BOINC-based Branch-and-Bound

Andrei Ignatov✉[1] and Mikhail Posypkin[2]

[1] Moscow State University, Moscow, Russia,
rayignatov@outlook.com
[2] Federal Research Center Computer Science and Control of Russian Academy of Sciences, Moscow, Russia,
mposypkin@gmail.com

Abstract. The paper proposes an implementation of the Branch-and-Bound method for an enterprise grid based on the BOINC infrastructure. The load distribution strategy and the overall structure of the developed system are described with special attention payed to some specific issues such as incumbent updating and load distribution. The implemented system was experimentally tested on a moderate size enterprise grid. The achieved results demonstrate an adequate efficiency of the proposed approach.

Keywords: BOINC · Branch and Bound · Distributed Computing

1 Introduction

Desktop grids is a rapidly growing platform for distributed computing. The most popular software for desktop grids is BOINC (Berkeley Open Infrastructure for Network Computing) [7]. In a nutshell, BOINC is a system that can harness the unused computing power of desktop machines for processing resource demanding applications. Potentially, such grids can collect a tremendous amount of resources. However, efficient usage of this power can be a rather complicated task due to heterogeneity and irregularity of desktop grids.

Traditional approaches developed for standard HPC platforms are based on some assumptions that are not valid in desktop grids. In particular, such approaches assume low-latency fast communications where each parallel processor can serve both as a sender or a receiver. Clearly such approaches are not suitable for BOINC grids where communications are possible only between the server and a client node and are initiated by a client. BOINC communications have high latency as they involve several services and a file system.

Thus, we can conclude that developing new efficient distributed algorithms suitable for desktop grids is an important research direction. In this paper, we consider the implementation of tree-structured computations in desktop grids. Such computational schemes are remarkably popular in global optimization, in particular, with Branch-and-Bound family of methods. The paper is organized as follows. Section 2 discusses existing approaches for solving global optimization

problems in distributed computing environment. Section 3 outlines the structural, algorithmic and implementation details of the proposed approach. Experimental results are presented in Section 4.

2 Related work

Branch-and-bound is a universal and well-known technique for solving optimization problems. It interprets the input problem as the root of a search tree. Then, two basic operations are recursively executed: branching the problem (node) into several smaller (hopefully easier) problems, and bounding (pruning) the tree node. At any point during the search tree traversal, all subproblems can be processed independently. The only shared resource is the incumbent. Hence, processing the search tree in a distributed way is considered rather natural and has been studied for decades. Since the size and the structure of the branch-and-bound tree are unknown in advance, the even distribution of computations among processors is a challenging task. Load balancing has been comprehensively studied for tightly-coupled multiprocessors. Most efficient schemes use intensive communication among processors to approach uniform distribution. Unfortunately, this approach is not suitable for volunteer desktop grids where direct communications among computing nodes are normally not allowed.

The solution for distributed systems consisting of computational resources connected via wide-area networks (WAN) was proposed in [5, 6].

In [20] authors describe the AMPLX toolkit that enables modifying any AMPL script to solve problems by a pool of distributed solvers. The toolkit is based on Everest platform [21] that is used to deploy optimization tools such as web services and run these tools across distributed resources.

The BNB-Grid framework proposed in [4], [11] was aimed at running exact (based on Branch-and-Bound) and heuristic search strategies on grids consisting of heterogeneous computing resources. BNB-Grid uses different communication packages on different levels: on the top level, it uses ICE middleware coupled with TCP/IP sockets, and within a single computing element, either MPI or POSIX Thread libraries are used. BNB-Grid was used to solve several challenging problems from various fields, see e.g. [19].

The approach closest to ours was proposed in [8], where authors described a grid enabled implementation of the branch-and-bound method for computational grids based on the Condor [16] middleware. The suggested approach uses a centralized load balancing strategy: the master keeps a queue of sub-problems and periodically sends them to free-working nodes (slaves). When a sub-problem is sent to the slave, it is either completely solved or the resolution process is stopped after a given number of seconds while unprocessed subproblems are sent back to the master. Authors reported successful results for several hard quadratic assignment instances.

In comparison to that research, our major contribution is the implementation of a branch-and-bound optimization problem solver for a conceptually different platform. In addition, in our work, we suggest an effective algorithm of filtering

tasks on the server side, which leads to a significant reduction in the computing time.

