A New Volunteer Computing Model for Data-Intensive Applications

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

Universidd Carlos III de Madrid, Department of Computer Science and Engineering, Computer Architecture Group, Leganés, Madrid, Spain

Abstract

Volunteer computing is a type of distributed computing in which ordinary people donate computing resources to scientific projects. BOINC is the main middleware system for this type of distributed computing. 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. This is why we have designed an alternative, using the same BOINC infrastructure, in order to improve the performance of BOINC projects that have reached their limit due to the I/O bottleneck in data servers. This alternative involves having a percentage of the volunteer clients running as data servers, called data volunteers, that improve the performance of the system by reducing the load on data servers. In addition, our solution takes advantage of data locality, leveraging the low network latencies of closer machines. This paper describes our alternative in detail and shows the performance of the solution, applied to three different BOINC projects, using a simulator of our own, ComBoS.

Keywords: BOINC; data locality; data volunteers; throughput; simulation; volunteer computing

1. Introduction

This paper is an extension of work originally reported in Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2016) [1].

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. BOINC (Berkeley Open Infrastructure for Network Computing) [2] is an open-source VC platform. It provides a complete middleware system for VC. In fact, BOINC is the most widely used middleware system.

Several BOINC projects, such as the CERN ATLAS@Home project [3], are currently experiencing a permanent saturation of their data servers [4]. This congestion is the result of the exchange of files

between the servers and the hundreds of thousands of volunteer users that participate in the project. As a result, the organizations that host these projects (such as universities and research centers) are forced to invest considerable amounts of money into infrastructures in order to meet the demand of users. This solution, which involves facing large disbursements in infrastructure, collides with the general purpose of VC, which consists on obtaining large computing capabilities by scientific projects by means of the participation of volunteer users.

The relevance of solving the problem is that VC is a really important paradigm for the following reasons [5]:

- Because of the huge number (> 1 billion) of computers in the world, VC can supply more computing power to science than any other type of computing. In 2011, the company Cisco Systems predicted that there will be 50 billion devices with Internet access by 2020 [6], so this means that the computing power provided by VC can increase over time.
  - Moreover, VC can also be used on mobile devices. In this kind of platforms, the BOINC application only computes when the device is plugged into a power source (AC or USB) and the battery is over 90% of charge, so it will not significantly reduce the battery life or the recharge time. Besides, BOINC transfers data only when the devices are connected to a WiFi network.
- VC power cannot 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.

Such is the magnitude of VC in our society, that there are projects that help fight climate change (Climateprediction.net), cure cancer or fight Alzheimer's (Rosetta@Home), or find extraterrestrial intelligence (SETI@Home). Solving the aforementioned problems would enable us to keep contributing to advances in scientific knowledge and solve common problems in society. On the contrary, not solving the problems would force VC projects to choose between two alternatives. The first one, mentioned previously, would be to invest more money into infrastructure, but not all organizations could cope with such an expense, and this would result in the termination of a large number of projects. The other alternative would consist on establishing a limit on the number of users, which would drastically curb the growth of these scientific projects. Dedicating time and resources to a type of computation without growth expectations would prompt users to abandon projects sooner or later [7, 8].

This paper proposes a new methodology for VC (especially applied to the current BOINC system), partially decentralizing the model without modifying the infrastructure, in order to obtain a better

throughput of the projects. This approach involves having a percentage of the volunteer clients running as data servers, called data volunteers. We have implemented and evaluated our solution using ComBoS[1] [9], an open-source VC simulator developed by the authors. The evaluation carried out consists on the analysis of different scenarios considering three different VC projects: ATLAS@Home, RNA World, and DrugDiscovery@Home. The results of the simulations demonstrate that our solution can improve the performance of the projects that have reached their limit due to the I/O bottleneck in data servers and also it can enhance the throughput of the projects by taking advantage of data locality.

More specifically, our work provides the following contributions:

- A new Volunteer Computing model, which uses a percentage of volunteer clients as data servers, called data volunteers, and does not require the modification of the current BOINC infrastructure.
- An implementation of the new proposed model in ComBoS, an open source simulator created by the authors, that can be used by BOINC designers in order to measure the performance of our approach in real projects.
- An explanation of why the aforementioned model helps to avoid the saturation of the servers in BOINC projects and how our solution leverages the data locality in file transfers between the clients and the data volunteers.
- An extensive simulation-based evaluation that demonstrates that our solution enhances the current BOINC model in terms of throughput.

The rest of the paper is organized as follows. Section 2 discusses the background and related work; Section 3 describes the current BOINC model, while Section 4 introduces our solution in detail; Section 5 presents and describes the simulator that we have developed as previous work; Section 6 analyzes the performance of our approach by performing multiple experiments using our simulator and considering three different BOINC projects; and finally, Section 7 concludes the paper and presents some future work.

2. Background and Related Work

The term Volunteer Computing (VC) was coined by Luis Sarmenta during his Ph.D. research [10]. BOINC [2] is the main middleware system for VC that makes it possible for scientists to design and operate public-resource computing projects. The applications supported by BOINC are diverse, and include communication and large storage data-intensive applications. In order for PC owners to become volunteers, they have to download and run a BOINC client program on their computers. Each volunteer can participate in multiple BOINC projects. If they choose to do so, they have the freedom to specify how

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

they would like their resources to be allocated among the projects. Examples of BOINC projects include Einstein@Home, Enigma@Home, LHC@Home, MilkyWay@Home, SETI@Home, and Universe@Home.

The BOINC architecture is based on a strict master/worker model; it has a central server that is responsible for dividing applications into thousands of small independent tasks. As the worker nodes request workunits, the central server distributes the tasks among them. If this server initiated communications, NAT (Network Address Translation) may arise from a bidirectional communication. For this reason, when a worker is ready to submit results or needs more work, it initiates communication. The centralized servers never initiates communication with worker nodes.

