Simulation of Volunteer Computing in a Desktop Grid System

Ksenia Petrenko[1] and Ilya Kurochkin[2](✉)

[1] National Research University Higher School of Economics, Moscow, Russia
kseniapetrenko6@gmail.com

[2] Institute for Information Transmission Problems of Russian Academy of Sciences, Moscow, Russia
qurochkin@gmail.com

Abstract. BOINC is a platform for volunteer computing (VC) developed by the University of California, Berkeley. It allows scientific workflows to be performed on non-dedicated devices. The benefits this platform provides include low cost and computational power which is comparable to that of supercomputers. BOINC development can be facilitated with help of hypothesis testing in simulators of volunteer computing. In this work one of them was explored which is stated to be the most complete BOINC simulator. We describe our experience with it, what modifications have been made and how it was validated using the real data gathered by RakeSearch project, along with running experiments in it.

Keywords: Volunteer Computing · BOINC · ComBoS · Simulator

1 Introduction

BOINC [1] is a system for performing scientific tasks using volunteers' devices. These devices can only be utilized partially and for a limited amount of time, they are highly heterogeneous and their performance is hard to predict. Nevertheless, they have significant potential [2] and their cumulative processing power currently reaches the order of PetaFLOPs. In addition, it costs less than renting hosts in the cloud [3]. Thus, BOINC is appealing to scientific teams who have large amount of computation to be done. BOINC specializes in high-throughput computing, and the teams need to express their jobs as sets of parallel, independent tasks [5].

Testing hypothesis can be essential for the development of a system [20]. For projects like BOINC, it becomes difficult due to the large number of hosts participating in computing and their unpredictable behavior. In these cases simulators and emulators are used [4]. An emulator fully describes the system by exploiting the code of the real platform. However, it may include unnecessary components that only make the emulator more complex for researchers. A simulator is simpler and easier to set up, but it requires maintenance to stay up to date with the real system. Otherwise, it becomes obsolete over time.

An important detail of a simulator is its flexibility, i.e., how easy it is to add new code or change a part of the system. This is helpful for maintaining the simulator, as well as for studying BOINC. Documentation and neat code are key components here.

In addition, a simulator needs to work correctly no matter what its input data or configuration is. It is possible that the data passed to the tool differs from the real data, and the simulator has to process it without crashing due to an error. Otherwise, researchers won't be able to test any arbitrary hypothesis.

Finally, the closer a simulator is to the real system, the more parts can be investigated and taken into account. However, this closeness can increase the simulation time.

2 Background

2.1 BOINC

BOINC provides ready code for both the server and client sides. A team that wants to use BOINC needs to express a computing job as a set of independent tasks. The volunteers' devices cannot be blindly trusted, as tasks may never finish, finish with incorrect or correct results [6]. Each task can be replicated to increase the likelihood of completion and to compare results across different hosts. The latter can be used when there is no way to prove the correctness of the returned output. In this case, results can be compared with each other to gather a quorum with the same values. Later in the article, we will refer to a job as a workunit and its replicas as instances of a workunit.

The scientific team needs to set up the server side, which consists of a scheduler, assimilator, validator, and database, where the project's state is saved. The state includes information about various types of project jobs, created and dispatched tasks, as well as parameters of participating hosts, such as average CPU power, available disk space, etc.

2.2 Tools for Researchers

SimGrid [7] is a general library used for simulations of distributed systems (hosts, network, communication, computations). Many authors chose it for their own simulators.

Both emulators and simulators have been implemented for BOINC. However, many simulators are not available for research, as it will be seen. ComBoS [8] is known as the most complete simulator. It enables the setup of both client and server sides using a configuration file containing projects and client clusters characteristics. Another tool, SimBOINC, focuses on the client side, but unfortunately, the code is no longer available. [9] describes ways to improve SimGrid's performance in VC simulations, what can be useful in general even though the code isn't accessible. SimBA [10] utilizes BOINC project traces to simulate components. It allows researchers to investigate server scheduling but not client scheduling. Furthermore, the code is not provided.

