






Using Mobile Devices in a Voluntary Distributed Computing Project to Solve Combinatorial Problems

Ilya Kurochkin^{1,2} , Andrey Dolgov², Maxim Manzyuk³ ,
and Eduard Vatutin⁴ 

¹ Institute for Information Transmission Problems of Russian Academy of Sciences, Moscow, Russia

² The National University of Science and Technology MISiS, Moscow, Russia

³ Internet Portal BOINC.Ru, Moscow, Russia

hoarfrost@rambler.ru

⁴ The Southwest State University, Kursk, Russia

evatutin@rambler.ru

Abstract. A large number of personal mobile devices and the presence of powerful computing processors on them allows one to use them as computing nodes. The idle computing power of mobile devices can be used in distributed computing systems. This paper discusses the use of personal mobile devices in the project of voluntary distributed computing. The features of computing on mobile devices are discussed. The results of computational experiments on a desktop grid system are presented. Setting up a BOINC-based desktop grid system for efficient use of mobile devices as computing nodes is discussed.

Keywords: Desktop grid · BOINC · Mobile device · Voluntary distributed computing project · Orthogonal diagonal Latin squares

1 Introduction

In recent years, the use of mobile devices has increased significantly worldwide. According to Digital Trends, the number of smartphone users in the world in 2020 reached 6.1 billion and it continues to increase. The current capabilities of these devices have also significantly increased. Thanks to the latest advances in low-power processors, mobile devices can perform resource-intensive operations, which allow ones to consider these devices as computing platforms.

Using the advantages of smartphones and tablets, over the past decade, some research projects have investigated the problem of including mobile devices in distributed computing systems. The inclusion of mobile devices in the network and computing infrastructure has led to the emergence of new categories of networks. One of these new categories is a mobile network, which has been defined as a network that includes at least one mobile device. Users can connect their mobile devices to the network (smartphones, tablets and etc.) mainly for two purposes: to gain access to network resources and/or to make the resources of their mobile devices available to

network users. A computing network consisting exclusively of mobile phones with the participation of millions of individual devices distributed around the world can reach the level of performance of a supercomputer.

The current work focuses on the use of mobile devices as providers of computing resources. The main idea of the work is to use idle computing resources of mobile devices for calculations, without compromising the use for their intended purpose. It is necessary to conduct various computational experiments to study the behavior of a grid system consisting of heterogeneous devices of different processor architectures and different operating systems, with subsequent adjustment of various project parameters. Correctly selected parameters of the distributed computing project can both increase the level of utilization of nodes in the grid system, and significantly reduce the calculation time of the entire experiment. As a result of the study, the parameters of a BOINC-based desktop grid are recommended, which effectively works with various types of processor architectures and OS of personal devices. This can be useful for large computational experiments, which last several months in a desktop grid.

2 BOINC Software

The use of idle resources of personal computers became popular in the late 1990s, when access to the global Internet became available to the average user [1]. The first public desktop grid systems appeared – projects of voluntary distributed computing: SETI@home [2], Folding@home, GIMPS. These first projects of voluntary distributed computing used the power of ordinary users' computers connected to the Internet to solve research problems that require large computing capacities [3].

The project of voluntary distributed computing is a deployed public grid system of personal devices that solves one or more computing tasks. A public grid system uses the computing resources of volunteers who provide the resources of their computing devices. The project can be deployed in the interests of one research group (SETI@home), several research groups using a common set of applications (Climateprediction.net, nanoHUB@home) or several independent groups of scientists (World Community Grid, BOINC@TACC).

There are platforms for organizing distributed computing, for example, BOINC, HTCondor, Askalon, Grid engine, Globus toolkit. One of the most common is BOINC. SETI@home, LHC@home, World Community Grid and several dozen other distributed computing projects use the BOINC platform.

BOINC consists of a server and a client part. The BOINC server part consists of many separate services (daemons) that share a common MySQL database. These services include a work generator, a scheduler, a feeder, a transition, a validator, an assimilator, a file deletion tool, and a database purger.

The client part of BOINC consists of three programs: the main client manages the execution of tasks and file transfer, the graphical interface (BOINC manager) allows volunteers to monitor and track calculations, an additional screensaver shows the graphics of the application for performing tasks in standby mode.