There have been several attempts to use BOINC desktop grids to run Branch-and-Bound method. Some preliminary experiments on solving knapsack problems on a small test BOINC system were presented in [22]. In [12] authors consider an implementation of a back-track search in volunteer computing paradigm. They present a simple yet efficient parallelization scheme and describe its application to some combinatorial problems: Harmonious Tree and Odd Weird Search, both carried out at the volunteer computing project yoyo@home. The paper also presents a simplified mathematical analysis of the proposed algorithm.

3 Outline of the approach

Any BOINC application consists of the server and the client parts. The server part running on the BOINC server is responsible for generating tasks and aggregating their results obtained by client applications running on computational nodes. In our case, the client part is a C++ application [3] that solves a global box-constrained optimization problem with the Branch-and-Bound method. Interval analysis [14] is used to compute bounds on the objective. We used our own implementation of interval analysis [2, 18] that allows to compute bounds based on the C++ representation of an algebraic expression.

The general outline of our system is presented in Fig. 1. Like all BOINC applications, it contains client and server parts. Server maintains creation, sending, cancellation of tasks, and receiving and processing results. There is also a number of client machines that receive tasks, perform calculations, and send the achieved results back to the server.

The client reads the "state" from input file "in.txt" and writes the produced output to "out.txt". Both files are encoded in JSON format. The input files contain the record (incumbent value) found so far, a set of sub-problems to process and the maximal step limit. The step limit helps to bound the running time to avoid long-running tasks. To make running times of tasks uniform, it is suggested to aggregate several subproblems in one task. Omitting this suggestion may lead to dramatic differences in time elapsed for tasks [15].

Similarly, an output file contains the number of iterations that were performed, the record value, and a set of unprocessed subproblems in case if the upper bound for iterations was exceeded. The workflow of this application is as follows:

1. The pool of subproblems is initialized according to the input file.
2. The first subproblem D is extracted from the pool. Having it, the function value $f(c_D)$ for the centre point c_D of the feasible set X_D of the subproblem D is calculated along with the bound of the function's values in X_D.
3. The discovered value $f(c_D)$ is compared to the current record value f_r. The latter is possibly updated and the new incumbent solution is stored.
4. The optimality test is performed: if the lower bound of the current domain is less than the record function value with the prescribed accuracy ϵ, the

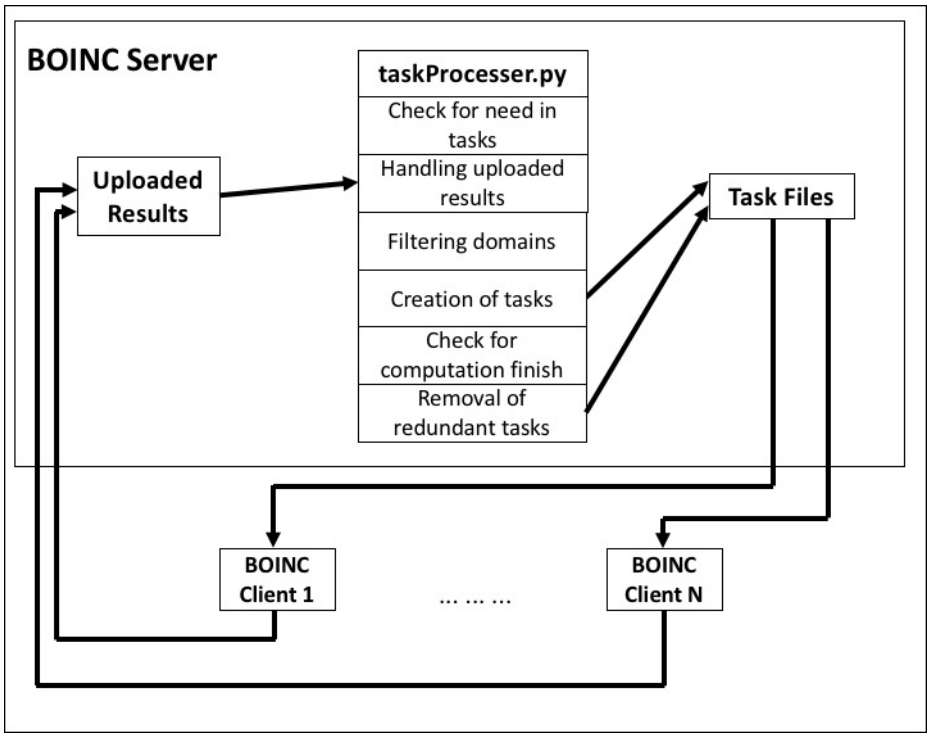

Fig. 1. Basic Workflow of the Developed System

feasible set is divided in two across the greatest dimension and the generated subproblems are added to the pool.
5. Steps 2-4 are repeated until the pool is empty or the number of performed steps exceeds the limit.
6. After the computation process is over, the output JSON line is formed and written to "out.txt" file.

