Parameter Sweep Experiments in Hybrid Computing Systems with R Language

Alexander Rumyantsev, Oleg Sukhoroslov, Anna Eparskaya, Enrico Blanzieri, Valter Cavecchia

Abstract*: High-performance and distributed computing are the backends aimed to equip researchers in addressing computationally intensive modeling and simulation problems. While high-performance systems, such as supercomputers, require specific knowledge and algorithms to be efficiently utilized, the so-called desktop grid systems and hybrid computing systems (as a combination of the former two classes) provide a relatively easy configurable and cheap solution for obtaining necessary computational power. Observing a large number of researchers using R language as a primary solution for applied studies (including modeling, simulation and statistical study in various fields), we introduce the RBOINC package for R language. The package allows to easily perform the so-called parameter sweep type of experiments in R language over BOINC infrastructure, the de-facto standard software solution for desktop grid systems. We discuss the implementation, uncover the possible use-cases and provide some directions for future work.

***Index Terms*: Desktop Grid, Embarrassingly Parallel, BOINC, Parameter Sweep, Everest, Causal Relationship.**

I. INTRODUCTION

Recent developments in computing systems allowed to replace many real-life experiments with the so-called "in silico" modeling. This, in turn, allowed to solve many problems of the *Grand Challenges* list [9], where a number of problems valuable to the humanity are stated. However, still many of them remain unsolved (e.g. climate change modeling, fighting diseases, fundamental nature of matter understanding etc.), in many cases lacking computational resources due to high computational complexity. Thus, a further increase in computing power is required.

For more than a decade, the so-called Moore's empirical law of doubling the performance of a chip every two years was pushed forward by increasing the number of compute cores of a central processor unit (CPU) [17], as an effect of saturation of the CPU frequency (with modern values likely between 2 and 3 GHz) due to a number of physical phenomena. As a result, multicore and manycore architectures are now widely used, starting from handheld computational devices, and up to high-performance computing clusters (HPC, also known as supercomputers). New systems required a new *parallel programming* paradigm for achieving full system utilization, and effective parallel algorithms still attract the attention of researchers [3, 4].

Typical hardware for Grand Challenges is an HPC or a group of HPCs known as Grid. Distinguishing features of an HPC are the high speed and low latency data transfer network called interconnect and high homogeneity of the resources. However, recent advances in co-processor design and introduction of general-purpose graphics processing units (GPGPU), together with tight limits for power consumption, lead to a more heterogeneous structure of a modern HPC. Moreover, due to hardware specifics, an HPC is expensive both in terms of capital and operational expenses. At that, Distributed Computing has been widely adopted, addressing the reliability, affordability and power consumption issues of an HPC.

One of the typical ways to organize Distributed Computing (DC) is the Desktop Grid (DG), being a loosely coupled system of computing devices (including personal computers, servers, laptops, and even smartphones) with centralized management. Being a widely adopted technology, the class of DG systems is subdivided into the so-called Enterprise DG (EDG [6]) consolidating the computational resources of an organization/group of organizations, and Volunteer Computing [10] mainly focused on utilizing idle time of volunteer donated resources for scientific computing purposes. Being a highly heterogeneous and mostly unstable result (in terms of resources available, compute time predictability etc.), the DG requires some additional treatment of the computational process, latter being software supported with an appropriate backend, e.g. the most popular BOINC software [1].

To overcome difficulties of developing an application that both fits the high-performance and distributed environment, many researchers are focused on providing a seamless and flexible way of both aforementioned hardware backend

Revised Manuscript Received on May 28, 2019.

Alexander Rumyantsev, Institute of Applied Mathematical Research, Karelian Research Centre of RAS, Petrozavodsk, Russia; Petrozavodsk State University, Petrozavodsk, Russia; ar0@krc.karelia.ru.

Oleg Sukhoroslov, Institute for Information Transmission Problems (Kharkevich Institute) RAS, Moscow, Russia; sukhoroslov@iitp.ru.

Anna Eparskaya, Petrozavodsk State University, Petrozavodsk, Russia.

Enrico Blanzieri, DISI, Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 9, 38123 Povo (TN), Italy; CNR-IMEM, Via alla Cascata 56/C, 38123 Trento, Italy.