Third-party developers have implemented alternative graphical interfaces; for example, BoincTasks is a graphical interface for Windows that can manage multiple

clients. For hosts without a display, BOINC provides a command-line program that provides the same functions as the manager.

The BOINC platform is used by many voluntary distributed computing projects. Some of them are based in universities and research laboratories, others are managed by private groups or individuals. As an example of public projects [4] that support computing on Android devices, we can cite: Asteroids@home, Einstein@home, LHC@home, Moo! Wrapper, Rosetta@home, Universe@Home, World Community Grid.

3 Integration of Mobile Devices into the Desktop Grid

Initially, mobile devices were integrated with networks as a user interface for managing and monitoring network resources [5]. Since the earliest mobile devices with wireless network capabilities had very limited resources, they could not share their resources with the network, and even if they could, their contribution would not be significant enough to justify the effort required to integrate them into the network. In addition, wireless networks were not as advanced, reliable and fast as they are today. However, mobile devices have been successfully integrated to effectively provide access to the grid network via a wireless network for management and monitoring [6, 7]. For this kind of integration, connectivity and energy efficiency are not important problems, because if the mobile device is disconnected from the network, there will be no reduction in resources. On the other hand, using mobile devices as a grid system user interface allows users to extract grid system resources from any place where there is a wireless connection. In addition, access to network resources from a mobile device allows you to perform computing tasks that are otherwise impossible to perform on a mobile device due to its lack of resources.

Over time, the capabilities of mobile devices have been improved at exponential rates. In addition, wireless networks have become faster, more reliable and more affordable. For example, 4G allows you to stay connected to the Internet most of the time, moving over large areas, such as cities or highways. In addition, many public places, such as airports, hotels and restaurants, have their own Wi-Fi networks, which are cheaper and faster than 4G networks. These facts allow you to connect mobile devices to the grid system. Even more, since modern mobile devices have a large number of available resources, they can share resources such as a processor, memory, video processor or storage space [7]. Public mobile grids (MoGrid, MORE, Mobile OGS.NET and other) began to appear [7], which used mobile devices as full-fledged computing nodes [8].

However, mobile devices have not only connection restrictions, but also the limited power of their batteries. Due to the limited power supply, mobile devices are less reliable in the grid network than stationary ones. In addition, the traditional network idea that a particular device will always be connected from the same place is no longer true. As a result, technological solutions that work for stationary devices [9] may not work for mobile devices [10–12]. Summing up, we can say that the main problems of integrating mobile devices with the grid are [13]:

- power consumption: each operation performed by the processor or each byte transmitted requires the consumption of a device battery, the charge of which remains limited;
- availability: since the devices are connected via wireless networks and due to their mobile nature, it is very likely that the time available on the grid system will not be constant.

These features can have a negative impact mainly on the timing of the work sent, and in the best case they will not be completed. If the device's battery is low or the user disconnects (voluntarily or unintentionally), the tasks are not completed. Therefore, it is important that mobile grids take into account these limitations and disadvantages and act in such a way as to ensure the reliability of the completion time.

That is why BOINC was chosen as the platform for the deployment of the mobile grid. The BOINC platform meets the following requirements:

- the ability to perform tasks written in C++;
- performing tasks on mobile devices;
- the availability of the platform's source code for making changes and the quality of the platform's technical documentation.

However, BOINC requires the expansion of certain functionality specific to the mobile grid by fine-tuning the server part, as well as the implementation of a computing application for Android OS. The choice of Android platform at this stage of the study is due to a large percentage of devices with such a platform and its openness (GNU GPL 2.0). Mobile devices running on iOS can also be integrated into a grid system, which is confirmed by ongoing researches [14, 15].

4 Computational Application

4.1 Algorithm

Combinatorial problems related to the research of DLS properties [16] have already been solved in several grid systems: SAT@home [17], RakeSearch, Gerasim@home [16].

The algorithm of the computational application [16, 18] works according to a general scheme, which can be represented as:

- generating a random diagonal Latin squares (DLS) [19];
- the selected numerical characteristic is calculated for it;
- the actions are repeated in a cycle, the extreme (minimum and maximum) values of the selected numerical characteristic and the corresponding Latin squares are remembered.