In the BOINC middleware, each client downloads input files from a fixed set of data servers. Clients upload their output files to these same data servers, after they have completed a computation. Large projects that include thousands of volunteer clients can suffer from bottlenecks when a large number of participants try to access the data servers, whereas CPU-intensive jobs that process small files do not face such problems. The BOINC middleware is thus appropriate for the latter. In projects like ATLAS@Home [3], each workunit requires large files (100 MB of input and output data), so a high number of volunteer participants can saturate the data servers. Some of the data-intensive projects that face the risk of saturation improve their performance by using file replication. An example of this is the Einstein@Home project [11], which uses large input files (40 MB). In this project, a large number of hosts may receive the same input file. This does not happen in other projects, such as SETI@Home [12, 13], in which all input files are different.

Data servers in the BOINC architecture [14] do not access the BOINC database: they are simply web servers. In addition, they can be located anywhere. BOINC-based projects such as Einstein@Home and Climateprediction.net [15, 16] have their servers at partner institutions, and these are replicated and distributed data servers. The download and upload traffic is spread across the commodity Internet connections of those institutions. These servers share data stored in disks, including file storage and relational databases. In order to ensure that only files with prescribed size limits are uploaded, data servers use a certificate-based mechanism to process file uploads. Downloads, however, are controlled by plain HTTP. The BOINC system uses a type of redundant computing where several clients are asked to perform the same computation, in order to contrast the results [17]. Only when a 'consensus' is reached, the results are deemed valid. In some cases, new tasks must be created and sent to the clients so as to perform the computation again.

Two relevant proposals from the literature are the WeevilScout framework [18] and the Comcute System [19]. Both solutions consist on using web browsers from anonymous users to perform master-slave VC tasks. In fact, the solution described in [19] proposes a multi-level volunteer computing architecture and it is similar to the approach introduced in [20], since both have volunteer users computing parallel executions. The use of VC systems for Big Data processing has been studied in [21]. In this article, the authors describe an architecture of intelligent agents to optimize Big Data processing. In [22], the

authors present a VC solution called FreeCycles, which supports MapReduce jobs. FreeCycles improves data distribution (among mappers and reducers) by using the BitTorrent protocol to distribute data, and improves intermediate data availability by replicating files through volunteers in order to avoid losing intermediate data. However, these solutions are not based in BOINC, and they have plenty of future challenges. We wanted to propose a new methodology based on BOINC, because it is the most relevant middleware for VC, and there are currently hundreds of thousands of volunteers participating in their projects.

In [23], in order to decentralize the data distribution of the BOINC architecture, the authors made use of the Attic File System (AtticFS), previously known as the Peer-to-Peer (P2P) Architecture for Data-Intensive Cycle Sharing (ADICS) [24]. This system is a decentralized P2P data sharing software similar to BitTorrent. AtticFS allows network access to distributed storage resources. The solution that we propose can avoid bottlenecks in the BOINC data servers. Input files downloaded by BOINC clients for processing are cached in order to make them available to be processed by other clients. This solution, however, requires the implementation of a centralized data lookup service in order to make available the list of BOINC clients that store the files to be processed, as well as the integration of AtticFS into the BOINC infrastructure.

The next sections describes our solution in more depth, highlighting the differences between previously proposed solutions and ours. The main advantage to our proposal is that it does not require the addition of any external elements to the BOINC infrastructure. Moreover, our solution attempts to decentralize VC servers by taking advantage of the concept of edge computing [25].

3. BOINC Description

In BOINC, servers are responsible for managing projects. The architecture of the server side is shown in Fig. 2. The server side of a project consist of two parts [17]:

- A *project back end* that supplies applications and workunits, and that handles the computational results. It includes: a *work generator*, which creates workunits and their 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*, which 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; 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.

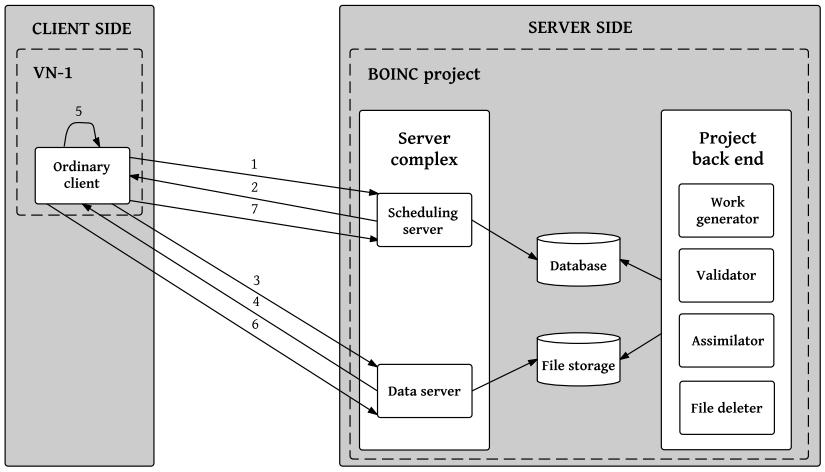


Figure 1: Current model.

Fig. 1 shows how BOINC currently works. It contemplates a scenario with just one Volunteer Node (VN-1: Ordinary client) and one project. When a volunteer (in this case, VN-1) wants to execute a task, it first makes a request to the Scheduling Server of the project (step 1 of Fig. 1). Then, the server answers with a workunit (step 2), which stores the computation to be performed and a list of addresses where the input files needed to perform the computation can be downloaded. Therefore, the next step for the client is to download the necessary files from the list addresses, which correspond to the data servers of the project (steps 3 and 4). Once the client has downloaded the files, it computes the task (step 5). When the execution is finished, the client simply has to send the output files generated during the execution to the data servers (step 6), and report the results obtained to the scheduling server (step 7).

In some projects, a large transfer of files (steps 3 and 4) has put a very high load of data on the servers, which has resulted in the number of running jobs reaching a plateau. The ATLAS@Home [3] project is an example of this issue.

4. Alternative to the Current BOINC Model

The following section consists of a detailed introduction to our alternative proposal. Since the alternative that we present includes two types of Volunteer Nodes (VN), which are the regular clients and the clients that work as data servers, we will use the terms “ordinary volunteers” or “ordinary clients” to refer to the VN of the current system; and “data volunteers” or “data clients” when referring to the new VN, which act as data servers.