This application was compiled for Linux, Windows, and OS X 32- and 64-bit systems.

To run an application in a BOINC project, it is required either to implement the API for server-client communication or to use special wrappers [17], [1] available in BOINC public resources. A wrapper is a special program for a particular operating system and architecture that runs the provided functional applications, 'feeds' the tasks to them, and manages all types of communication with the server. For our project, we decided to use the BOINC wrapper [1] for all platforms, which resulted in lesser amounts of code along with the convenience of tracking possible issues as wrappers provide detailed information regarding the running application.

In the server part, there is a permanently running Python script [3] that finds all uploaded files and processes their contents. Information from each file is extracted and analyzed: the number of performed iterations is added to the global counter, record data is updated, and received subproblems are added to the global pool. After processing, the files are deleted as they are no longer needed, and the formed pool of subproblems is merged with the previous ones saved in the special file containing the pool of subproblems. Subproblems should be also examined at the task formation step in order to avoid sending clearly redundant ones.

Another job performed by the server part is the creation of new workunits. One workunit aggregates a fixed number of subproblems S. It is well known that in some cases the Branch and Bound method may not converge in a short time, so it is worth restricting the total number of iterations for the whole problem with a value denoted below as $N_{totalmax}$. Besides subproblems and the record value, each workunit stores the maximum number of iterations. This number if computed as follows:

$$N = \min\left(\left\lfloor \frac{N_{totalmax} - N_{performed}}{N_{pool}} \right\rfloor, N_{max}\right), \tag{1}$$

where $N_{performed}$ is the total number of iterations performed so far and N_{max} is the iteration threshold. Initially, it was suggested to generate workunits once per a fixed amount of time, but then it turned out that decrease in this delay time led to creation of a great number of redundant tasks. On the other hand, increasing that amount of time caused a lack of workunits and consequently long idle periods of client PCs. In general, we came to the following conclusion: the later a subproblem is packaged to a workunit, the higher chances that it will discarded by the optimality test are. Therefore, it was decided to implement the 'on-demand' model where new workunits are created when they are really

needed. The script checks the number of unsent tasks, and if there are more than T of them, it is concluded that new tasks are not necessary and the script stops generating them.

Regardless of the application's 'on-demand' policy, it is still possible that some submitted or running tasks could be discarded according to the fresh record value. To handle this, all unsent or currently running tasks are examined. It leads to the following rule: if for a task t

$$\min_{s \in t} \ lb_s \ \geq f_r - \varepsilon, \tag{2}$$

then the task should be cancelled. Here, s is a subproblem and $lb(s)$ is its lower bound. Indeed, if the inequality is valid than all subproblems in a task are subject to discard and thus should not be processed. In case such redundancy is discovered, the workunit is canceled using the respective BOINC command.

4 Experimental Results and Discussion

For our experiments, we use the following small enterprise [13] BOINC grid system. The server runs on a single-board 64-bit 'Orange Pi 2' machine, which has 1 GB RAM and sustainable Internet access. Clients are collected from the computer class: 12 32-bit machines with Intel Core 2 Duo E4600 CPUs with 2 GB RAM. The server machine uses Armbian OS, while clients run Linux Mint.

Parameters that were used by our load distribution algorithm are summarized in Table 1. These values were experimentally selected.