The first M elements of the matrix, together with the main and side diagonals, are filled in recursively. Next, the DLS is filled in according to the principle of the minimum of possibilities $|S| \rightarrow \min$, i.e. the cell a_{ij} is selected for filling, in which you can put a minimum of values. This principle makes it possible to reduce the arity of nodes in the combinatorial search tree, thereby significantly increasing its pace. Each next

i,j -th element is filled with a value from the allowed set of values S_{ij} [20], which can be defined as:

$$S_{ij} = U \setminus \bigcup_{k=1}^N \{a_{ik}\} \setminus \bigcup_{k=1}^N \{a_{kj}\} \setminus \bigcup_{k=1}^N \{a_{kk}\} \setminus \bigcup_{k=1}^N \{a_{k,N-k}\},$$

where $U = \{0, 1, 2, \dots, N - 1\}$ – the set of available values of the elements of the square, determined by its order N , a_{ij} is the value of the i,j -th element of the square.

The intermediate set of used values will be stored together with the generated DLS and changed simultaneously with the change of its elements.

The allowed set of S_{ij} values at each new iteration is estimated using the selected WRS heuristics:

$$f_{WRS}(s_{ij}^l) = g_{ij}^{(s_{ij}^l)} \cdot (1 + 2d(r_k - 0, 5)) \rightarrow \max, l = \overline{1, M(S_{ij})}$$

Where function $g_{ij}^{(s_{ij}^l)}$ calculates the weight of possible values, and the final element is selected according to the maximum value of the function $f_{WRS}(s_{ij}^l)$ with some pseudo-randomness r_k .

The resulting value is entered both in the DLS matrix and in the arrays of used values. If it is impossible to select an element at the i,j -th step, i.e. $|S_{ij}| = 0$ the algorithm performs a combinatorial return, moving to the previous step based on the path traveled, where the elements are re-evaluated taking into account the values used. The process of generating all new DLS is considered complete when the entire possible subset of DLS has been generated from the original incomplete DLS, i.e. the first cell of the path has no valid elements to choose from. The generation stops, the algorithm proceeds to record the results.

Each new received DLS is sent to estimate the number of partial and full cycles. The algorithm selects all possible pairs of numbers from the first row and counts the number of formed cycles in the composition of this DLS, i.e. it alternately finds the second number from the selected pair in a row or column. An example of one of the found cycles can be seen in Fig. 1.

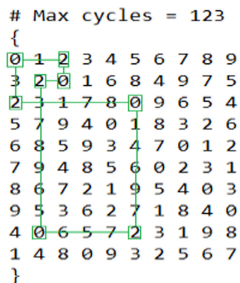


Fig. 1. One of the circles for pair 0–2

This operation is performed for all possible pairs of the set $U = \{0, 1, 2, \dots, N - 1\}$. The minimum and maximum values found are stored in variables for subsequent entries in the results.

4.2 Development

The computational application of the project for calculating the characteristics of the DLS is developed in the C++ language. This programming language was chosen for several reasons:

- BOINC mainly supports the C++ language, since the BOINC API is a set of C++ functions;
- the implemented algorithm implies complex and resource-intensive mathematical calculations;
- it is necessary that the application has versions for several different platforms (Windows, Linux, Android);
- it is necessary to minimize conflict situations when building the application, and ensure ease of debugging the source code.

Thus, the computing application will run on all major operating systems supported by BOINC. This, in turn, allows you to increase the number of potential computing nodes involved in the project.

5 Setup Project Parameters

The main configuration of the BOINC project is controlled by a file named `cofig.xml`. This file is created when executing the `make project` script, with the ability to edit and configure various parameters for the project tasks.

The option of one result per host (*one_result_per_host_per_wu*) sends no more than one result of this working unit to this host. This is useful if most of the hosts of a certain class belong to a single user and you need to reduce the impact of faulty hosts on validation.

The limit parameter for sending tasks to the user (*one_result_per_user_per_wu*) allows you to send no more than one instance of a given task to the current user. This increases the efficiency of replication-based validation, because it becomes more difficult to get all the instances of a given task to a single user.

