Improving the Performance of Volunteer Computing with Data Volunteers: A Case Study with the ATLAS@home Project

Saúl Alonso-Monsalve(✉), Félix García-Carballeira, and Alejandro Calderón

Department of Computer Science and Engineering, Computer Architecture Group, University Carlos III of Madrid, Leganés, Madrid, Spain
{saul.alonso,felix.garcia,alejandro.calderon}@uc3m.es

Abstract. Volunteer computing is a type of distributed computing in which ordinary people donate processing and storage resources to scientific projects. BOINC is the main middleware system for this type of 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 physical limit. This alternative involves having a percentage of the volunteer clients running as data servers, called data volunteers, that improve the performance by reducing the load on data servers. This paper describes our alternative in detail and shows the performance of the solution using a simulator of our own, ComBoS.

Keywords: BOINC · Data volunteers · Throughput · Simulation · Volunteer computing

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 [24]:

- 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 addition, this advantage will increase over time, because the number of computers is in continuous growth.

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

– 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.

BOINC (Berkeley Open Infrastructure for Network Computing) [3] is an open-source VC platform. It provides a complete middleware system for volunteer computing. In fact, BOINC is the most widely used middleware system. According to BOINCstats [11], currently there are 57 projects, with more than 13 million hosts participating in them. 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 [23] list of the 500 fastest supercomputers of the world. From the data storage point of view, 13 million hosts with an average of 25 GB available per client can provide a total capacity of 310 PetaBytes.

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 an alternative, using the same BOINC infrastructure, in order to improve the performance of BOINC projects that have reached their limit. This alternative involves having a percentage of the volunteer clients running as data servers, called data volunteers. We have evaluated the performance of our alternative using a simulator of our own, ComBoS [2].

The rest of the paper is organized as follows. Section 2 discusses related work; Sect. 3 presents our alternative to the current functioning of BOINC, using data volunteers; Sect. 4 presents and describes the simulator that we have developed; Sect. 5 analyzes the performance of our alternative, showing some case studies considering the ATLAS@home project; and finally, Sect. 6 concludes the paper and presents some future work.

2 Related Work

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

BOINC [3] is the main middleware system for VC 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.

The BOINC architecture is based on a strict master/worker model, with a central server responsible for dividing applications into thousands of small independent tasks and then distributing the tasks among 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.

The BOINC middleware uses a fixed set of data servers to provide input files to each client. Clients download input files from this set of data servers. Once the computation has been completed, they upload the output files to the same data servers. In projects with thousands of participants, the access to the data servers can form a bottleneck. The BOINC middleware is, therefore, appropriate for CPU-intensive jobs that process small files. Projects like ATLAS@home [1], in which each workunit requires large files (100 MB of input data and 50 MB of output data), a high number of volunteer participants can saturate the data servers. Some data-intensive projects use file replication to improve the performance; Einstein@home [21] 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 [5,22,25], in which each input file is different).

In the BOINC architecture [6] data servers can 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 [17] and Climateprediction.net [14]) use replicated and distributed data servers, located at partner institutions. The download and upload traffic is spread across the commodity Internet connections of those institutions. These components share data stored in disks, including relational databases and file storage. Data servers handle file uploads using a certificate-based mechanism to ensure that only legitimate files, with prescribed size limits, are 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], and the results are compared. Results are only validated when a ‘consensus’ is reached. In some cases new tasks must be created and sent to the clients to perform the computation again.

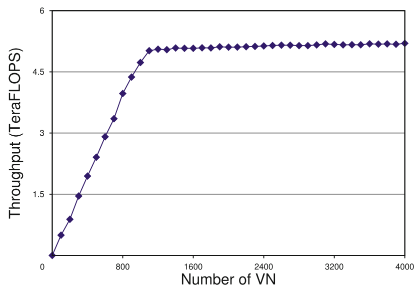
In [18], authors use the Attic File System (AtticFS), previously the Peer-to-Peer (P2P) Arhictecture for Data-Intensive Cycle Sharing (ADICS) [20], to decentralize data distribution in the BOINC achitecture. AtticFS is a decentralized P2P data sharing software for accessing distributed storage resources over the network in a similar way to BitTorrent. In this solution, when a BOINC client downloads input files to process, it caches them to be available to other clients, who can then process the same job. Although this solution can prevent the bottleneck in the BOINC data server, it requires the integration of AtticFS in the BOINC infrastructure, and a centralized data lookup service to obtain the list of BOINC clients that store the files to process. The use of VC systems for Big Data processing has been studied in [9]. In this article, the authors describe an architecture of intelligent agents to optimize Big Data processing. In [12], 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.