4.1. Proposed Methodology

The aim of Volunteer Computing (VC) is that organizations be able to attain large computing power thanks to the participation of volunteer clients instead of a high investment in infrastructure. This is why we have designed a new methodology, using the same infrastructure, in order to improve the performance of BOINC projects that have reached their limit. This approach involves having a percentage of the VN running as data servers, called "data volunteers" (as we introduced before). Each file needed by a workunit must be replicated in N data volunteers (*dcreplication* attribute).

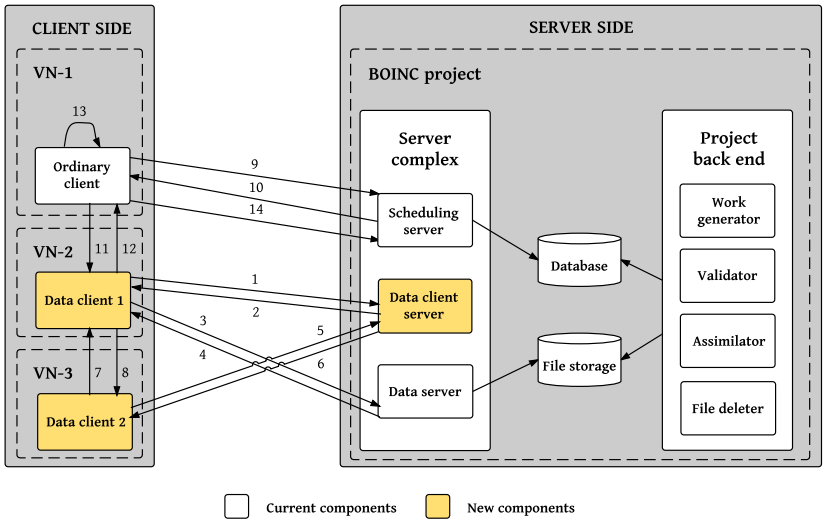


Figure 2: Alternative with data volunteers.

Fig. 2 shows the functioning of the system considering our methodology. It would only need another process on the server side (Data client server, Algorithm 1) and the data client software (Data client, Algorithm 2). Each data volunteer works as an ordinary VN (downloading input files) and as a server (sending input files) at the same time. Fig. 2 contemplates an scenario with three VN (Ordinary client, Data client 1, and Data client 2) and one project. First, Data client 1 requests and downloads some work from the Data client server (steps 1 and 2 of Fig. 2). Then, Data client 1 downloads the corresponding input files from the Data server (steps 3 and 4) and stores them in its file system. Data client 2 repeats the same process (steps 5 and 6), but this time it downloads the corresponding input files from Data client 1 (steps 7 and 8), which had downloaded them before (and it is the nearest data volunteer that has the files). Note that in this case, Data client 2 does not access the Data server, thus reducing its load. Now, Ordinary client wants to execute tasks in the regular way, so it requests and downloads some work from the Scheduling server (steps 9 and 10), but now it can download the corresponding input files from Data client 1, Data client 2, or the Data server, because the same input files are replicated in all of them. In our solution, ordinary clients only download workunits that need files that are replicated *dcreplication* times in data clients. In the example of Fig. 2, the client downloads the input files from Data client

1 (steps 11 and 12, because Data client 1 is the nearest data volunteer that has the files), executes the tasks (step 13) and returns the computation results to the Scheduling server (step 14). In some projects, it is necessary to upload the output files of the computation to the data server. In this case, each project should decide whether to upload the output files to the actual data servers or to the data clients.

Algorithm 1 The data client server process executes this code in order to meet the data volunteer requests.

```
1: function DATA_CLIENT_SERVER( )
2:     while 1 do
3:         DEQUEUE message from received_messages_queue
4:         if message.type == Request then
5:             ans = CREATE_ANSWER
6:             for each workunit w in current_workunits do
7:                 if w.status in progress and
8:                     w.ndata_clients < dcreplication and
9:                     (w.ndata_clients == 0 or w.ndata_clients_confirmed > 0) then
10:                     w.ndata_clients+ = 1
11:                     ans.workunit = w
12:                     break
13:                 end if
14:             end for
15:             SEND ans to client
16:         else if message.type == Confirmation then
17:             w = FIND_WORKUNIT(message.workunit)
18:             w.input_files_locations.PUSH(message.client_location)
19:             if w.ndata_clients == 1 then
20:                 CREATE target_nresults instances of w
21:             end if
22:         end if
23:     end while
24: end function
```

Normally, a workunit has a list of associated input files [17], and each input file is defined as a list of addresses from where it can be downloaded, giving priority to the data volunteers (we do not want to collapse the data servers). For example, in the previous case, the definition of an input file that has been downloaded by Data client 1 and Data client 2 should be the list {Data client 1 address, Data client 2 address, Data server address}. Obviously, the fewer volunteer clients that access the data servers, the better this system works. In some projects, each input file is shared by multiple workunits, but each workunit describes a different computation using the same file. For these cases, locality scheduling can be used. The goal of locality scheduling [14, 17] is to minimize the amount of data transfer to hosts by preferentially sending jobs to hosts that already have some or all of the input files required by those jobs. This is also our ideal scenario, because each input file might be downloaded from a data server only once, and from data volunteers the rest of the time. For instance, consider a project where each input file is shared by five workunits and there is only one data server. With our approach, only the first data client should download the file from the data server. In the current BOINC system, the same input file would have to be downloaded five times, one per workunit. With our alternative, the load on the data server would be reduced by a factor of five and allow for more VN in the system. In the next section we will show some examples of our approach, considering real scenarios. An advantage of our alternative is that

it can also be used when jobs do not share input files, unlike locality scheduling. For example, we can use our alternative to reduce the load on the data servers when the same job is sent to multiple VN in order to reach a consensus.

