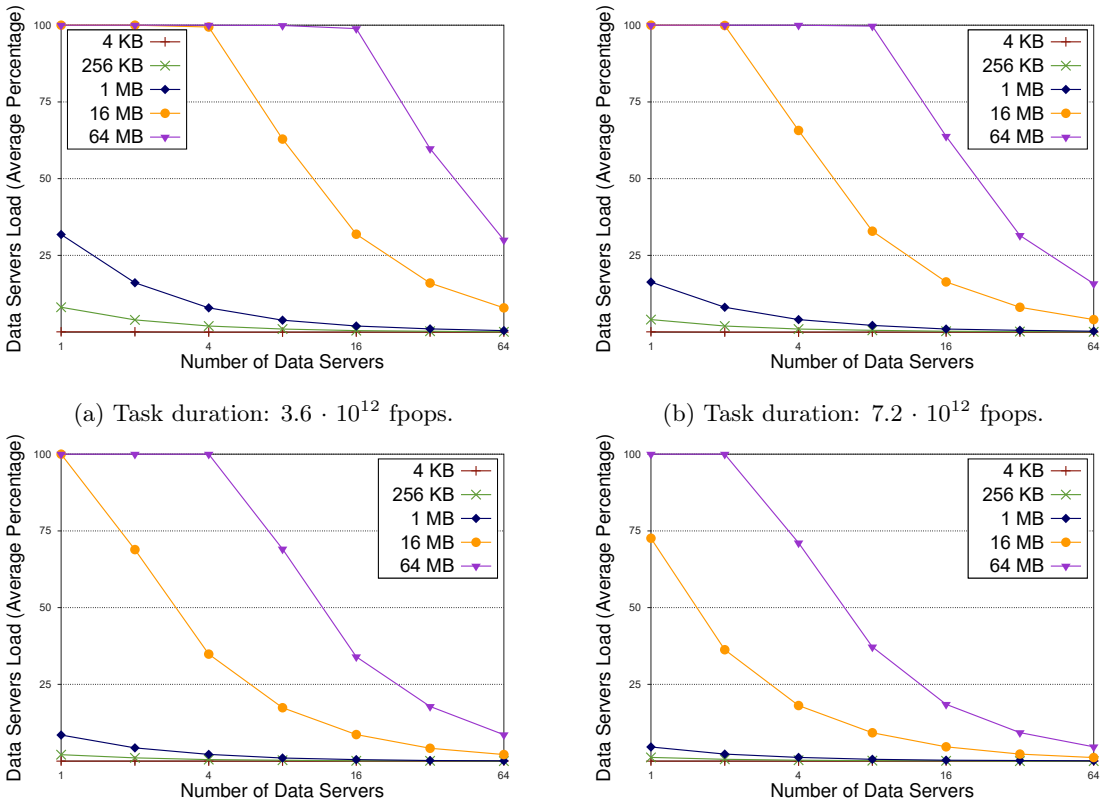
In this section we will present different case studies using ComBoS. Our goal is to show the performance that would result when Volunteer Computing and Desktop Grid platforms are used to process big amounts of data. We are especially interested in analyzing possible bottlenecks and limits that an architecture like BOINC presents. We will show a few practical examples of the simulator usage, with the subsequent analysis of the execution results. In this Section, we do not compare ComBoS with the current BOINC simulators described in Section 2 because our

simulator is the only one that can manage simulations that take into account the whole BOINC infrastructure.

The scenario used in the evaluation consists of a single project (one scheduling server, and different data servers) and three groups of Volunteer Nodes. The 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. All other parameters are fixed in all simulations, and the size of the output files has not been taken into account. Every execution in this section has simulated 100 hours. We have used the same host availability and unavailability parameters as those used in Section 4.2. In order to consider the randomness of our simulator, and to gather confidence into the meaningfulness of the results, each simulation result presented in this section is based on the average of 20 runs. For a 95% confidence interval, the error is less than $\pm$ 2% for all values.

6.1. Data Servers Load

Most volunteer computing participants use home computers, so the network traffic goes over the commodity Internet. BOINC scheduler request and reply messages are just about 10 KB, so this network traffic is exclusive about file upload and download (input and output files), which may be a performance issue [12]. VN download input files from data servers. In this case study we analyzed the load of the data servers of a single project through the download of input files.



(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 7: Data servers load with 30,000 VN (10,000 VN per group).