The main difference between our solution and the solutions described above, as described in the next section, is that our solution does not require that any external elements be added to the BOINC infrastructure. Besides, our solution uses the idea of edge computing [19], as it tries to decentralize VC servers.

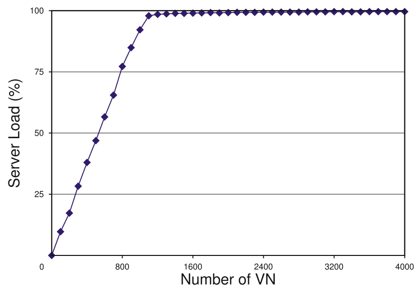
3 Alternative with Data Volunteers Using BOINC

Even though the clients volunteer for free, the manpower required to set up and maintain a BOINC project is not negligible. Nevertheless, it is only a fraction of the power needed for a regular Grid site. On the other hand, there are projects in which the number of running jobs has reached a plateau. An example is the ATLAS@home project [1]. This project uses a single server host [7] that includes all the functionalities of the BOINC server side: upload/download server, scheduler, file deleter, etc. In this project, each workunit requires about 100 MB of input data, which are downloaded each time from the ATLAS@home server, and 50 MB of output data, which are uploaded to the ATLAS@home server when each task is computed. This is causing a problem in the server, as the current setup has reached its limit (mainly because I/O). The ATLAS@home project team are exploring to use multiple data servers, which makes it necessary to improve the infrastructure. Another problem is that there are not enough volunteers joining the project, but the one server is already saturated, so the team deliberately do not advertise too much. We have evaluated the performance of the ATLAS@home project using ComBoS [2], in terms of throughput (Fig. 1a) and the load of the server (Fig. 1b). As shown in Fig. 1b, data servers get saturated when there are about 1,200 Volunteer Nodes (VN), causing a severe deceleration in throughput (Fig. 1a). In this section, we will present a solution for this problem.

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 an alternative, using the same infrastructure, in order to improve the performance of BOINC projects that have reached their limit. This alternative involves having



(a) Throughput (TeraFLOPS).



(b) Server Load (%).

Fig. 1. ATLAS@home current performance.

a percentage of the VN running as data servers, called data volunteers. Each file needed by a workunit must be replicated in N data volunteers (*dcreplication* attribute).

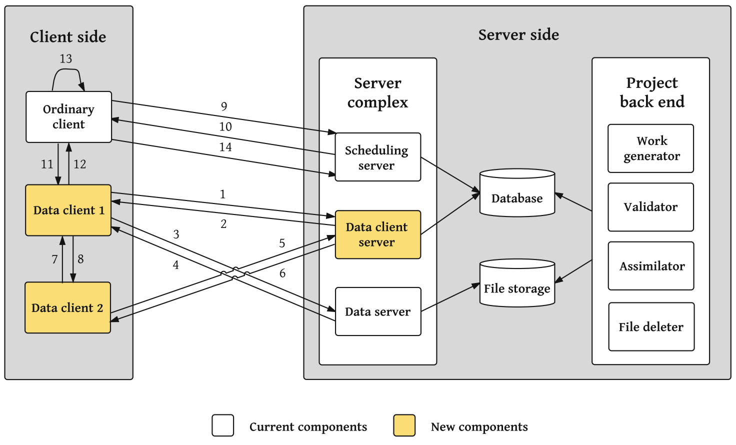


Fig. 2. Alternative with data volunteers.