EmBoinc [11] is an emulator that reuses a project's server code and simulates clients using collected traces. The maintenance effort is minimal. The limitations of EmBoinc include its ability to set up only a single project and the absence of client scheduling.

2.3 ComBoS

ComBoS is one of the simulators with available source code. It uses SimGrid to model hosts, network and tasks. All parameters for simulations are set in the configuration file. For each project, they are task duration, input and output file sizes, quorum size, acceptable delay in execution, and other parameters. The client side is represented as clusters with hosts that request input files and download them and other hosts that execute tasks. When hosts' CPU powers may be passed as a list, disk capacity, hosts' availability and nonavailability are simulated using distributions.

ComBoS measures the number of FLOPs computed, the number of sent and received results, how much of them are valid, those with errors, and those with missed deadline. Additionally, it shows the load on scheduling and data services.

It can be helpful to note, that in ComBoS tasks can be halted if there are others with higher priority (for instance, deadline is close). Halted tasks then are resumed from the moment when they were stopped, not from the beginning. There is no system with checkpoints.

Several limitations may prevent the use of ComBoS. Hosts are generated with only one CPU, therefore there is no CPU scheduling. Secondly, when a client chooses what tasks to request, it follows a "debt-based" policy, which has been deprecated in BOINC. Lastly, there are two types of clients in ComBoS's cluster: ones that download input files and others that perform computations. In the real world, each volunteer's host both downloads data and computes tasks.

3 Running the Simulator

Given all options, using ComBoS as a BOINC simulator is reasonable. In the next part, our experience with it will be described.

The first difficulty was associated with installing SimGrid to run the simulator. SimGrid is an actively developing project, and at the moment of writing this article, the 35th version had been released, while ComBoS supported only the 11th one. There are notes in Changelog that bugs have been fixed since the 11th version. It hasn't been checked but this could potentially impact the simulation. Even after installing the 11th version and making some modifications to the configuration file, the simulator failed with an error in the SimGrid code. That's why this dependency was switched to the latest version. This involved updating the old API, which was deprecated, to the new one.

The original simulator was implemented in a single file with 4000 lines of code in the C programming language. The code was written in a highly asynchronous manner using message queues. This made difficult to understand the logic behind the code. Besides this, there was no documentation and very few comments. For instance, there were components such as data client, data server and data client server with no description of what each of them does. The language was switched to C++. Also, the main file was split into the structured project, and comments were added to describe each components. The block diagram with workflows was drawn to aid in understanding [12].

Moreover, after altering simulation's parameters in the configuration file, the program crashed with segmentation faults. The code had several undefined behaviors (UB) and bugs. Many of them were found with valgrind [13]. At some point during the development, the program stopped crashing when the configuration file was changed. Although it still happened later during experiments.

There was another strange behavior in ComBoS. When a client requested a certain amount of work in FLOPs from a scheduling server, the scheduler selected one instance of an available workunit and duplicated it until the total computation cost met the request. It was changed to the policy where the server chooses several instances of different workunits that sum up to the requested FLOPs.

As it can be seen, ComBoS was hard to understand and modify, and it didn't work with some configurations. In addition, at the current moment, maintenance is stopped because ComBoS implements a "debt-based" scheduling strategy that was removed many versions ago in BOINC.

To verify the modified simulator, it can be compared with the original version. However, there are several factors that may cause difference between simulations. First of all, it's the SimGrid version: as this dependency is crucial for ComBoS, differences between its versions may have a significant impact. Both initial ComBoS and the modified version were run, using the configuration file provided in the git repository. In the original version, network was not a bottleneck, but in the modified one, it was. Next, the logic behind the code was changed because it was incorrect in some places. For instance, the authors of ComBoS calculated a task's deadline since its creation time rather than its sending time. Thus, many tasks were marked as received too late. Another example was that if the scheduler didn't manage to send even a single task, the client stopped querying the project altogether. In the BOINC the client retries its request with exponential backoffs so that when available tasks appear again, the client can continue to work.

4 Verifying the Simulator

Another method to verify the simulator is to test its work with data from a real project. In our case, the data was provided by RakeSearch [14,15]. It includes records from the project's database over the week. Firstly, it is necessary to calculate dataset features in order to set up a configuration file for a simulation.