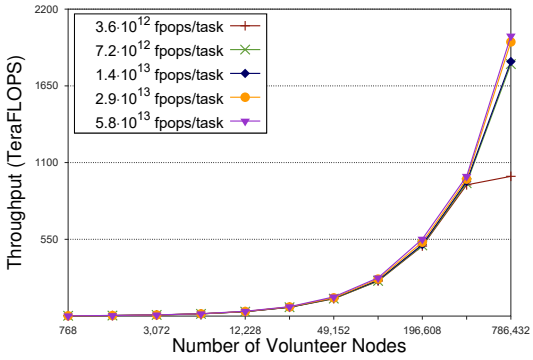
In our case study (Figure 7), we consider 30,000 VN (10,000 VN per group). In this experiment, our aim was to analyze the data servers' average load (CPU load) using different number of data servers in each simulation (from 1 to 64). We collected results from five input file sizes: 4 KB, 256 KB, 1 MB, 16 MB and 64 MB. The average number of floating-point operations required to

complete the computation of each task (fpops) in figure 7a are $3.6 \cdot 10^{12}$ (1 hour for a 1 GigaFLOP machine), $7.2 \cdot 10^{12}$ in figure 7b (2 hours for a 1 GigaFLOP machine), $1.4 \cdot 10^{13}$ in figure 7c (4 hours for a 1 GigaFLOP machine), and $2.9 \cdot 10^{13}$ in figure 7d (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.

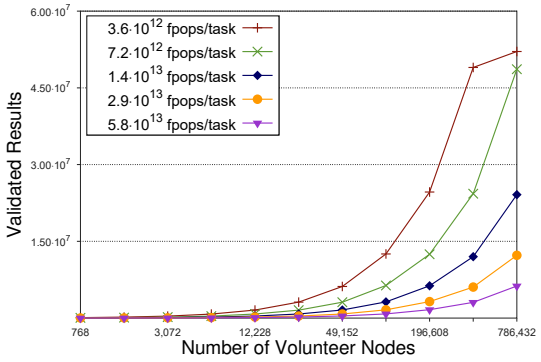
Using these results, we can estimate the average load of data servers in real scenarios. The larger the size of each input file and the lower the task duration, the greater the data servers' load becomes. Thus, in some simulations data servers become the bottleneck of the system. This type of simulations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

6.2. Combined Results

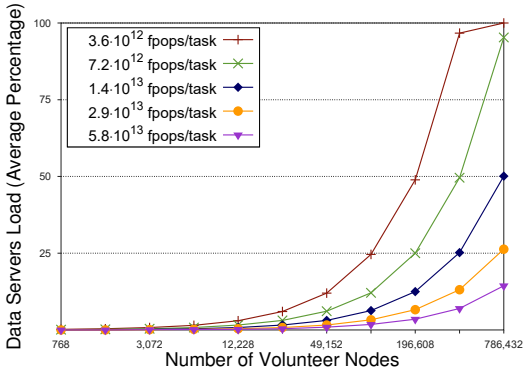
In the previous experiment we studied the scalability of the simulated environment by varying the duration of the tasks, the size of the input files, and the number of data servers. Now, we want to analyze the performance of the same infrastructure by increasing the number of VN, and setting the number of data servers to 4 and the input file size to 1 MB. The results of these simulations are shown in Figure 8. We have not included the network load results because there was no congestion in it.



(a) Throughput.



(b) Validated results.



(c) Data servers load.

Figure 8: Combined results.

The results obtained show that we can estimate the system's throughput (Figure 8a, constant for the same number of floating point operations executed) and the number of validated results (Figure 8b, inversely proportional to the task duration). In addition, as in the previous experiment, we can find out the load of the data servers (Figure 8c). As shown in Figure 8c, 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 8a) and validated results (Figure 8b).

7. Conclusions and Future Work

In this paper we have described the design of ComBoS, a complete simulator of BOINC infrastructures that, unlike other BOINC simulators, can simulate realistic scenarios taking into account the whole BOINC infrastructure: projects, servers, network, redundant computing, and volunteer nodes. The main objective of this simulator is to guide the design of BOINC projects. ComBoS has been implemented using SimGrid, a simulation-based framework for evaluating cluster, grid and P2P algorithms and heuristics. The paper describes the design of ComBoS, taking into account the server side and the client side. 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 demonstrated that ComBoS can guide the design of BOINC projects, using the simulator with different data intensive workloads and platforms in order to analyze possible bottlenecks and limits that an architecture like BOINC presents. As future work, we want to use the simulator in order to design new models and architectures for volunteer computing platforms.

Acknowledgements

This work has been partially funded by the grant TIN2013-41350-P of the Spanish Ministry of Economics and Competitiveness.

References

[1] Volunteer computing, Last visited (Dec. 2015).
URL `http://boinc.berkeley.edu/trac/wiki/VolunteerComputing`

[2] D. P. Anderson, BOINC: A system for public-resource computing and storage, in: 5th IEEE/ACM International Workshop on Grid Computing, 2004, pp. 4–10. `doi:10.1109/GRID.2004.14`.