The maximum number of working units (*max_wus_in_progress*, N) (*max_wus_in_progress_gpu*, M) are parameters that limit the number of tasks performed on this host and, thus, limit the average task execution time. The BOINC client reports the resources used by the running jobs; in this case, the maximum number of running jobs on the CPU is $N * NCPUS$.

The parameter (*max_wus_to_send*, L) responsible for the maximum number of tasks returned to the scheduler request, which is calculated according to $L * (NCPUS + GM * NGPUS)$.

The node's confidence coefficient (*daily_result_quota*, *MRD*). Each host has an *MRD* field in the interval $[1 \dots \text{daily_result_quota}]$. Initially, the coefficient is equal to the value of *daily_result_quota* and is adjusted when the host sends good or bad results. The maximum number of tasks sent to this host for a 24-h period is $MRD * (NCPUS + GM * NGPUS)$.

It should be used in order to limit the impact of faulty hosts.

The configuration of work units (workunits) is carried out at the stage of their creation by the job generator (work generator). The characteristics of the working unit are set as properties of the `DB_WORKUNIT` class, with the subsequent transfer of the object to the task creation function in the `create_work` database using the BOINC API functionality.

The minimum quorum size (*min_quorum*) is designed so that when the number of successful results of the set value is reached, the validator runs a check for matching the values of the working unit. If the strict majority agrees, this result is considered correct and is entered into the project database.

The replication parameter (*target_nresults*) means the number of work units created at the job generation stage. The set value must not be lower than the value of the minimum quorum size.

The *delay_bound* option is the upper limit of the time (in seconds) between sending the result to the client and receiving the response. The scheduler will not return a result if the estimated completion time exceeds this value. If the client does not respond within this interval, the server “refuses” the result and generates a new result that will be assigned to another client.

The complexity of the working unit (*rsc_fpops_est*) is an estimate of the number of floating-point operations required to complete the task. It is used to estimate how long a task will take on a given host.

The maximum allowable complexity of the working unit (*rsc_fpops_bound*). The upper limit of the number of floating-point operations required to complete the task. If this limit is exceeded, the task will be aborted.

The computational complexity parameters (*rsc_fpops_est*, *rsc_fpops_bound*) are expressed in terms of the number of floating-point operations. For example, suppose that the task takes 1 h to complete on a machine with a reference performance of 1 GFLOPS; then the “size” of the *j*-th working unit is $3.6 * 10^{12}$ FLOP.

To get estimates of the size of the working units averaged over the last tasks, you can use the link to the statistics of the number of flops in the administrative web interface of the project.

Table 1 shows the values of the configurable parameters during the four experiments.

Table 1. Server parameter values in various experiments.

Server parameter	Experiment 1	Experiment 2	Experiment 3	Experiment 4
one_result_per_host_per_wu	FALSE	FALSE	TRUE	TRUE
one_result_per_user_per_wu	FALSE	FALSE	TRUE	TRUE
max_wus_in_progress	1	1	10	10
max_wus_in_progress_gpu	1	1	1	1
gpu_multiplier	default	default	0	0
max_wus_to_send	1	1	50	50
daily_result_quota	1	1	8	8
min_quorum	3	3	2	2
target_nresults	3	3	3	2
delay_bound	172800	259200	432000	432000
rsc_fposts_est	1.00E + 15	1.00E + 14	1.00E + 13	1.00E + 13
rsc_fposts_bound	1.00E + 16	1.00E + 16	1.00E + 16	1.00E + 16

Based on the results of the experiments, the average time spent by the connected devices on problems of the same order of complexity was calculated.

The average time for working units of complexity 10^{15} FLOP was 9 h. For the complexity of 10^{14} FLOP, the average time spent was 3.5 h. For 10^{13} FLOP, the average time decreased to 0.9 h.

6 Results

To assess the state of the deployed desktop grid, as well as to study the operation of mobile devices and tablets as computing nodes together with personal computers and laptops, a number of computational experiments were launched using the implemented application for generating and calculating the characteristics of DLS. During the experiments, the parameters of the server part of the project were configured.