Figure 1 illustrates tasks that were created, computed and sent back to the server. The green lines mark the period when a large amount of them were created and executed. Particular this period was simulated with ComBoS. The red lines indicate the period when many tasks were received after their deadlines.

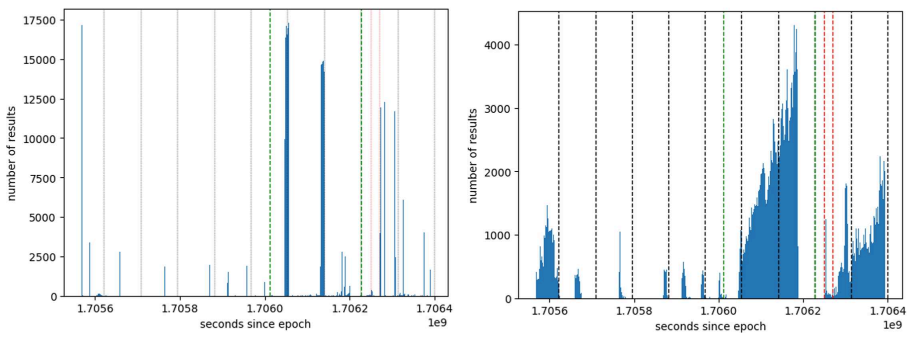

Fig. 1. Tasks created during the period (left) and tasks with correct results received by the server (right)

N clients, which execute tasks, and N data clients, which download input files, were set up in order to simulate N real clients.

Parameters for CPU (power and quantity) can be found in the database. ComBos simulated only one CPU per host. Several approaches can be tried - either increasing the power by a factor of k, where k is the number of CPUs, or duplicating the host k times. This will be investigated further. For now, each host is configured with only one CPU.

One of the tricky moments was with the *gbw* parameter - the bandwidth of the network in the cluster of clients. An incorrect value can easily make the network a bottleneck, which doesn't accurately represent the real-world scenarios. With the default value of 10Mbps, the number of finished tasks was 3336 for 100 clients and 3328 for 200 clients. After that, the *gbw* was set to 50 Mbps and observed 7434 tasks for 100 clients and 14250 for 200 clients. This is more plausible because doubling the number of clients results in a doubling of tasks. If it is required to test a bottleneck, a different network configuration can be used and the SimGrid documentation can be viewed [16] in addition to the code.

The next parameter is related to a model of availability. The data from RakeSearch didn't include exact periods of availability and unavailability. It provided only the percentage of time each host was available for computations. There is a paper [17] where authors investigated these periods using data from SETI@Home and derived distributions for them. The necessary distributions were supported in ComBoS to enable the generation of these periods for hosts. Then the percentages of time each host executed tasks were calculated, sorted by values and plotted along with the parameters from the dataset (Fig. 2). The distributions were sufficiently close.

The last parameter is the amount of disk space dedicated for the project. Initially, it was generated for each client by setting a distribution in the config, but it was inconvenient as the distribution is hard to fit to the real data. Now users can create a file with specified values to configure this parameter, similar to CPU power.

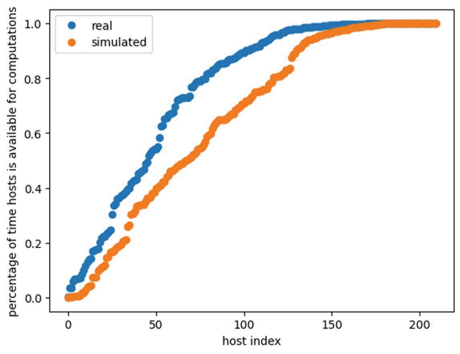

Fig. 2. The percentage of time when hosts were available for computation in the simulator and from the database

Unfortunately, the fraction of a volunteer's computational power that was dedicated to the project was unknown. This amount is calculated via the *resource_share* parameter, which is set up by a user for each project they participate in. There was no information about other projects, therefore it was not possible to calculate the fraction. If the simulation results closely match the characteristics of the real data in terms of order, it will be satisfying.

Finally, the simulation was run and metrics were compared (table 1).