Algorithm 2 Data volunteers run the following code in order to download input files.

```
1: function DATA_CLIENT_ASK_FOR_FILES( )
2:     while 1 do
3:         if current_storage < max_storage then
4:             SEND request to data client server
5:             message = RECEIVE from to data client server
6:             if message.workunit then
7:                 message.workunit.input_files_locations.SORT_BY_NEAREST_LOCATION
8:                 for each url in message.workunit.input_files_locations do
9:                     if url is active then
10:                        SEND request to url
11:                        input_files = RECEIVE from url
12:                        STORE input_files
13:                        break
14:                    end if
15:                end for
16:                SEND confirmation to data client server
17:            else
18:                EXPONENTIAL_BACKOFF
19:            end if
20:            SLEEP until current_storage < max_storage (files are deleted)
21:        end if
22:    end while
23: end function
```

On the other hand, data clients need to keep the files for longer to feed them to other volunteer clients. This files should be deleted later using the BOINC file management [26]: a 'sticky' flag in a client file prevents it from being deleted, so it can only be deleted either from an explicit way or through a regular expression matching.

4.2. Availability of Volunteers

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. In order to encourage the computational work volunteers contribute to the system, BOINC projects grant credit to users. Credit has no monetary value; it is only a measure of how much a volunteer has contributed to a project (credits are calculated from the floating point operations that a volunteer has computed) [27].

We have defined a replication factor to face the availability issue in our approach. As is descibed in Algorithm 2 and as we have stated before, when both a data or a ordinary volunteer need to download a file, they access the nearest data volunteer; if the data volunteer is not available, it access the next location in the list. The last element(s) in the list is always the location of the data server(s) of the

project, so it is essential to define a replication value that prevents the ordinary clients from accessing the data server(s) of the project, as we show in the experiments performed.

4.3. Security Concerns

BOINC allows the project designers to use Secure Socket Layer (SSL) in their projects, so HTTPS (port 443) can be used in the log in processes. Besides, BOINC uses the ports 31416 and 1043 to exchange data, so the client has to unblock them if they are behind a firewall. In a similar way, the implementation of our approach must use specific ports that should be unblocked from the firewall to manage the access between clients. Apart from that, our solution needs to include the security solutions that have already been implemented in P2P (Peer-to-Peer) systems [28].

4.4. Advantages of our Solution

Our solution offers two potential advantages over the current BOINC model:

- Firstly, it forces the volunteer clients to download the files from other volunteer clients (in this case, from data volunteers). In this way, this new model helps the servers of the projects to prevent saturation. Although data volunteers do download input files from data servers, they only have to do it once for each file, whereas in the current system the same file has to be downloaded by multiple clients from the data servers.

- Secondly, in our solution, every time a data volunteer downloads the input files associated with a workunit, the project server stores the address and location of that volunteer in a special list of the corresponding workunit. Once an ordinary volunteer downloads a task, in order to download the associated files it first sorts the list elements according to the proximity of the data volunteers that store the input files and accesses the nearest data volunteer available. Therefore, our alternative takes advantage of data locality, since the volunteer clients work as an intermediate cache between the ordinary clients and the data servers.

5. Complete Simulator of BOINC Infrastructures

In order to study the performance of the methodology presented in the previous section, we have implemented this functionality in ComBoS. ComBoS [9] is a Complete simulator of BOINC Infrastructures developed by the authors and can be downloaded from `http://www.arcos.inf.uc3m.es/~combos/`. ComBoS has been implemented with the help of the tools provided by SimGrid [29]. The ComBoS architecture has two main parts: the server side and the client side. The specification of the networks that connect both groups is detailed in the client side.

5.1. Server Side

The server side consist of BOINC projects, where jobs are created and distributed to the clients. ComBoS allows for the definition of multiple projects. For each project, users must define the parameters described in Table 1. In addition, the new functionality includes a new server process, the *data client server*, as described in Section 4 (see Fig. 2 and Algorithm 1).

Table 1: Project parameters (those in bold are the ones introduced in the implementation of our approach).

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.
ndata_client_servers	Number of data client servers of the project.
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.
dsreplication	Number of replicas of each file in the data servers.
dcreplication	Number of replicas of each file in the data clients.
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.
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.
output_file_storage	Where to upload output files [0 -> data servers, 1 -> data clients]. Default is 0.

5.2. Client Side

In ComBoS [9], 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.

Table 2: VN group parameters (those in bold are the ones introduced in the implementation of our approach).

Parameter	Description
nclients	Number of VN of the group.
ndata_clients	Number of data volunteers 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).
st_distri	Storage fit distribution of the data volunteers (in case there is not a traces file).
pv_distri	VN power fit distribution (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 (in case there is not a traces file).
nv_distri	VN non-availability fit distribution (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.

The BOINC client implements two related scheduling policies: (1) CPU scheduling, and (2) Work fetch. Both policies are included in the ComBoS scheduling and are also detailed in [9]. The scheduling is based on a round-robin between projects, weighted according to their resource share. This scheduling is described in detail in [27]. In ComBoS, each client is implemented with at least three different threads: the client main thread, which updates the client parameters every scheduling interval; the work fetch thread, which selects the project to ask for work; and the execution threads, one per attached project, that execute the tasks. Our simulator is also complemented by the most important features of the real scheduler (deadline scheduling, long term debt, fair sharing and exponential back-off).

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 in Table 2. As indicated in Table 2 (VN group parameters), users can define the power and availability of the VN hosts via either a traces file or distribution functions. In this way, we can simulate groups of VN where each node can be any type of device: PCs, laptops, smartphones and tablets. In order to evaluate our approach, we have included the data client functionality described in Sect. 4.

5.3. Use Case

ComBoS can be used to perform simulations as in the scenario shown in Fig. 3. This figure shows seven groups of VN at different geographical locations. All these groups represent sets of client machines that access the same BOINC project.