Figure 2 shows the functioning of the system considering our alternative. It would only need another process on the server side (Data client server, Pseudocode 1) and the data client software (Data client, Pseudocode 2). Each data volunteers works as an ordinary VN (downloading input files) and as a server (sending input files) at the same time. Figure 2 contemplates an scenario with three volunteer nodes (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 (1 and 2). Then, Data client 1 downloads the corresponding input files from the Data server (3 and 4) and stores them in its file system. Data client 2 repeats the same process (5 and 6), but this time it downloads the corresponding input files from Data client 1 (7 and 8), which had downloaded them before. 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 (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 alternative, 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 (11 and 12), executes the tasks (13) and returns the computation results to the Scheduling server (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.

Pseudocode 1. Data client server dispatcher

```
1: function DATA_CLIENT_SERVER_DISPATCHER( )
2:     while 1 do
3:         POP message from $received\_messages\_queue$
4:         switch $message.type$ do
5:             case $Request$
6:                 CREATE_ANSWER $ans$
7:                 for each workunit $w$ in $current\_workunits$ do
8:                     if $w.status$ in progress and
9:                         $w.dataclients <$ dcreplication and
10:                        ($w.dataclients == 0$ or $w.dataclients\_confirmed > 0$) then
11:                        $w.ndata\_clients += 1$
12:                        $ans.workunit = w$
13:                    end if
14:                    $break$
15:                end for
16:                SEND $ans$ to client
17:            case $Confirmation$
18:                $w =$ FIND_WORKUNIT(message.workunit)
19:                PUSH(w.input_files_urls, message.client_address)
20:                CREATE $target\_nresults$ instances of $w$
21:    end while
22: end function
```

Normally, a workunit has a list of associated input files [15], 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 [6,15] 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 would also be 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 alternative, 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 data server would have five times less load and allow for more VN in the system. In the next section we will show some examples of our alternative, 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.

Pseudocode 2. Data client ask for files

```
function DATA_CLIENT_ASK_FOR_FILES( )
    while 1 do
        if $current\_storage < max\_storage$ then
            SEND $request$ to data client server
            $message$ = RECEIVE from to data client server
            if $message.workunit$ then
                for each $url$ in $message.workunit.input\_files\_urls$ do
                    if $url$ is active then
                        SEND $request$ to $url$
                        $input\_files$ = RECEIVE from $url$
                        STORE $input\_files$
                        $break$
                    end if
                end for
                SEND $confirmation$ to data client server
            else
                EXPONENTIAL_BACKOFF
            end if
            SLEEP until $current\_storage < max\_storage$ (files are deleted)
        end if
    end while
end function
```

4 Complete Simulator of BOINC Infrastructures

In order to study the alternative presented in the previous section, we have implemented the functionality in ComBoS. ComBoS [2] is a Complete simulator of BOINC Infrastructures developed by the authors. ComBoS has been implemented in C programming language, with the help of the tools provided by the MSG API of SimGrid [13]. In this section we describe the architecture of the simulator in the simplest possible way. We have divided all the 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 [10]:

Table 1. ComBoS new parameters (necessary with our alternative).

Server parameter	Description
ndata_client_servers	Number of data client servers of the project.
dsreplication	Number of replicas of each file in the data servers.
dcreplication	Number of replicas of each file in the data clients.
output_file_storage	Where to upload output files [0 -> data servers, 1 -> data clients]. Default is 0.
Client parameter	Description
ndata_clients	Number of data volunteers of the group.
st_distri	Storage fit distribution of the data volunteers:
	Weibull, Gamma, Lognormal, Normal, Hyperexponential

– 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.

4.1 Server Side

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 [15]:

– 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.

ComBoS allows for the definition of multiple projects. For each project, users must define the parameters described in [2] and in Table 1. In addition, we have included the data client server functionality described in Sect. 3.

4.2 Client Side

In ComBoS [2], 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:

– 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.

The scheduling is based on a round-robin between projects, weighted according to their resource share. This scheduling is described in detail in [4]. In addition, we have relied on the client scheduler code implemented in [16]. 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. However, our simulator is complemented with 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 [2] and in Table 1. To simulate VN groups using SimGrid, we have used the *cluster* entity. Like real clusters, each cluster contains many hosts interconnected by some 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. 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. In order to evaluate our alternative, we have included the data client functionality described in Sect. 3.

4.3 Validation of the Simulator

To validate the complete simulator, we have relied on data from the BOINCstats website [11], 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 and unavailability of the hosts, we used the results obtained in [8]. 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

Table 2. Validation of the whole simulator.