Table 1. Metrics for comparison simulated and real data

Result state	number of results in the dataset	number of results in the simulation
Results valid	176,620	87,759
Results failed	1228	894
Results too late	1051	2

It appeared that the simulator produced valid results that are half of what is desired, with a total computation of $8.767 \cdot 10^7$ GigaFLOPs compared to $2.035 \cdot 10^8$ GigaFLOPs in the dataset. The main reason for discrepancy may be that only one CPU was simulated, whereas tasks on the real clients run simultaneously on multiple CPUs. Results received too late differed a lot too, prompting us to investigate the cause.

5 Characteristics of the Simulator

5.1 Resource Share and Timelines

An experiment was conducted involving two projects: one with long tasks and other with short ones. Different *resource_share* parameters were assigned for these projects so that they sums up to the same number. There was only one client for simplicity. The seed for a random generator was the same for each run. The generator was used to get periods of availability and unavailability for clients. The expectation was that increasing the *resource_share* of one of the projects would lead to an increase in the number of completed tasks for that project. Instead, the results are shown in the Fig. 3.

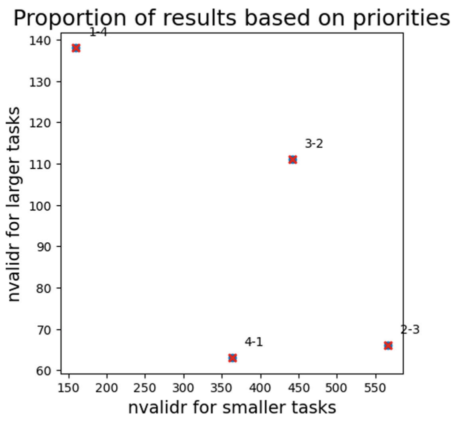

Fig. 3. Amount of tasks, executed for each of the projects during different runs of the experiment

The X-axis (OX) represents the number of tasks completed for a project with shorter duration, the Y-axis (OY) - with longer tasks. Labels follow the format $\{resource\_share\_quick\} - \{resource\_share\_long\}$. It was expected that the higher the *resource_share*, the more tasks would be computed for a project. However, the graph illustrates that points aren't ordered and the maximum value for the project with shorter tasks is reached when the *resource_share* is equal to two, rather than four.

Up to that point, there was no instrument to understand precisely how a host executed tasks, so it was developed. Logs were added to track what each host did at each moment of time (what project it executed, or if it's idle or unavailable) and recorded this information in a file. Subsequently, a Python script processed these logs and generated a timeline similar to ones shown in the Fig. 4.

The Y-axis represents the *resource_share* for a project with shorter tasks, X-axis - time in the simulation in seconds. Green segments denote the quick project, blue ones - the project with longer tasks. Red color means that the host was available but wasn't busy with any project. Black segments represent periods of unavailability. In short, our expectation was that the higher timeline, the more green color it would have. However, different results were obtained.

Moreover, periods of unavailability were different, even if the same seed was set for each run. It seemed that not only *resource_share* varied between runs. Upon debugging, it was found out that a single random generator was

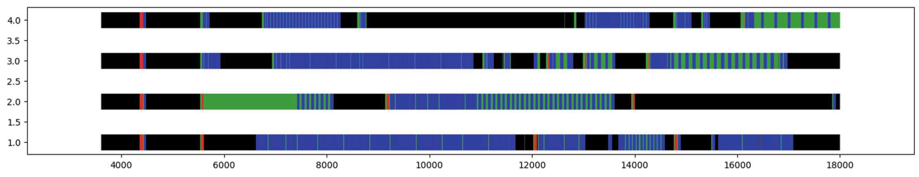

Fig. 4. Timelines for runs of the experiment

used for both type of clients and it was a problem. When the *resource_share* was changed, it affected the behavior of data clients, which downloaded input files, causing the generator to be called more or less frequently across different runs. That's why it also generated different period of unavailability for clients, which executed tasks, and different black segments were seen in timelines. By employing multiple random generators, the situation was improved (Fig. 5).