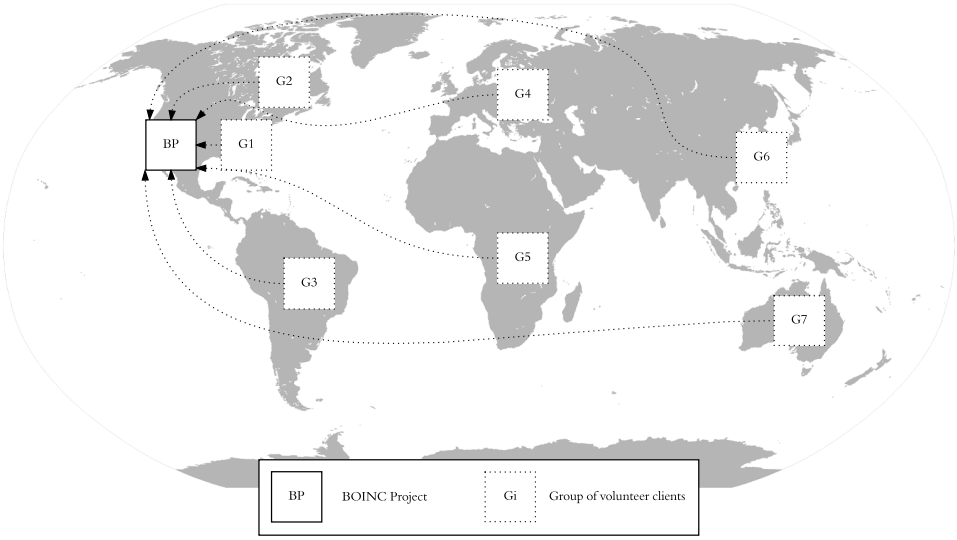


Figure 3: Use case using ComBoS.

To create a simulation, the user must define an scenario by specifying the parameters listed in Tables 1 and 2 in an XML file. 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.

5.4. Validation of the Simulator

To validate the complete simulator, we have relied on data from the BOINCstats website [30], 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 ATLAS@Home projects.

We have used the CPU power traces of the client hosts that make up the VN of each project. We have not used any other traces. In order to model the availability of the hosts, we used the results obtained in [31]. 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 noted 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 [31] too.

Table 3 compares the actual results of the SETI@Home, Einstein@Home and ATLAS@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 6.7% for credit/day and 2.3% for GigaFLOPS compared to the ATLAS@Home project. We consider that these results allow us to validate the simulator.

Table 3: 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
ATLAS@Home	17,795	3,206	7,183	1,035,840	7,018	966,786

6. Evaluation and Analysis Results

In this section we will present different test cases using ComBoS [9]. Our goal is to assess the performance of our approach, which involves some volunteer clients working as data servers, and has been described in Section 4. We are especially interested in analyzing the viability of this new model and comparing its performance to the current BOINC architecture. In particular, this section shows the results of different simulations of three different VC projects: ATLAS@Home, RNA World, and DrugDiscovery@Home, with the subsequent analysis of the execution results. These simulations have been performed using ComBoS, the simulator that we have presented in Section 5. We have used the same host availability and unavailability parameters as those used in Section 5.4. In order to account for the randomness of the simulations and to deem the results reliable, each simulation result presented in

this section is based on the average of 25 runs. For a 95% confidence interval, the error is less than ± 3% for all values.

6.1. ATLAS@Home

As described in Section 4, the ATLAS@home project uses a single download/upload server (host atlasathome.cern.ch). Each workunit in this project requires from 1 to 100 MB of input data, which are downloaded each time from the ATLAS@Home data server, and about 100 MB of output data, which are uploaded to the ATLAS@Home data server when each task is computed. This download/upload traffic is causing a problem in the server, as the current setup has reached its limit (mainly because of I/O) [4]. This is why we wanted to check this saturation problem using ComBoS. In order to define the current scenario, we have configured our simulations using as parameters those published in [3], [4], [32], and [33]. These parameters include the power, the geographical location and the total number of the machines involved in the project; the size of the input and output files; etc. We have evaluated the performance of the ATLAS@Home project in terms of throughput and server load (Fig. 4). As shown in this figure, data servers get saturated when there are about 1,200 Volunteer Nodes (VN), causing a severe deceleration in the system throughput. The results of Fig. 4 can be compared to the actual ATLAS@Home results published in BOINCstats [33].

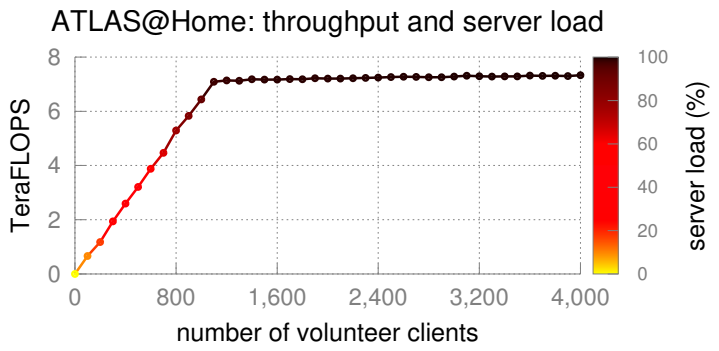


Figure 4: ATLAS@Home current performance (obtained using ComBoS).

The results presented demonstrate that the current number of machines involved in the system is not helping to improve the performance of the project, but is in fact saturating the data server. This shows that, at present, this project does not scale with the number of participants.

As introduced in Section 2, in our approach each input file must be replicated in one or more data volunteers (apart from the main server) depending on a replication factor. The results presented in this section try to show the throughput (in TeraFLOPS) and the load of the ATLAS@Home server using five different values for the replication parameter: from 1 to 5 (e.g. a replication of 3 means that each input file must be stored by three different data volunteers). We also compared these results with the results of the original system (without using our alternative). The storage capacity of the data volunteers of the

experiment follows a normal distribution with a mean of 20 GB of storage per host. All simulations were performed with 3,200 hosts, which is the current average number of active hosts of the ATLAS@Home project. Fig. 5 shows these results.