Project	Total hosts	Active hosts	*BOINCstats*		*ComBoS*	
			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	13,144	3,206	5,293	1,052,649	5,172	968,370

are a distribution with mean $\mu = -0.586$ and standard deviation $\sigma = 2.844$. All these parameters were obtained from [8] too.

Table 2 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 8.1% for credit/day and 2.3% for GigaFLOPS compared to the ATLAS@home project. We consider that these results allow us to validate the whole simulator.

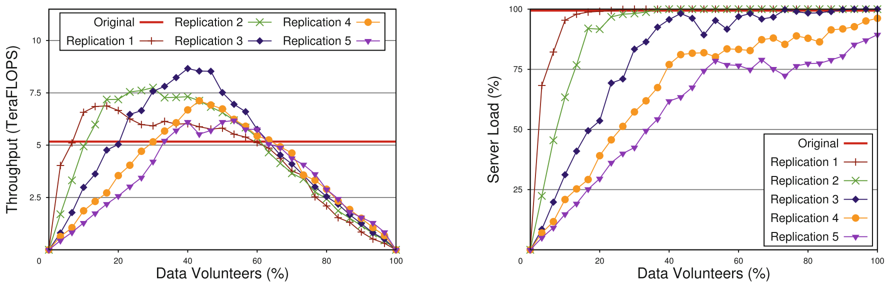
5 Evaluation and Analysis Results

In this section we will present different test cases using ComBoS [2]. Our goal is to assess the performance of our alternative, which involves some volunteer clients working as data servers, and has been described in the previous sections. We are especially interested in analyzing bottlenecks and limits that this architecture presents compared to the current BOINC architecture. We will show some practical examples of the ATLAS@home project using ComBoS, with the subsequent analysis of the execution results. We have used the same host availability and unavailability parameters as those used in Sect. 4.3. 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 ± 3% for all values.

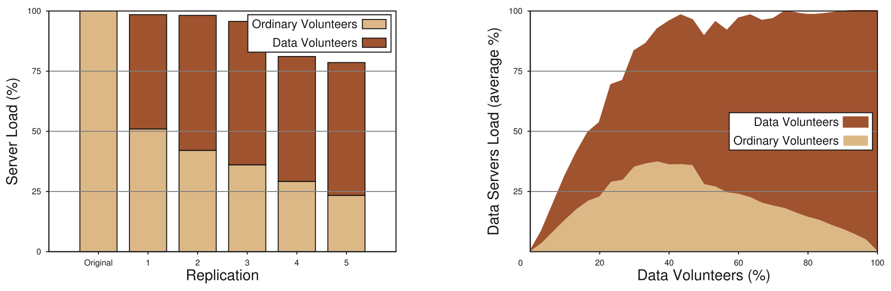
5.1 Files Replication

Input files can be replicated in one or more data volunteers, in addition to the main server. In this case study we analyzed the throughput (in TeraFLOPS) and the load of the server of the ATLAS@home project using five different values for the replication parameter: from 1 to 5 (number of data volunteers that must store a copy of each input file). We also compared these results with the results of the original system (without 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 number of active hosts of the ATLAS@home project. Figure 3 shows these results.

Figure 3a shows that for a certain percentage of data volunteers, the performance of our alternative considerably surpasses the performance of the original system. For example, with a replication factor of 3 and an amount of data



(a) Throughput (TeraFlops). **(b)** Server load (%).

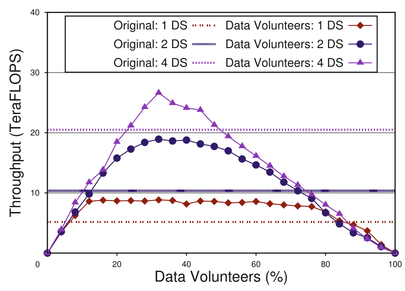


(c) Server load (% caused by ordinary volunteers and by data volunteers) at the peek of throughput.