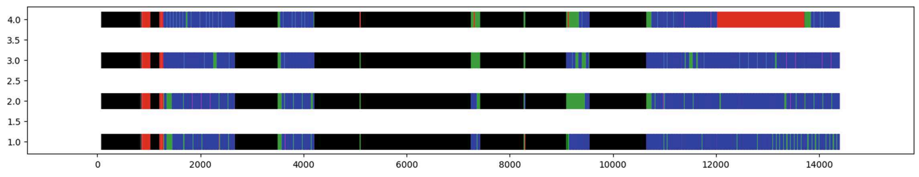

Fig. 5. Timelines for runs of the experiment

Finally, the periods of unavailability were the same, but there was another problem - the large red segment in the forth run. This indicated that the host didn't perform computations. After a small investigation, the code was fixed and it disappeared.

Lastly, it still was not seen that the higher the timeline, the more green segments appear. It turned out that the short tasks were too small initially, and when the duration of tasks was scaled up, the desired behavior was finally achieved (Fig. 6).

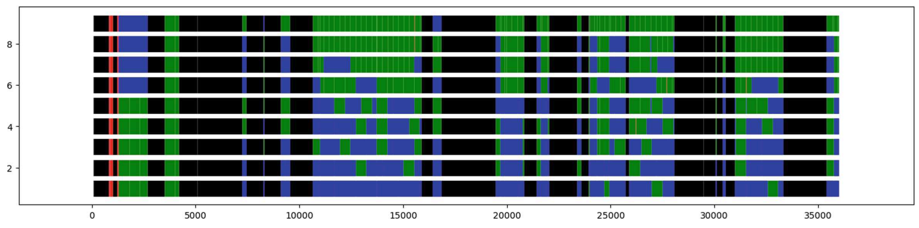

Fig. 6. Timelines for runs of the experiment

In summary, several places in ComBoS were fixed thanks to these timelines. They were used a lot during other experiments to monitor the activities of hosts.

5.2 Results Sent Too Late

During verification the simulator generated too few results received too late. It was decided to investigate why there were so many results in the dataset with this state.

Working with the same dataset, it can be shown that these tasks were computed around the same time as others, but they remained on hosts for much longer. Besides that, they were more of an exception than a rule: hosts sent far fewer results too late compared to those that met deadlines. These late results usually were received by clients right before the end of availability period.

Tasks in RakeSearch aren't too long, typically lasting up to 15 min, and their deadlines set for 4 h. That's why the situation with missed deadlines doesn't have a large impact on the system - after four hours the task will be resent to another host and finish before deadline with high probability. Projects like Einstein@Home, which have tasks with higher durability, usually set deadline longer. What can be problematic is the end of an experiment if there is a requirement to complete tasks as soon as possible.

Returning to ComBoS, there were only two results sent after the deadline. The problem was partially with the metrics, as clients could cancel a task's execution, if it can't be finished in time, and such tasks weren't accounted for by the system. They started to be tracked but there were only 67 cancelled results, what is too few as well. It means that the model of availability was not approximated sufficiently.

Since tasks are short, the periods of availability for a host can be approximately extracted from graphs that depict task sent and received times (Fig. 7). The Y-axis represents time, red points indicate task sent times to clients and blue points - times the server received results back. It's possible that during gaps clients contribute to other projects, but if only one project is simulated, they're equivalent to periods of unavailability. Figure 7 shows different patterns of unavailability, and for more accurate simulations, they have to be modeled separately by creating several clusters for each pattern. However, right now there is no tool to derive parameters for the config from such graphs.

5.3 Metrics

In addition to timelines described in Subsect. 5.1, it was beneficial to measure inner work via metrics to understand the simulation better. For instance, the time required to compute a task was measured. During the process, a strange behavior was encountered: tasks with the same duration were executed for different lengths of time on the same host. It indicated another bug in ComBoS.