Table 1. Load distribution algorithm parameters

Parameter	Description	Value
T	the minimal number of workunits in a queue to start generating new tasks	5
S	the (maximal) number of subproblems in a workunit	2
N_{max}	the maximal number of steps performed by a client	10^5
$N_{totalmax}$	the maximal total number of steps to perform	$5 \cdot 10^9$

For testing the system performance, we conducted a number of computational experiments within an enterprise grid consisting of identical machines that have been described above. For testing, we considered benchmark optimization problems Biggs Exp5 and Exp6 function from the test suite [2, 18]:

$$\begin{cases} F_{BiggsEXP6}(x_1, x_2, ..., x_N) \to \min \\ x_i \in [-20, 20],\ i \in 1, \ldots, N. \end{cases} \tag{3}$$

The objective function is defined as follows:

$$F_{BiggsEXP6}(x_1, x_2, ..., x_N) = \sum_{n=1}^{13}(x_3 e^{-t_i x_1} - x_4 e^{-t_i x_2} + x_6 e^{-t_i x_5} - y_i)^2, \quad (4)$$

where

$$t_i = 0.1i, \; y_i = e^{-t_i} - 5e^{10t_i} + 3e^{-4t_i} \quad (5)$$

The problem was solved for 5- and 6-dimensional functions. In the 5-dimensional case, Biggs Exp6 function degenerates into Biggs Exp5 one, having 'x_6' replaced with '3' in the described formula. The obtained results as functions of the number of machines are shown in Fig. 2-6 presenting the 5-dimensional experiments measurements in blue color and the 6-dimensional ones in orange.

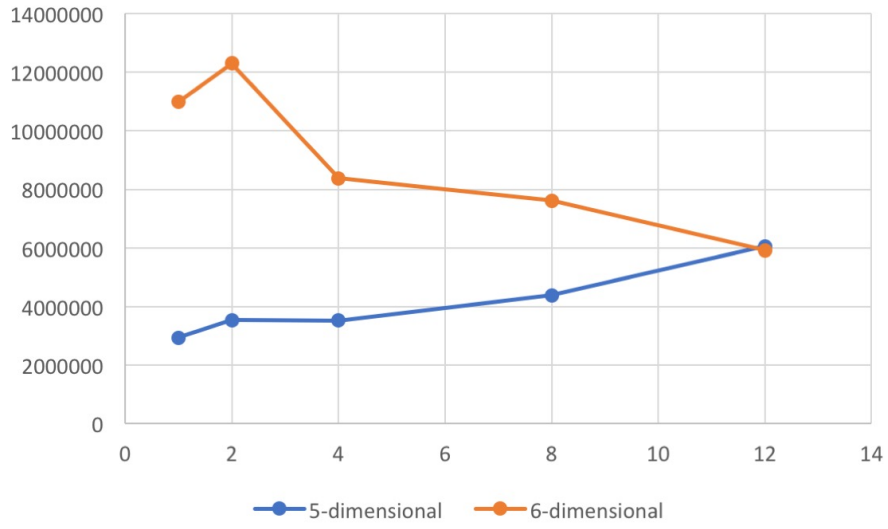

Fig. 2. Number of Performed Iterations

To evaluate efficiency of the implemented system, we use two main metrics, which are the elapsed computational time and the relative time for one task that is calculated in the following way:

$$T_{rel} = \frac{T_{elapsed}}{N_{iterations}} \quad (6)$$

The latter metrics is needed because the number of iterations may be affected by the rate of record updating ("search anomaly"). It separates the speedup due to changing the number of iterations and the parallelization.