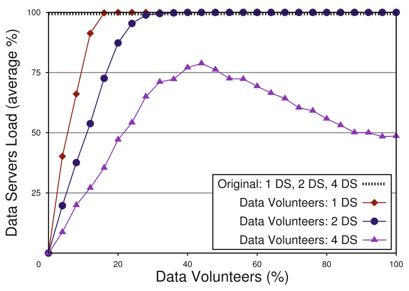
(d) Server load (% caused by ordinary volunteers and by data volunteers) with replication 3.

Fig. 3. ATLAS@home performance using our alternative and varying the input files replication in data volunteers (3,200 active hosts).

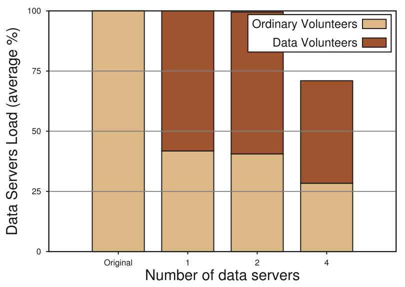
volunteers that ranges between 20% to 60% of the total number of VN, our alternative outperforms the original system in terms of throughput. Figure 3b shows the server load in each simulation. The results obtained show that the load of the server with our alternative is greatly reduced compared to the original server load. Note that the higher the percentage of data volunteers, the larger the load on the server is, because more data volunteers request input files from the server. This shows that in order to implement this model, designers must carefully choose the number of data volunteers. Figure 3c shows the server load at the throughput peek of each replication factor. This figure shows the server load caused by the ordinary volunteers and by the data volunteers. The lower the replication factor, the greater the load that ordinary volunteers cause on the main server, 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 input file address list, in which the last address is always that of the main 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. Finally, Fig. 3d shows in detail the server load for a replication factor of 3.



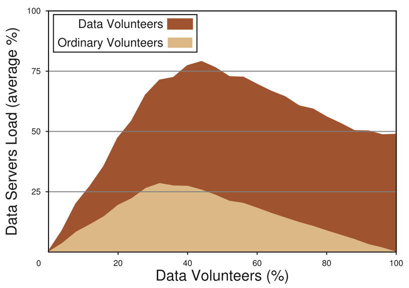
(a) Throughput (TeraFlops).



(b) Data servers load (%).



(c) Data servers load (% caused by ordinary volunteers and by data volunteers) at the peek of throughput.



(d) Server load (% caused by ordinary volunteers and by data volunteers) with 4 data servers.

Fig. 4. ATLAS@home performance using our alternative and varying the number of data servers (10,000 active hosts).

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. 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.

5.2 Data Servers

In this experiment, we have tested our alternative using different numbers of data servers. In these tests we have combined our alternative (usage of data volunteers) with an improvement of the infrastructure (increasing the number of data servers). We have set the replication parameter to 3 and increased the number of volunteer nodes to 10,000. The other simulation parameters were the same as in the previous experiment. Figure 4 shows the results of this experiment for 1, 2, and 4 data servers. Like in the previous experiment, we have focused our work on showing the throughput and the average load of the data servers of the system.

Like in the previous case, there is a range in the percentage of data volunteers in which the system throughput considerably outperforms that of the system without data volunteers (Fig. 4a). For example, with twice as many servers, our alternative renders a throughput of 18 GigaFLOPS, which doubles the throughput of the same system without our alternative. Furthermore, Fig. 4b, c and d show the average server load analogously to the previous experiment.

It is interesting to mention how a small improvement in the infrastructure, combined with our solution, can enhance the project throughput. In addition, our solution allows for more volunteers in the system.

6 Conclusion and Future Work

This paper has presented a solution in order to improve the performance of BOINC projects that have reached their limit due to the I/O bottleneck in data servers. This solution involves having a percentage of the volunteer clients running as data servers, called data volunteers, using the same BOINC infrastructure. We have evaluated the performance of our alternative using a simulator of our own, ComBoS. Our solution combines volunteer computing with peer-to-peer computing (P2P), since the data volunteers run as clients when downloading data files, and also as data servers when sending files to ordinary clients. To be implemented, our alternative needs to include the security protocols (for example, to traverse firewalls) that the P2P communications use. For future work, we want to analyze more case studies and use the simulator in order to analyze the energy consumption of the machines involved in a volunteer computing project.