All tasks were executed within a separate actor [18] and had a data type Exec. When the simulator modeled period of unavailability, the actor was suspended

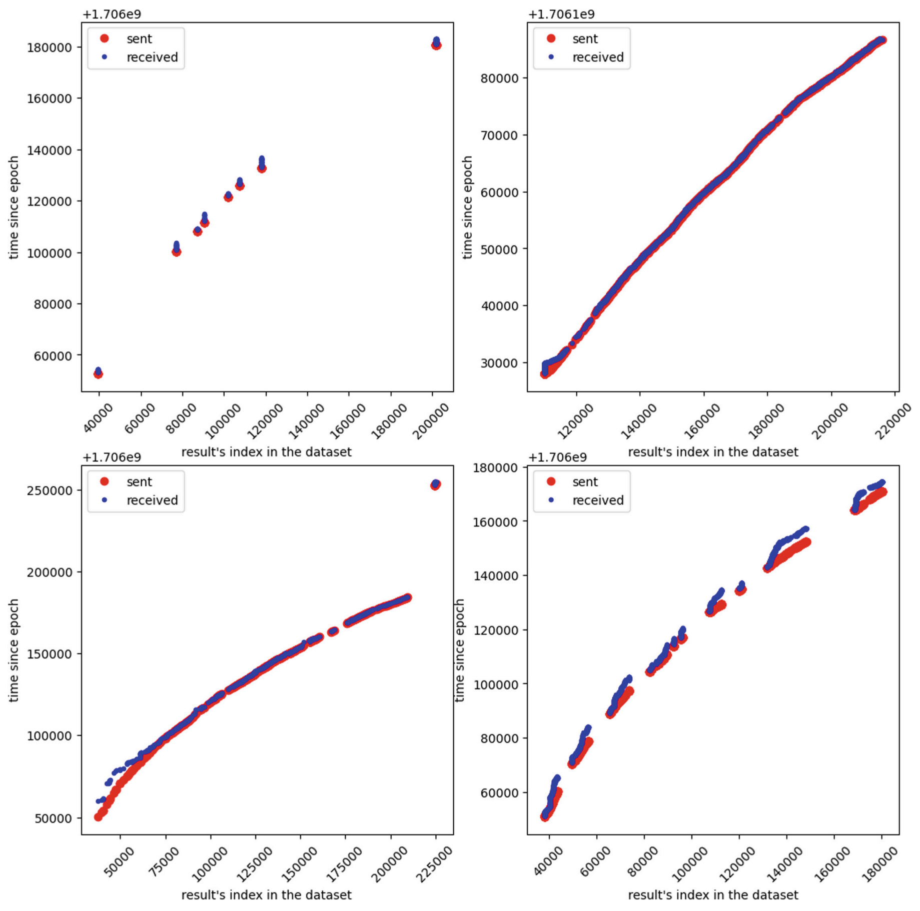

Fig. 7. Approximation of unavailability model via sent_time and receive_time

but not Exec-s. SimGrid solves a set of linear equations [19] to advance the time of the simulator until the next event. In this set the task, which wasn't suspended, "continued to be executed" even though the client was unavailable for computation.

Even after fixing this behavior in ComBoS, the simulator kept to act peculiarly. The tasks continued to progress after the second call of *suspend()*. In this case, the bug was found in SimGrid and it was reported.

To conclude, those metrics proved to be helpful, and they were kept in the code for the next experiments.

6 Verification of a Single Host Simulation

In the sections above, the simulator was verified via observing its inner work (Subsects. 5.1 and 5.3). Also, it is not trivial to fit the availability model for hosts (Subsect. 5.2). In this section, the work of a single host will be simulated.

First of all, several hosts have been chosen from dataset (Table 2).

Table 2. Hosts from RakeSearch for verification

ID	CPU number	CPU power in GigaFLOPs per second
A	4	3.575967
B	32	6.836439

The median execution time of tasks and their quantity were used for comparison with the simulation. Instead of modelling availability periods, it was calculated approximately how much time the host computed tasks for BOINC and configured a simulated client to work for this duration. The results are presented in Table 3.