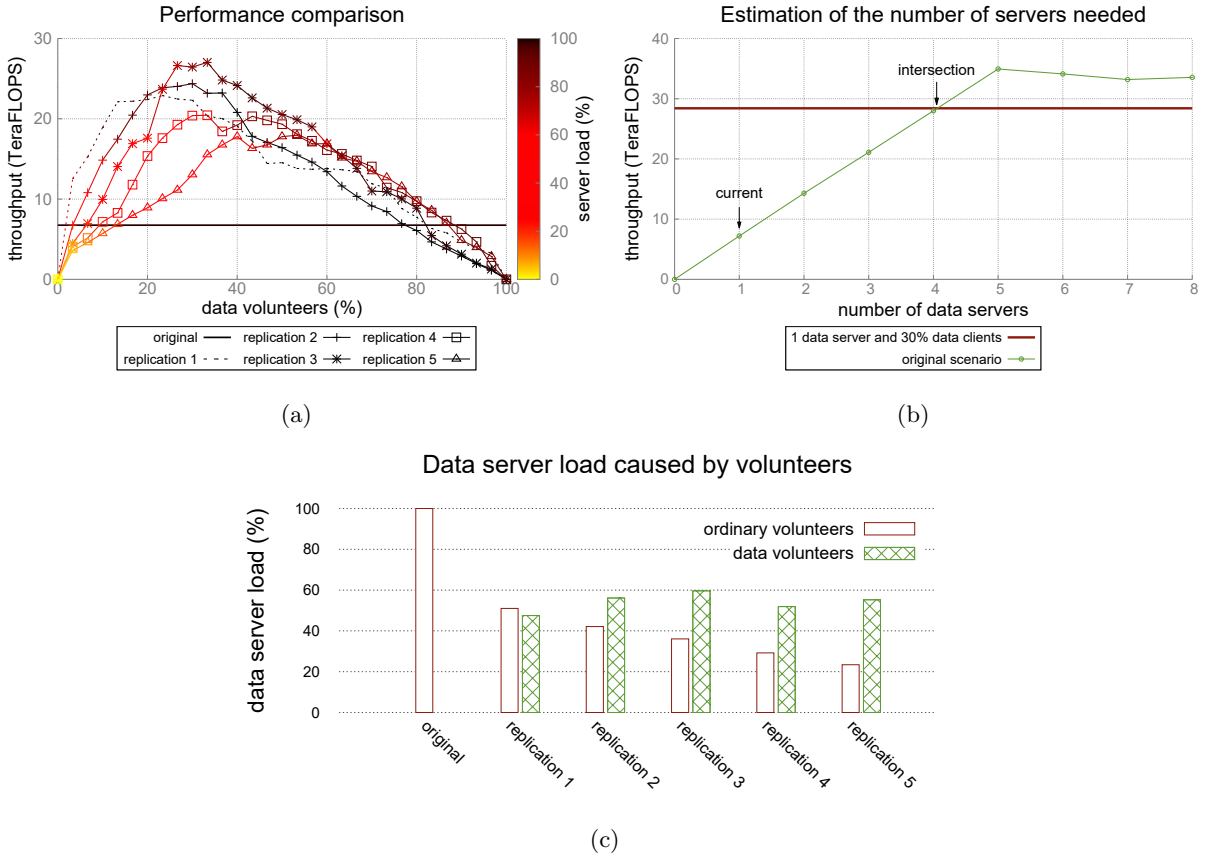


Figure 5: Performance of the ATLAS@Home project with 3,200 volunteer clients. Parameters that vary: total percentage of data volunteers, replication factor in the data volunteers, number of servers.

Specifically, Fig. 5a shows a comparison of the performance of the original system with our alternative by varying the replication factor and the percentage of data volunteers. As can be seen, for a certain percentage of data volunteers, the performance of our alternative considerably enhances the performance of the original system. For example, with a replication factor of 3 and an amount of data volunteers that ranges between 10% to 80% of the total number of VN, our alternative outperforms the original system in terms of throughput. This same figure also provides the server load in each simulation. The results obtained show that the load of the server using our proposed model is greatly reduced compared to the server load in the original scenario. Note that the higher the percentage of data volunteers, the larger the load on the server, because more data volunteers request input files from the server per unit of time. Consequently, this fact makes it necessary that, in order to implement this model, BOINC designers carefully choose the number of data volunteers of their projects. For its part, Fig. 5b shows the

number of servers that the original system would need in order to get the throughput of our alternative considering a replication factor of 3, the same server and 30% of the total clients working as data volunteers (intersection point in Fig. 5b). This figure shows that the ATLAS@Home project would need to improve its infrastructure by four to achieve the throughput of our model, which simply consists of modifying the client and server software.

Finally, Fig. 5c shows the server load at peak of throughput of each replication factor. This figure shows the server load caused by both the ordinary volunteers (OV) and the data volunteers (DV). As can be seen in the graph, the lower the replication factor, the greater the load that ordinary volunteers cause on the main server. This is because when an ordinary client tries to download a file from a data volunteer that turns out to be unavailable, the client then tries to download the file from the next host on the ordered input file address list, in which the last element is always the address of the data server. With a high replication factor (e.g. 5), there are more options to download the same input file, so the load that ordinary volunteers cause on the data server is solely due to the upload of output files. On the other hand, some projects, such as the ATLAS@Home project, validate each result just by checking that the corresponding output file exists, without checking the file content (the contents of the output are verified in a later merging process) [3]. Therefore, for future work, it would be interesting to analyze the performance of the project if ordinary volunteers upload output files to data volunteers instead of uploading them to data servers.