References

1. Adam-Boundarios, C., Cameron, D., Filipcic, A., Lancon, E., Wu, W.: ATLAS@Home: harnessing volunteer computing for HEP. In: 21st International Conference on Computing in High Energy and Nuclear Physics, CHEP2015, Okinawa, Japan (2015)
2. Alonso-Monsalve, S., García-Carballeira, F., Calderón, A.: Analyzing the performance of volunteer computing for data intensive applications. In: 14th International Conference on High Performance Computing & Simulation, HPCS 2016, Innsbruck, Austria (2016)
3. Anderson, D.P.: BOINC: a system for public-resource computing and storage. In: 5th IEEE/ACM International Workshop on Grid Computing, pp. 4–10 (2004)
4. Anderson, D.P.: Local scheduling for volunteer computing. In: IEEE International Parallel and Distributed Processing Symposium, IPDPS 2007, pp. 1–8. IEEE (2007)
5. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: an experiment in public-resource computing. Commun. ACM **45**(11), 56–61 (2002)
6. Anderson, D., Korpela, E., Walton, R.: High-performance task distribution for volunteer computing. In: 2005 First International Conference on e-Science and Grid Computing, pp. 8–203 (2005)
7. ATLAS@home Project Status. http://atlasathome.cern.ch/server_status.php

8. Javadi, B., Kondo, D., Vincent, J.-M., Anderson, D.P.: Discovering statistical models of availability in large distributed systems: an empirical study of SETI@home. IEEE Trans. Parallel Distrib. Syst. **22**, 1896–1903 (2011)
9. Balicki, J., Korłub, W., Paluszak, J.: Big data processing by volunteer computing supported by intelligent agents. In: Kryszkiewicz, M., Bandyopadhyay, S., Rybinski, H., Pal, S.K. (eds.) PReMI 2015. LNCS, vol. 9124, pp. 268–278. Springer, Heidelberg (2015). doi:10.1007/978-3-319-19941-2_26
10. BOINC Jobs. https://boinc.berkeley.edu/trac/wiki/JobIn
11. BOINCstats. http://boincstats.com/en/stats
12. Bruno, R., Ferreira, P.: FreeCycles: efficient data distribution for volunteer computing. In: CloudDP 2014 Proceedings of the Fourth International Workshop on Cloud Data and Platforms (2014)
13. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. J. Parallel Distrib. Comput. **74**(10), 2899–2917 (2014)
14. Climateprediction.net. http://climateprediction.net
15. Creating BOINC Projects. https://boinc.berkeley.edu/boinc.pdf
16. Donassolo, B., Casanova, H., Legrand, A., Velho, P.: Fast and scalable simulation of volunteer computing systems using SimGrid. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC 2010, pp. 605–612. ACM, New York (2010)
17. Einstein@home. http://www.einsteinathome.org
18. Elwaer, A., Taylor, I.J., Rana, O.: Optimizing data distribution in volunteer computing systems using resources of participants. Scalable Comput.: Pract. Exp. **12**, 193–208 (2011)
19. García-López, P., Datta, A., Barcellos, M., Montresor, A., Higashino, T., Felber, P., Epema, D., Iamnitchi, A., Riviere, E.: Edge-centric computing: vision and challenges. ACM SIGCOMM Comput. Commun. **45**, 37–42 (2015)
20. Kelly, I., Taulor, I.: Bridging the data management gap between service and desktop grids. In: Kacsuk, P., Lovas, R., Nemeth, Z. (eds.) Distributed and Parallel Systems In Focus: Desktop Grid Computing. Springer, Heidelberg (2008)
21. LIGO Scientific Collaboration, Anderson, D.P.: Einstein@Home search for periodic gravitational waves in early S5 LIGO data. Phys. Rev. D, 80, 042003 (2009)
22. Paul, P.: SETI@home project and its website. Crossroads **8**(3), 3–5 (2002)
23. Top. 500 Supercomputer list. http://www.top500.org/
24. Volunteer Computing. http://boinc.berkeley.edu/trac/wiki/VolunteerComputing
25. Werthimer, D., Cobb, J., Lebofsky, M., Anderson, D., Korpela, E.: SETI@HOME–massively distributed computing for SETI. Comput. Sci. Eng. **3**(1), 78–83 (2001)