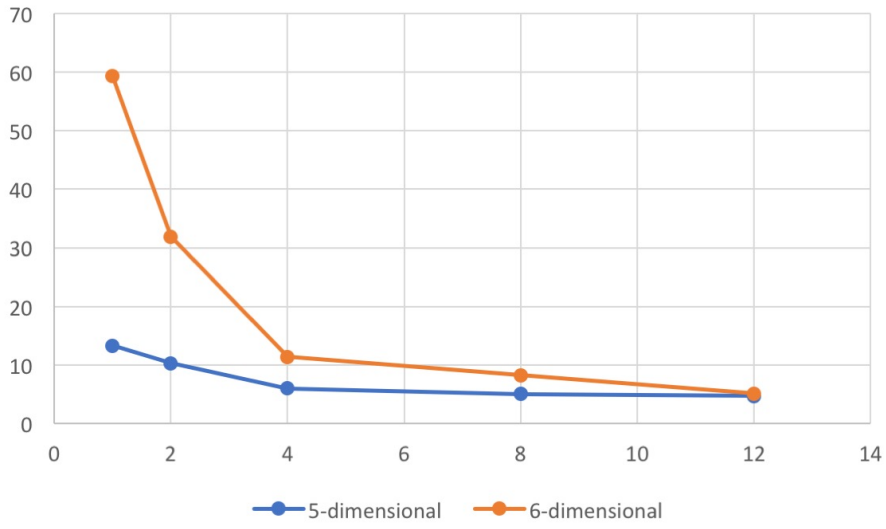

Fig. 3. Elapsed Time, min

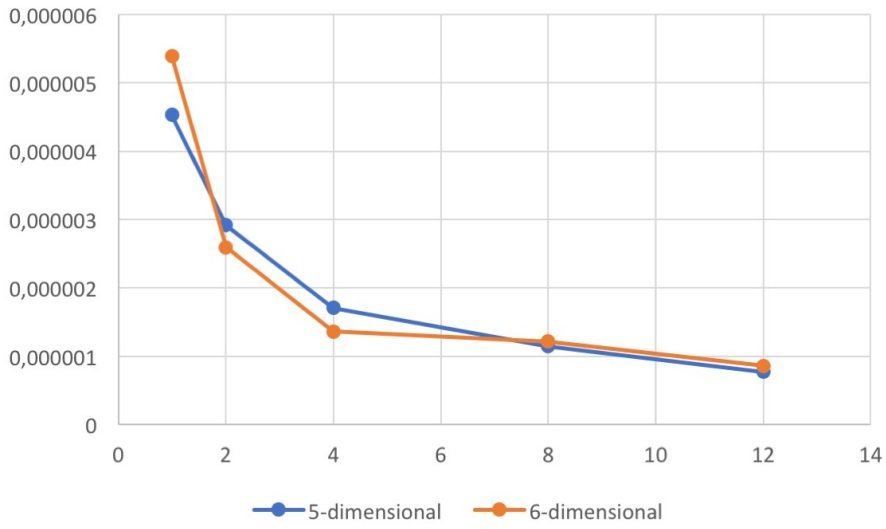

Fig. 4. Average Time for One Task, min

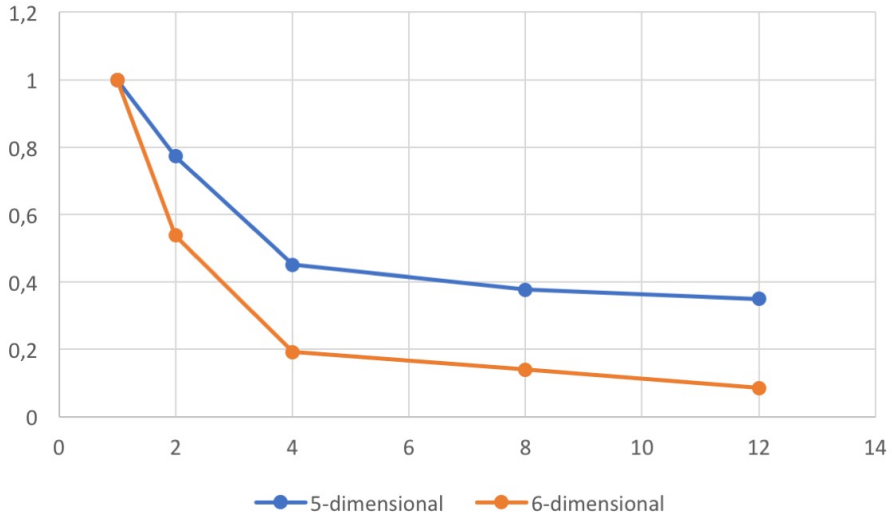

Fig. 5. Elapsed Time Decrease Rate

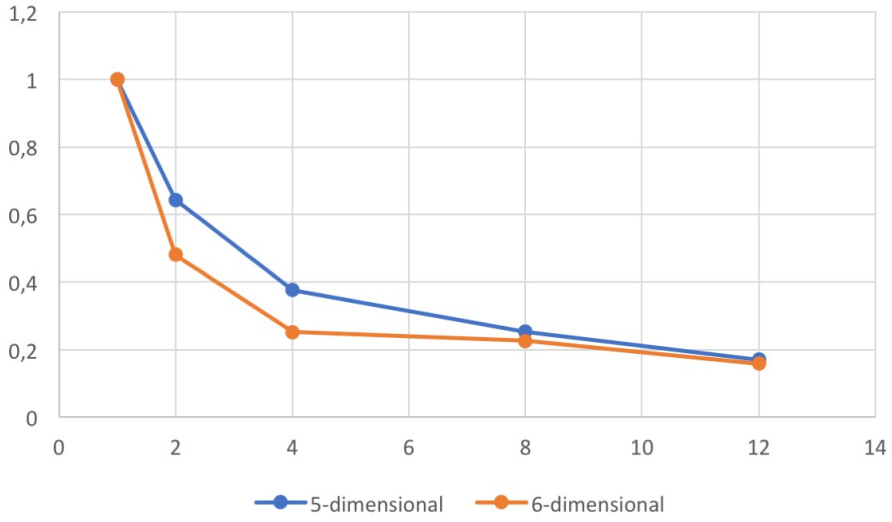

Fig. 6. Time for Task Decrease Rate

As seen from the obtained graphs, both metrics decrease when the number of machines grows. However, both graphs turn out to be far from linear as the rate of metrics decrease dramatically falls after more than four machines are involved. The major reason for this is that the number of tasks provided by the server turns out to be less than the overall computational ability of clients, which leads to the situation when a part of clients stay idle due to the lack of workunits.

In order to relieve this issue, we tried to change the criterion of necessity of new tasks so that the number of currently processing tasks would matter as well, but it led to rise in iterations number of nearly 10% in average and, therefore, to rise in elapsed time, so this strategy was rejected. We conclude that this issue needs to be observed in a more computationally complicated problem where the number of tasks would be considered as much greater than the number of computational units. In case this effect repeats in such an experiment, the strategy of distributing tasks between units should be corrected in a way, otherwise it would be considered as normal behaviour of the system.

Another remarkable result is that trends for the overall number of performed iterations do not match in two experiments. As it can be seen in the first graph in Fig.1, there is a clear decreasing trend in the 6-dimensional case. It leads to a superlinear speedup in elapsed time between 1 and 4 nodes. Surprisingly the situation with 5-dimensional case is opposite. this issue needs further investigation.

Using the strategy that has been described before, we get the following dependency: the sooner the current record value is updated, the less tasks would need to be processed, and it means the decrease in the number of performed iterations. As using multiple machines speeds up the process of updating record values, the number of iterations is supposed to decrease along with the time elapsed for the whole problem. However special efforts are needed to ensure fast record propagation and to filter the jobs queue before sending to client nodes.

5 Future Work