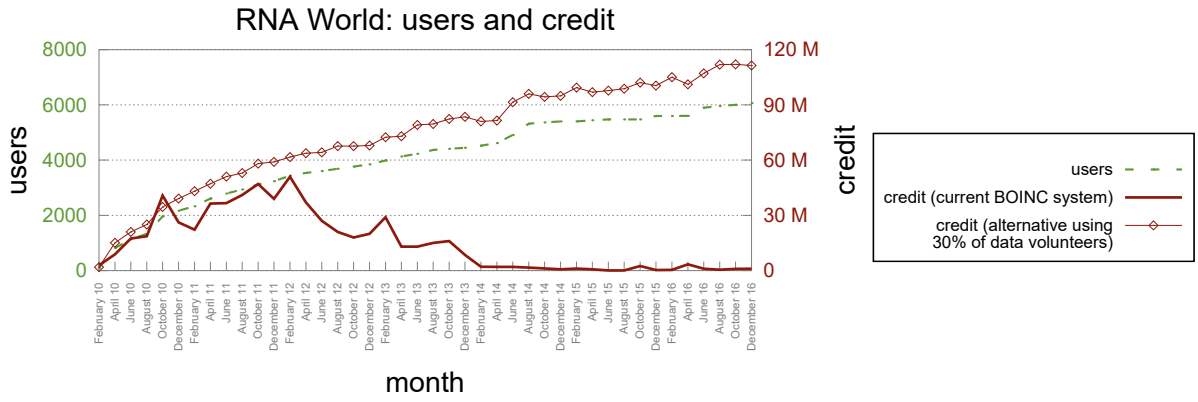
6.2. RNA World and DrugDiscovery@Home

We have also tested our solution in two other BOINC projects: RNA World and DrugDiscovery@Home. RNA World [34] uses VC in order to advance in RNA research, while DrugDiscovery@Home [35] tries to discover new drugs for the most dangerous and widespread diseases. As it is shown in Fig. 6, both projects are suffering a performance problem, since the increasing number of volunteers does not correspond to the project throughput (remember that credit is calculated from the amount of computation performed by the volunteers).

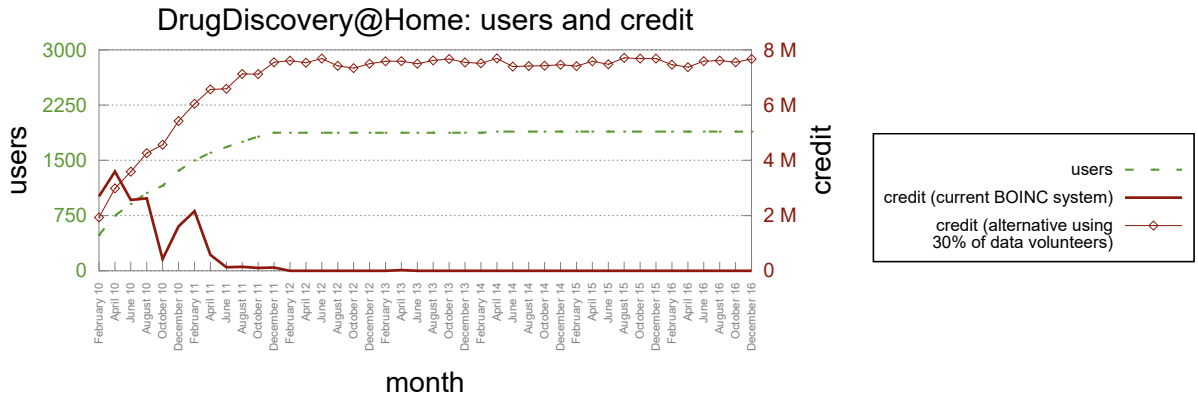
In addition, Fig. 6 also shows the credit both projects would have achieved if they had used our approach with a 30% of data volunteers. We obtained these results using ComBoS and, as can be seen, the throughput of both projects is almost directly proportional to the number of users, while with the current model the throughput is stuck.

We have again simulated both projects using ComBoS in order to evaluate the scalability of our approach. The results of the simulations are shown in Fig. 7. In these simulations we have considered again a 30% of the total number of users as data volunteers. The aim of these experiments is to measure the performance of our alternative applied to both projects by increasing the number of volunteers.

The results obtained show that our approach improves the current performance of both projects. In fact, our solution allows both projects to scale with the number of volunteers, something that is not hap-



(a)



(b)

Figure 6: Users and credit of the RNA World and DrugDiscovery@Home projects during last months (data obtained from BOINCstats [30]) and credit estimation considering our solution (data obtained using ComBoS).

pening with the current configuration. In particular, with our approach and considering 10,000 volunteers we obtain almost 2 times better throughput for the RNA World project and 5 times better throughput than in the current model for the DrugDiscovery@Home project. This is something unimaginable in the current model for the two projects, where more users do not guarantee better throughput (see Fig. 6).

7. Conclusion and Future Work

This paper has presented a new methodology that can improve the performance of VC, and it has been applied to different BOINC projects. Our proposal consists on selecting a number of volunteer clients to run as data servers (called data volunteers), and works in the same BOINC architecture (we only need to modify the client and server software). It also improves the throughput of the projects by taking advantage of data locality. We have evaluated the performance of our alternative using ComBoS, a simulator developed by the authors in a previous work. The results of the evaluation demonstrate that the use of data volunteers enhances the system throughput in two ways:

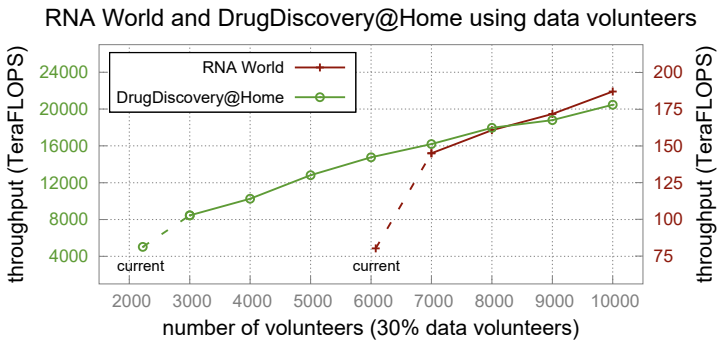


Figure 7: Scalability evaluation of our approach applied to the RNA World and DrugDiscovery@Home projects.

- Scalability: our solution forces the clients to download the input files from the data volunteers, rather than from project servers, thus reducing the load on those servers, and allowing for a larger number of users in the system, thereby increasing the scalability of the projects.

- Efficiency: once the clients download the list with the locations of the data volunteers that contain the required input files, the clients just access the nearest available data volunteer of the list. In this way, data volunteers work as an intermediate cache between ordinary clients and the project servers, taking advantage of data locality thanks to the low latencies of the networks that connect volunteers that are geographically close.