For the tests carried out, a set of tasks was defined in the amount of 1000 input text files, on the basis of which the working units of the project were generated in an amount equal to the replication parameter (target_nresults).

At the initial stage, the computing application was tested for compatibility with various OS and node processor architectures, during which all new hosts were gradually connected. In the end, the number of devices that took part in the calculation of tasks was 14 pieces. Of these, 1 desktop computer (Linux $\times 64$), 2 laptops (Windows $\times 64$), 10 mobile devices (Android arm, arm64), as well as 1 tablet (Android arm).

The first experiment (experiment 1) was conducted with the default parameters of the BOINC server. One working unit was calculated at each node. Based on the estimates obtained at the first stage, the computational complexity of the problem (rsc_fposts_est) averaged 1015 FLOP. The average number of results obtained for the presented period was 22 units. The maximum time to complete one task was 27 h. The average time spent by the device on the task was about 9 h. Summing up the results of the first experiment, it was concluded that the complexity of the initial task is too great

not only for mobile devices, but also for PCs. It was decided to reduce the computational complexity of the tasks and reduce the number of combinations specified in the input files.

In the second experiment (experiment 2), the number of tasks sent to the nodes did not change, however, the complexity of the task was reduced by an order of magnitude and amounted to 1014 FLOP. This approach provided an average execution time of one working unit of 3.5 h and raised the average number of results obtained to 49 pieces. According to the results of the second experiment, it can be concluded that the time required for the node to calculate the working unit has decreased at least three times.

An acceptable rate of the obtained results was found in the course of experiment 3. The computational complexity of the problem was 1013 FLOP, which led to 0.9 h of average time spent. This allowed the server to process an average of 721 results per day.

The diagrams (Fig. 2) show the distribution of the average daily number of completed tasks of some computing nodes during the third experiment.

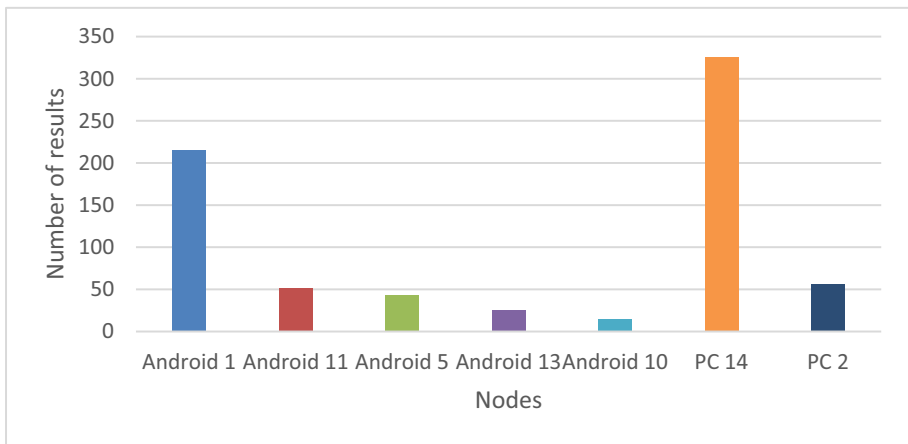


Fig. 2. Experiment 3, the number of results obtained on average per day.

The analysis of the dynamics of the results obtained by the server over a 48-h period of time is carried out. The beginning of measurements on 27.04.2021 21:03 and 30.05.2021 22:31, the end of 29.04.2021 21:05 and 01.06.2021 22:43 for experiments 2 and 3, respectively. As can be seen from the graphs in Fig. 3, in the intervals between 01:00 and 03:00, the server does not receive any results, which is explained by the unavailability of the server at this time due to a reboot. This situation should be taken into account in the process of computational experiments, preloading the nodes with additional tasks in order to prevent idle hosts waiting for new working units from the server. From the graphs presented, it can be seen that the number of results increases after 03:00 at night, since each node tends to send calculated tasks when the server starts responding to client requests. In Fig. 3, in addition to the dynamics for all computing nodes, the dynamics of the results obtained for 48 h is shown only for Android devices.