Analysis of the achieved results showed that the implemented system proves its efficiency. It should be noticed however that the experiments were conducted in a homogeneous enterprise grid which is a simplified infrastructure w.r.t. large-scale volunteer grid. Porting to a public volunteer grid will involve more experiments and multiple modifications. In particular, public grids are rather heterogeneous and have a very dynamic structure so considering these issues should be accounted in workunit generation process.

It was noticed that despite subproblems aggregation, the running time may still differ significantly for different tasks. Thus, an intelligent way of combining subproblems in a workunit is required. This implies the necessity of some way to predict the number of iterations for processing a subproblem.

In this paper, we tested our system using only unconstrained global optimization problems. Thus, we plan to make series of experiments with other math-

ematical problems, including different optimization benchmarks and problems of different types, e.g. constrained and discrete problems. Obviously the type of the problem can significantly affect the choice of the parallelization strategy. For example, finding the solution of systems of inequalities by Branch-and-Bound method [10] do not involve records exchange and thus can be viewed as a classical back-tracking. Conversely another popular approach to global optimization — branch and cut method assumes passing "cuts" (cutting planes) between processes [9] and thus can significantly increase the network overhead. The mentioned problems may require different load distribution strategies.

6 Conclusion

In this paper, we described a system implementing the Branch and Bound method on a small enterprise grid based on the BOINC infrastructure. Techniques to ensure uniform load distribution were proposed as well as subproblems filtering before sending to working nodes. The series of experiments on two benchmark problems were performed. The experimental results show that this algorithm is rather efficient in terms of total execution time decrease.

The obtained results, however, demonstrate a dramatic drop in time decrease after more than four machines are involved. As we suppose, it may be caused by a lack of workunits for computational nodes, and this assumption should be examined in a more complicated problem.

Acknowledgements. The work was supported by the RAS Presidium program No. 26 "Fundamentals of algorithms and software development for advanced ultra-high-performance computing." Authors are grateful to the head of supercomputer department of Federal Research Center "Computer Science and Control" Vadim Kondrashov and members of this department Ilya Kurochkin and Alexander Albertyan for helping in deploying and maintaining test BOINC grid.