For future work, we want to explore variations of this solution; for example, a model in which the data clients can also run tasks. We are also interested in evaluating our solution in a real project, since the results of our simulations indicate that our approach can be a feasible alternative to the current model.

This work has been partially supported by the Spanish MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD under the project grant TIN2016-79637-P TOWARDS UNIFICATION OF HPC AND BIG DATA PARADIGMS.

[1] Alonso-Monsalve S, García-Carballeira F, Calderón A. Improving the performance of volunteer computing with data volunteers: A case study with the atlas@ home project. *Algorithms and Architectures for Parallel Processing*. Springer, 2016; 178–191.

[2] Anderson DP. Boinc: A system for public-resource computing and storage. *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, IEEE, 2004; 4–10.

[3] Adam-Boundarios C, Cameron D, Filipcic A, Lancon E, Wu W. ATLAS@Home: Harnessing Volunteer Computing for HEP. *21st International Conference on Computing in High Energy and Nuclear Physics*, CHEP2015, Okinawa, Japan, 2015.

[4] Cameron D. Atlas@ home. *Technical Report*, Worldwide LHC Computing Grid 2014.

URL https://indico.cern.ch/event/272793/contributions/1612806/attachments/490419/677935/atlasathome_pre-gdb_11.11.14.pdf.

[5] Volunteer Computing. http://boinc.berkeley.edu/trac/wiki/VolunteerComputing.

[6] Evans D. The internet of things: How the next evolution of the internet is changing everything. *Whitepaper, CISCO Internet Business Solutions Group (IBSG)* Apr 2011; .

[7] Durrani MN, Shamsi JA. Volunteer computing: requirements, challenges, and solutions. *Journal of Network and Computer Applications* 2014; **39**:369–380.

[8] Kurochkin I, Saevskiy A. Boinc forks, issues and directions of development. *Procedia Computer Science* 2016; **101**:369–378.

[9] Alonso-Monsalve S, García-Carballeira F, Calderón A. Analyzing the performance of volunteer computing for data intensive applications. *High Performance Computing & Simulation (HPCS), 2016 International Conference on*, IEEE, 2016; 597–604.

[10] Sarmenta LF. Volunteer computing. PhD Thesis, Massachusetts Institute of Technology 2001.

[11] Abbott B, Abbott R, Adhikari R, Ajith P, Allen B, Allen G, Amin R, Anderson S, Anderson W, Arain M, *et al.*. Einstein@ home search for periodic gravitational waves in early s5 ligo data. *Physical review d* 2009; **80**(4):042 003.

[12] Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D. SETI@home: an experiment in public-resource computing. *Commun. ACM* 2002; **45**(11):56–61.

[13] Paul P. SETI@home project and its website. *Crossroads* 2002; **8**(3):3–5.

[14] Anderson DP, Korpela E, Walton R. High-performance task distribution for volunteer computing. *First International Conference on e-Science and Grid Computing (e-Science'05)*, IEEE, 2005; 8–pp.

[15] Einstein@home. http://www.einsteinathome.org.

[16] Climateprediction.net. http://climateprediction.net.

[17] Creating BOINC Projects. https://boinc.berkeley.edu/boinc.pdf.

[18] Putra GHH. Workflow orchestration on weevilscout. PhD Thesis, Masters thesis, University of Amsterdam, 3 2013. https://staff. fnwi. uva. nl/asz belloum/MSctheses/thesis_Ganesh. pdf 2013.

[19] Czarnul P, Kuchta J, Matuszek M. Parallel computations in the volunteer–based comcute system. *International Conference on Parallel Processing and Applied Mathematics*, Springer, 2013; 261–271.

[20] Calderón A, García-Carballeira F, Bergua B, Sánchez LM, Carretero J. Expanding the volunteer computing scenario: A novel approach to use parallel applications on volunteer computing. *Future Generation Computer Systems* 2012; **28**(6):881–889.

[21] Balicki J, Korłub W, Paluszak J. Big data processing by volunteer computing supported by intelligent agents. *International Conference on Pattern Recognition and Machine Intelligence*, Springer, 2015; 268–278.

[22] Bruno R, Ferreira P. freecycles: efficient data distribution for volunteer computing. *Proceedings of the Fourth International Workshop on Cloud Data and Platforms*, ACM, 2014; 4.

[23] Elwaer A, Taylor I, Rana O. Optimizing data distribution in volunteer computing systems using resources of participants. *Scalable Computing: Practice and Experience* 2011; **12**(2):193–208.

[24] Kelley I, Taylor I. Bridging the data management gap between service and desktop grids. *Distributed and Parallel Systems*. Springer, 2008; 13–26.

[25] Garcia Lopez P, Montresor A, Epema D, Datta A, Higashino T, Iamnitchi A, Barcellos M, Felber P, Riviere E. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review* 2015; **45**(5):37–42.

[26] BoincFiles - BOINC. `https://boinc.berkeley.edu/trac/wiki/BoincFiles`.

[27] Anderson DP, McLeod J. Local scheduling for volunteer computing. *2007 IEEE International Parallel and Distributed Processing Symposium*, IEEE, 2007; 1–8.

[28] Bailes JE, Templeton GF. Managing p2p security. *Communications of the ACM* 2004; **47**(9):95–98.

[29] Casanova H, Giersch A, Legrand A, Quinson M, Suter F. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing* Jun 2014; **74**(10):2899–2917.

[30] BOINCstats. `http://boincstats.com/en/stats`.

[31] Javadi B, Kondo D, Vincent JM, Anderson DP. Discovering statistical models of availability in large distributed systems: An empirical study of seti@ home. *IEEE Transactions on Parallel and Distributed Systems* 2011; **22**(11):1896–1903.

[32] ATLAS@home Project Status. http://atlasathome.cern.ch/server_status.php.

[33] ATLAS@home project detail. `http://boincstats.com/es/stats/152/project/detail/`.

[34] RNA World. `http://www.rnaworld.de/rnaworld/`.

[35] DrugDiscovery@Home. `http://www.drugdiscoveryathome.com/`.