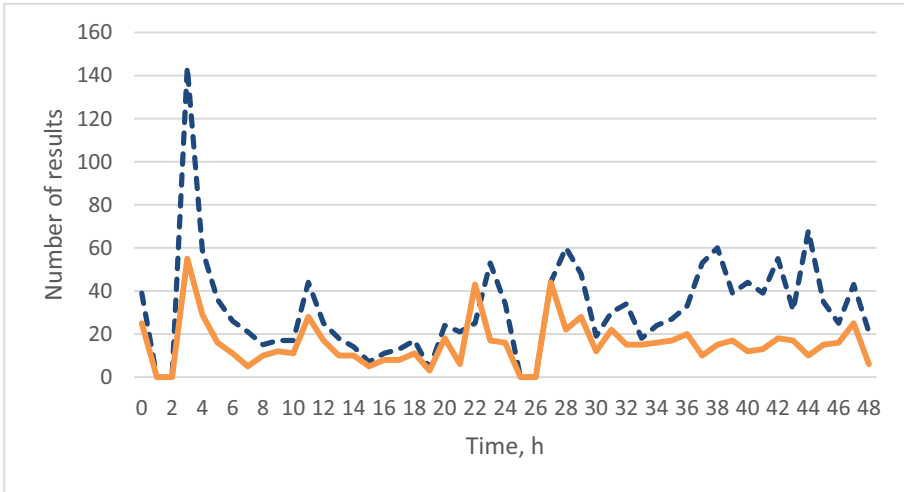


Fig. 3. Experiment 3, the dynamics of the results for all nodes (dotted line) and for mobile devices only (solid line)

Adjusting the total number of sent and the maximum possible number of executed work units on the CPU allows you to keep the node in constant operation, without wasting time on simple: requests to the server, receiving a new small portion of tasks, no calculations when the server is unavailable. Figure 4 shows a diagram of the percentage of node utilization on average per day under the conditions of the default settings (experiment 2) and the selected parameters (experiment 3).

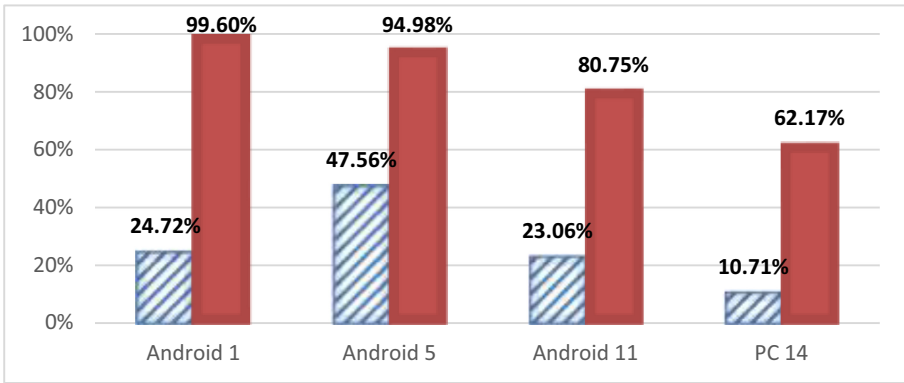


Fig. 4. Experiment 2 and 3, average percentage of device utilization per day

The parameters `max_wus_in_progress`, `max_wus_in_progress_gpu`, `max_wus_to_send` were equal to 1, which allowed limiting the number of tasks on the nodes. This approach, in turn, is not effective, since the volume of tasks on the hosts is very limited,

which leads to idle computing power after the completion of the received working units, time is spent on frequent requests of the device to the server, and in the case of a reboot or problems on a remote server, computing work is completely stopped until the connection is restored.

Subsequently, the parameters *max_wus_in_progress* and *max_wus_to_send* were changed to the values 10 and 50, respectively, which allowed to increase the number of stored working units to $10 * N_{CPU}$, and the number of tasks sent to 50.

Based on the estimates obtained in the diagram (Fig. 3), it can be noted that the selected parameters led to almost constant (more than 90%) loading of Android 1 and Android 5 devices, which were connected to the network all the time, and were used only for calculations. The percentage of PC 14 downloads increased almost 6 times when using the computer for other purposes.