Table 3. Metrics compared with the simulations

metric	A		B	
	dataset	simulation	dataset	simulation
50% of execution time	348.83	275	184.22	142
# completed	434	350	12304	746
GigaFLOPs	$575.47 \cdot 10^3$	$341.71 \cdot 10^3$	$15.20 \cdot 10^6$	$0.73 \cdot 10^6$

The execution times are close, but it isn't true for the number of tasks. Tasks can be executed simultaneously on several CPUs, so several polices were tried to simulate this for host B. When the host power was multiplied by the number of cores, 1840 completed results were obtained. When hosts were duplicated by the number of cores, the simulation generated 23718 results. The second option appeared to be better. The difference could be explained by the fact that host B might participate in multiple projects, so half of the time is spent on other projects. However, there were hosts that were dedicated solely to RakeSearch, yet the simulation also generated twice the amount of work compared to what they computed during the measured period. This led us to another assumption - hosts might not be fully utilized.

Hosts were selected from the database based on those for which CPU power could be obtained, and among them, those that worked all the time during the period described in Sect. 4 (according to graphs similar to Fig. 7). For these hosts, the total amount of work computed (TC_{real}) was calculated, as well as for how long they worked (τ), and the number of CPU cores they possess ($ncpus$). Using this data, actual CPU power (HP_{df}) was introduced using Eq. 1.

$$HP_{df} = \frac{TC_{real}}{\tau \cdot ncpus} \tag{1}$$

A coefficient $coef_{slow\_down}$ was also defined, which represents how much CPU power (HP_{db}) from the database exceeds the actual one (Eq. 2):

$$coef_{slow\_down} = \frac{HP_{db}}{HP_{df}} \tag{2}$$

The intuition behind this coefficient is to measure how much work could be performed if all CPU were busy all the time and if their powers were the same as stated in the database (HP_{db}). The actual power may be lower if the host works with interruption. For instance, there is a parameter in the user settings which defines the percentage of CPU time, and if the user's processes exceed this value, then the BOINC client will halt.

The configuration file was filled for the simulation, it was run and the completed work in FLOPs (TC_{sim}) was recorded. The coefficient in Eq. 3 also means how much work could be accomplished if the client acted as in the simulation.

$$coef_{flops} = \frac{TC_{sim}}{TC_{real}} \tag{3}$$

Results are provided in Table 4.

Values for $coef_{slow\_down}$ and $coef_{flops}$ are close, what means (4):

$$coef_{slow\_down} = \frac{HP_{db}}{HP_{df}} = \frac{HP_{db} \cdot ncpus \cdot \tau}{TC_{real}} \approx coef_{flops} = \frac{TC_{sim}}{TC_{real}}, \tag{4}$$

$$HP_{db} \cdot ncpus \cdot \tau \approx TC_{sim} \tag{5}$$

Equation 5 indicates that the simulation with one client produces the same amount of work as if all cores operated all the time and demonstrated performance equivalent to the CPU power specified in the configuration.

In conclusion, the simulation of a single host was verified. The correct work of the simulator with periods of availability and unavailability, as well as simulations involving multiple hosts, can be verified using timelines similar to those from Sect. 5.1.

In addition, based on the experiment, it has been seen that the clients aren't fully utilized, even if they are dedicated solely to the project, what might be caused by user settings. It can be investigated further.

What is more, the tasks in the simulation are of fixed size and they can be generated in a larger amount compared to the dataset. The amount of work in FLOPs is a more preferable metric for experiments in ComBoS.

Table 4. $coef_{slow\_down}$ and $coef_{flops}$ calculated for chosen hosts

host	$coef_{slow\_down}$	simulation time	#tasks in real	#tasks in simulation	$coef_{flops}$
1	1.243	20	2659	3797	1.138
2	1.694	18	8934	13298	1.497
3	2.084	37	1176	2426	1.909
4	2.108	38	2215	5574	1.922
5	2.252	38	1282	2605	2.083
6	2.267	37	2296	8578	2.057
7	2.373	38	2608	6040	2.158
8	2.435	38	4730	16436	2.230
9	2.461	37	1333	4588	2.257
10	2.541	38	2109	6105	2.306
11	2.726	38	1442	3856	2.514
12	3.338	38	2249	9707	3.030
13	4.620	37	1798	8743	4.196
14	6.924	37	1450	11904	6.247
15	8.016	37	1261	8214	7.178
16	27.094	38	1037	5511	24.776

7 Conclusion

BOINC is a widely and actively used platform for volunteer computation that contributes to science. The current paper provides an overview of the BOINC system and the tools which models it for research and development purposes. It also describes the experience with running one of the available simulators, ComBoS, problems that were encountered and how they were solved. The paper also includes the steps were made to validate the new version of the simulator.