References

1. The BOINC Wrapper . http://boinc.berkeley.edu/trac/wiki/WrapperApp/, 2008. [Online; accessed 15-May-2018].
2. Box-constrained test problems collection . https://github.com/alusov/mathexplib.git, 2017. [Online; accessed 17-June-2018].
3. BnB BOINC project codes. https://github.com/AndrewWhyNot/BOINC-Interval-Based-BnB.git, 2018. [Online; accessed 17-June-2018].
4. A Afanasiev, Yu Evtushenko, and M Posypkin. The layered software infrastructure for solving large-scale optimization problems on the grid. *International Journal of Computer Research*, 18(3/4):307, 2011.
5. Kento Aida, Wataru Natsume, and Yoshiaki Futakata. Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 156–163. IEEE, 2003.

6. Enrique Alba, Francisco Almeida, M Blesa, J Cabeza, Carlos Cotta, Manuel Díaz, Isabel Dorta, Joaquim Gabarró, Coromoto León, J Luna, et al. Mallba: A library of skeletons for combinatorial optimisation. In *European Conference on Parallel Processing*, pages 927–932. Springer, 2002.
7. David P Anderson. Boinc: A system for public-resource computing and storage. In *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004.
8. Kurt Anstreicher, Nathan Brixius, Jean-Pierre Goux, and Jeff Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91(3):563–588, 2002.
9. Matthias Elf, Carsten Gutwenger, Michael Jünger, and Giovanni Rinaldi. Branch-and-cut algorithms for combinatorial optimization and their implementation in abacus. In *Computational Combinatorial Optimization*, pages 157–222. Springer, 2001.
10. Yuri Evtushenko, Mikhail Posypkin, Larisa Rybak, and Andrei Turkin. Approximating a solution set of nonlinear inequalities. *Journal of Global Optimization*, pages 1–17, 2017.
11. Yuri Evtushenko, Mikhail Posypkin, and Israel Sigal. A framework for parallel large-scale global optimization. *Computer Science-Research and Development*, 23(3-4):211–215, 2009.
12. Wenjie Fang and Uwe Beckert. Parallel tree search in volunteer computing: a case study. *Journal of Grid Computing*, pages 1–16, 2017.
13. Evgeny Evgenevich Ivashko. Enterprise desktop grids. *Programmnye Sistemy: Teoriya i Prilozheniya [Program Systems: Theory and Applications]*, (1):19, 2014.
14. Luc Jaulin. *Applied interval analysis: with examples in parameter and state estimation, robust control and robotics*, volume 1. Springer Science & Business Media, 2001.
15. Samtsevich A. Posypkin M. Sukhomlin V. Khrapov N., Rozen V. and Oganov A. Using virtualization to protect the proprietary material science applications in volunteer computing. *Open Engineering*, 8(1):57–60, 2017.
16. Michael J Litzkow, Miron Livny, and Matt W Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111. IEEE, 1988.
17. Attila Csaba Marosi, Zoltan Balaton, and Peter Kacsuk. Genwrapper: A generic wrapper for running legacy applications on desktop grids. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–6. IEEE, 2009.
18. Mikhail Posypkin and Alexander Usov. Implementation and verification of global optimization benchmark problems. *Open Engineering*, 7(1):470–478, 2017.
19. Alexander Semenov, Oleg Zaikin, Dmitry Bespalov, and Mikhail Posypkin. Parallel logical cryptanalysis of the generator a5/1 in bnb-grid system. In *International Conference on Parallel Computing Technologies*, pages 473–483. Springer, 2011.
20. Sergey Smirnov, Vladimir Voloshinov, and Oleg Sukhoroslov. Distributed optimization on the base of ampl modeling language and everest platform. *Procedia Computer Science*, 101:313–322, 2016.
21. Oleg Sukhoroslov, Sergey Volkov, and Alexander Afanasiev. A web-based platform for publication and distributed execution of computing applications. In *Parallel and Distributed Computing (ISPDC), 2015 14th International Symposium on*, pages 175–184. IEEE, 2015.
22. B Tlan and Mikhail Posypkin. Efficient implementation of branch-and-bound method on desktop grids. *Computer Science*, 15(3)):239–252, 2014.