Delay_bound is an important parameter of the project, since overdue tasks are recreated by the scheduler on the server, thereby losing intermediate results and increasing the total time of the experiment. Given the specifics of using Android devices in grid systems, it is necessary to set this parameter with a margin of several days. Figure 5 shows the ratio of overdue results to successful ones (experiments 1, 2 and 3).

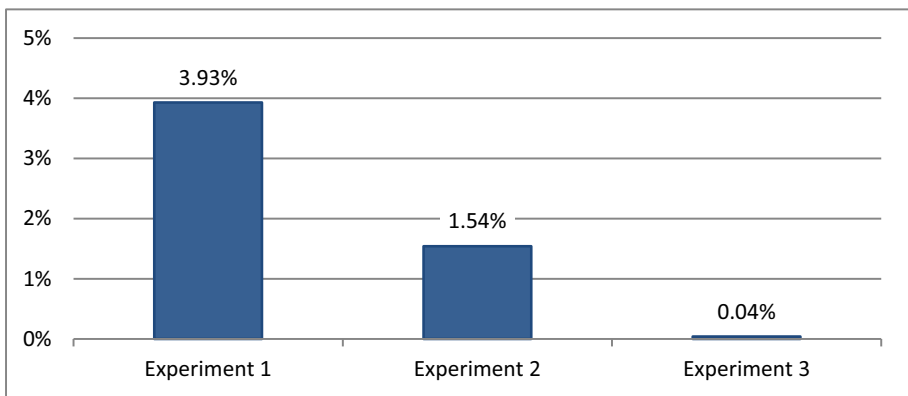


Fig. 5. Percentage of overdue tasks

Initially, the *delay_bound* parameter was 48 h, which led to the loss of almost 4% of the results and was a strict restriction, provided that the average computational complexity of the working unit was 1015 FLOP, and the average calculation time reached 9 h. At these values, the device should work only on one task, without disconnecting from the power supply.

In experiment 2, *delay_bound* increased to 72 h, while the complexity of the working unit decreased by an order of magnitude. In total, the percentage of lost tasks was 1.54%.

By the experiment 3, there were no working units that did not have time to be completed, with the exception of 1 task that was intentionally stopped. The *delay_bound* parameter, set to 5 days, is well suited for conducting further experiments.

Given the mechanics of task replication in the BOINC project, it is important to remember that it is unacceptable for one device to receive all copies of one task, because in case of a malfunction or incorrect operation of the application, situations arise when the same, but incorrect result of all replicas from the device is returned to the server, after which it successfully passes validation and is entered into the database. All these nuances were taken into account in the process of 3 experiments. In order to minimize the number of incorrect results, it is necessary to prohibit the issuance of replicas of the same task to one user or host by activating the options *one_result_per_host_per_wu*, *one_result_per_user_per_wu*. The *daily_result_quota* parameter was also adjusted and reduced. Sometimes there are hosts in the project that “make mistakes” in all the working units that are sent to them. Often these machines are incorrectly configured or have some other operating system or a problem with the installation of BOINC that needs to be fixed. To reduce the impact of these devices on the project, we use *daily_result_quota* so that these hosts do not destroy hundreds or thousands of tasks per day.

The *daily_result_quota* parameter is usually 8 tasks per processor core. The host can request and receive up to this number of tasks per day. Each time the host returns an unsuccessful or overdue result (does not return a result by the deadline), its daily quota of results is reduced by one. Each time the host returns a successful result, its daily quota of results is doubled.

Provided that the node returns at least some successful results, its daily quota of results should remain around 8. Hosts with a daily quota of results of 1 should be checked: there may be something wrong with them.

7 Conclusion

A test grid system was deployed on the BOINC platform. Mobile devices, tablets, as well as personal computers and laptops can act as computing nodes. The current project covers processor architectures: $\times 86$, aarch64, arm, $\times 86-64$. The computing application is available for several operating systems, including Android. Several computational experiments were conducted with various parameters on a test grid system from personal devices in order to configure the project parameters. The parameters of the voluntary distributed computing project were configured, the configuration made it possible to reduce the time of conducting computational experiments, increase the percentage of device utilization and reduce the percentage of overdue tasks.