Overall, the simulator has become more convenient to work with, many experiments were run to find and fix bugs. It also was validated by drawing helpful graphs, measuring metrics and comparing performance of hosts with the real data.

Our version of ComBoS is available by [21].

Acknowledgments. This work was funded by Russian Science Foundation (No 22-11-00317). We would also like to thank the RakeSearch team for providing the data.

References

1. Anderson, D.P.: BOINC: a platform for volunteer computing. J. Grid Comput. **18**(1), 99–122 (2020)
2. Anderson, D.P., Fedak, G.: The computational and storage potential of volunteer computing. In: Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2006), vol. 1, pp. 73–80. IEEE (2006)
3. Kondo, D., Javadi, B., Malecot, P., Cappello, F., Anderson, D.P.: Cost-benefit analysis of cloud computing versus desktop grids. In: 2009 IEEE International Symposium on Parallel & Distributed Processing, pp. 1–12. IEEE (2009)
4. McGregor, I.:. The relationship between simulation and emulation. In: Proceedings of the Winter Simulation Conference, vol. 2, pp. 1683–1688. IEEE (2002)
5. Smaoui Feki, M., Nguyen, V.H., Garbey, M.: Parallel genetic algorithm implementation for BOINC. In: Parallel Computing: From Multicores and GPU's to Petascale, pp. 212–219. IOS Press (2010)
6. Anderson, D.P., Korpela, E., Walton, R.: High-performance task distribution for volunteer computing. In: First International Conference on e-Science and Grid Computing (e-Science 2005), 8-p. IEEE (2005)
7. Casanova, H., Legrand, A., Quinson, M.: Simgrid: a generic framework for large-scale distributed experiments. In: Tenth International Conference on Computer Modeling and Simulation (uksim 2008), pp. 126–131. IEEE (2008)
8. Alonso-Monsalve, S., García-Carballeira, F., Calderón, A.: ComBos: a complete simulator of volunteer computing and desktop grids. Simul. Model. Pract. Theory **77**, 197–211 (2017)
9. 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, pp. 605–612 (2010)
10. Taufer, M., Kerstens, A., Estrada, T., Flores, D., Teller, P.J.: SimBA: a discrete event simulator for performance prediction of volunteer computing projects. In: 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS 2007), pp. 189–197. IEEE (2007)
11. Estrada, T., Taufer, M., Reed, K., Anderson, D.P.: EmBOINC: an emulator for performance analysis of BOINC projects. In: 2009 IEEE International Symposium on Parallel & Distributed Processing, pp. 1–8. IEEE (2009)
12. Block diagram of ComBoS architecture. https://drive.google.com/file/d/1AiNDxQ6wiof9eOykej56L1AG8mgznK_Z/view?usp=sharing
13. Debugging tool Valgrind. https://valgrind.org/
14. BOINC project RakeSearch. https://rake.boincfast.ru/rakesearch/about.php
15. Vatutin, E., et al.: Diagonalization and canonization of latin squares. In: Russian Supercomputing Days, pp. 48–61. Springer, Cham (2023)
16. Network examples in the SimGrid documentation. https://simgrid.org/doc/latest/Platform_examples.html#network-topology-examples
17. 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**(11), 1896–1903 (2011)
18. Description of the S4U interface in SimGrid. https://simgrid.org/doc/latest/app_s4u.html#api-s4u-actor
19. 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)

20. Kurochkin, I., Kondrashov, N.: Comparison of various algorithms for scheduling tasks in a desktop grid system using a ComBos simulator. In: High-Performance Computing Systems and Technologies in Scientific Research, Automation of Control and Production: 10th International Conference, HPCST 2020, Barnaul, Russia, 15–16 May 2020, Revised Selected Papers 10. Springer (2020)
21. The code for the author's version of ComBoS. https://github.com/Ksenia-C/combos/tree/master