[3] BOINCstats, Last visited (Feb. 2016).
URL `http://boincstats.com/en/stats`

[4] Top 500 supercomputer list, Last visited (Sep. 2016).
URL `http://www.top500.org/`

[5] C. Tapparello, C. F. B. Karaoglu, H. Ba, S. Hijazi, J. Shi, A. Aquino, W. Heinzelman, Volunteer Computing on Mobile Devices: State of the Art and Future, 2015.

[6] H. Casanova, A. Giersch, A. Legrand, M. Quinson, F. Suter, Versatile, scalable, and accurate simulation of distributed applications and platforms, Journal of Parallel and Distributed Computing 74 (10) (2014) 2899–2917. `doi:10.1016/j.jpdc.2014.06.008`.

[7] 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, IEEE Computer Society, Washington, DC, USA, 2004, pp. 4–10. `doi:10.1109/GRID.2004.14`.

[8] L. S. Collaboration, D. P. Anderson, Einstein@home search for periodic gravitational waves in early s5 LIGO data, Physical Review D 80 (2009) 4–15. `doi:10.1103/PhysRevD.80.042003`.

[9] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, E. Korpela, SETI@HOME-massively distributed computing for SETI, Computing in Science and Engineering 3 (1) (2001) 78–83. doi:10.1109/5992.895191.

[10] P. Paul, SETI@home project and its website, Crossroads 8 (3) (2002) 3–5. doi:10.1145/567162.567164.

[11] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, SETI@home: an experiment in public-resource computing, Commun. ACM 45 (11) (2002) 56–61. doi:10.1145/581571.581573.

[12] D. Anderson, E. Korpela, R. Walton, High-performance task distribution for volunteer computing, in: e-Science and Grid Computing, 2005. First International Conference on, 2005, pp. 8 pp.–203. doi:10.1109/E-SCIENCE.2005.51.

[13] A. P. Society, Einstein@home, Last visited (Apr. 2016).
URL http://www.einsteinathome.org

[14] Oxford University, Climateprediction.net, Last visited (Apr. 2016).
URL http://climateprediction.net

[15] Creating BOINC Projects, Last visited (Jan. 2016).
URL https://boinc.berkeley.edu/boinc.pdf

[16] A. Rowstron, P. Druschel, Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility, SIGOPS Oper. Syst. Rev. 35 (5) (2001) 188–201. doi:10.1145/502059.502053.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, B. Zhao, Oceanstore: an architecture for global-scale persistent storage, SIGARCH Comput. Archit. News 28 (5) (2000) 190–201. doi:10.1145/378995.379239.

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

[19] D. Kondo, Scheduling Task Parallel Applications for Rapid Turnaround on Enterprise Desktop Grids, Ph.D. thesis, University of California at San Diego La Jolla (2005).

[20] B. Donassolo, H. Casanova, A. Legrand, 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, ACM, New York, NY, USA, 2010, pp. 605–612. doi:10.1145/1851476.1851565.
URL http://doi.acm.org/10.1145/1851476.1851565

[21] T. Estrada, D. A. Flores, M. Taufer, P. J. Teller, A. Kerstens, D. 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, 2006, p. 88. doi:10.1109/E-SCIENCE.2006.261172.

[22] T. Estrada, M. Taufer, K. Reed, D. P. Anderson, EmBOINC: An emulator for performance analysis of BOINC projects, in: Parallel and Distributed Processing Symposium, International, 2009, pp. 1–8. doi:10.1109/IPDPS.2009.5161135.

[23] BOINC Jobs, Last visited (Jan. 2016).
URL https://boinc.berkeley.edu/trac/wiki/JobIn

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

[25] SETI@home, CPU performance of SETI@home volunteer computers, Last visited (Mar. 2016).
URL http://setiathome.berkeley.edu/cpu_list.php

[26] EINSTEIN@home, CPU performance of EINSTEIN@home volunteer computers, Last visited (Mar. 2016).
URL https://www.einsteinathome.org/cpu_list.php

[27] LHC@home classic, CPU performance of LHC@home volunteer computers, Last visited (Mar. 2016).
URL http://lhcathomeclassic.cern.ch/sixtrack/cpu_list.php

[28] B. Javadi, D. Kondo, J. Vincent, D. P. Anderson, Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home, Parallel and Distributed Systems, IEEE Transactions 22 (2011) 1896–1903. doi:10.1109/TPDS.2011.50.

[29] A. Legrand, Scheduling for large scale distributed computing systems: Approaches and performance evaluation issues, Tech. rep., Université Grenoble Alpes (2015).