Acknowledgements. This work was by RFBR according to the research projects № 18-29-03264 and № 19-07-00802.

References

1. Foster, I., Kesselman, C.: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Burlington (1998)
2. Anderson, D.P.: BOINC: a platform for volunteer computing. *J. Grid Comput.* **18**(1), 99–122 (2019). <https://doi.org/10.1007/s10723-019-09497-9>
3. BOINC projects: List BOINC projects, 20 March 2021. <https://boinc.berkeley.edu/projects.php>
4. Zeng, W., Zhao, Y., Song, W., Wang, W.: Wireless grid and storage system design. In: 2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing (2008)
5. Rings, T., et al.: Grid and cloud computing: opportunities for integration with the next generation network. *J. Grid Comput.* **7**, 375–393 (2009)
6. Black, M., Edgar, W.: Exploring mobile devices as Grid resources: using an $\times 86$ virtual machine to run BOINC on an iPhone. In: 2009 10th IEEE/ACM International Conference on Grid Computing, pp. 9–16. IEEE (2009)
7. Kumar, M.P., Bhat, R.R., Alavandar, S.R., Ananthanarayana, V.S.: Distributed public computing and storage using mobile devices. In: 2018 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), pp. 82–87. IEEE (2018)
8. Curiel, M., Calle, D.F., Santamaría, A.S., Suarez, D.F., Flórez, L.: Parallel processing of images in mobile devices using BOINC. *Open Eng.* **8**(1), 87–101 (2018)
9. Mazalov, V.V., Nikitina, N.N., Ivashko, E.E.: Task scheduling in a desktop grid to minimize the server load. In: Malyshekin, V. (ed.) PaCT 2015. LNCS, vol. 9251, pp. 273–278. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21909-7_27
10. Curiel, M.J., Payares, A.G.: Scheduling strategies for mobile devices in BOINC. *Revista Electrónica Argentina-Brasil de Tecnologías da Informação e da Comunicação* **1**(13) (2021). <https://doi.org/10.5281/zenodo.4445243>
11. Wang, S.-D., Hsu, I.-T., Huang, Z.-Y.: Dynamic scheduling methods for computational grid environments. In: *International Conference on Parallel and Distributed Systems*, vol. 1 (2005)
12. Freund, R.F., et al.: Scheduling resources in multiuser, heterogeneous. In: Presented at the 7th IEEE Heterogeneous Computing Workshop (1998)
13. Du, L., Yu, Z.: Scheduling algorithm with respect to resource intermittence in mobile grid. In: 2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM) (2010)
14. Black, M., Edgar, W.: Exploring mobile devices as grid resources: using an $\times 86$ virtual machine to run BOINC on an iPhone. In: 2009 10th IEEE/ACM International Conference on Grid Computing, pp. 9–16. IEEE (2009)
15. Shah, S.C.: Recent advances in mobile grid and cloud computing. *Intell. Autom. Soft Comput.* **2017**, 1–13 (2017)
16. Vatutin, E., Zaikin, O., Kochemazov, S., Valyaev, S.: Using volunteer computing to study some features of diagonal Latin squares. *Open Eng.* **7**(1), 453–460 (2017)
17. Zaikin, O., Kochemazov, S.: The search for systems of diagonal Latin squares using the SAT@ home project. *Int. J. Open Inf. Technol.* **3**(11), 4–9 (2015)
18. Colbourn, C.J., Dinitz, J.H.: *Handbook of Combinatorial Designs*, 2nd edn. Chapman & Hall/CRC, Boca Raton (2006)
19. Shao, J., Wei, W.: A formula for the number of latin squares. *Discrete Math. Semant. Scholar* **110**(1–3), 293–296 (2021). [https://doi.org/10.1016/0012-365X\(92\)90722-R](https://doi.org/10.1016/0012-365X(92)90722-R)
20. Vatutin, E., Belyshev, A., Nikitina, N., Manzuk, M.: Evaluation of efficiency of using simple transformations when searching for orthogonal diagonal Latin squares of order 10. In: Jordan, V., Filimonov, N., Tarasov, I., Faerman, V. (eds.) HPCST 2020. CCIS, vol. 1304, pp. 127–146. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-66895-2_9