Valter Cavecchia, CNR-IMEM, Via alla Cascata 56/C, 38123 Trento, Italy.

This research is partially supported by RFBR, projects 18-07-00147, 18-07-00156, 18-37-00094, 19-07-00303.

utilization. This is mainly done by some hybrid computing software running on top of the HPC and DG management software, and providing some application programming interface (API) for applications development. Such hybrid computing approach, while narrowing the class of problems that can be solved with (mostly to the so-called embarrassingly parallel applications), in general allows to dramatically increase the computational power of a system by combining multiple heterogeneous environments in a single pool of resources. Moreover, appropriate tuning allows to utilize idle resources of HPC, that increases the overall efficiency of such systems [20, 16].

The class of applications that allow to use DC and hybrid environments for parallelization are the so-called *parameter sweep applications (PSA)*. PSA require a large amount of computing resources in order to run a large number of similar computations varying combinations of parameter values. These applications are becoming extremely important in science and engineering. PSAs involve some input set of computational parameters and files. Each parameter has its range of values, and multiple computations, or tasks, are then run for different combinations of parameters' values. As a rule, each task runs the same executable but with different arguments and input files that depend on parameter values. Each task is supposed to produce some output, typically in the form of the model's output parameters, describing the obtained characteristics. The resulting set of all task outputs represents the result of the whole parameter sweep experiment.

While PSAs can be dramatically time-consuming and require enormous amount of processor time, the individual tasks are independent and can be run in parallel. Therefore, this class of applications is naturally suited for DC. The potential speedup that can be achieved by running PSAs across DC resources is significant. However, the heterogeneous and complex nature of such environments requires the use of high-level tools that automate task submission, scheduling, data movement and failure recovery.

One important case of PSA class application is the so-called Causal relation discovery (CRD) problem. CRD from observational data is a research topic that has recently attracted attention in the artificial intelligence and machine learning communities. Finding direct causal relationships between variables of interest can impact the analysis of the data originated in scientific and industrial practice. The PC-algorithm [14] infers causal relationships between variables by means of Conditional Independence (CI) tests. PC application permits to discover direct causal relationships between correlated variables and it has been applied to a wide range of domains including Yeast gene expression data [7] and QSAR/QSPR analyses [12]. To overcome the computational limitations of PC, the NES^{2}RA [2] approach was introduced recently, which is based on stratified subsetting of the variables (one of the possible implementations of PSA approach) and whose backend relies on the *gene@home* DG project running on the BOINC platform. However, to provide a deeper flexibility in the approaches used for causality attribution in CRD problem, it is necessary to use the methods implemented in R language, at the same time keeping the supporting BOINC infrastructure and the volunteers of gene@home. Development of an R package for supporting BOINC infrastructure in PSA class tasks will not only equip applied scientists studying the CRD problem, but also be useful as a parallel backend for various fields of applications requiring PSA-type numerical experiments. Thus, the main purpose of this paper is to introduce the RBOINC package that allows to perform parameter sweep type experiments in R language using BOINC infrastructure.

The structure of this paper is as follows. First, we study the available solutions in R language for parallel and distributed computing, thus, motivating the new RBOINC package introduction. Then we motivate the usage of DG and hybrid systems as parallel backend for PSA type applications. We also give a brief introduction into CRD application as a use-case of PSA class, and study its efficiency. Finally, we make conclusions and suggest future work.

II. PARALLEL COMPUTING IN R LANGUAGE

In this section, we consider organization of parallel and distributed computing in R programming language. Since the standard extension source is the Comprehensive R Archive Network (CRAN), we based our classification on the solutions present in CRAN package repository.

It is necessary to note that one of the key features of R language is the widely implemented vectorization of computations, starting from the level of arithmetic operations and basic functions. Vectorization is one of the ways to speed up the computations and, if possible, perform parallel computations. The main tool for vectorizing the code in R language is the `apply` family of functions (which is quite similar to the for loop with independent, and not necessarily sequential, iterations). Selection of a specific function from the `apply` family is based on the type of data processed, and the required output. Usually this command allows you to apply some function to a vector (or array) of parameters and collect the result in a vector. Such an approach can potentially lead to a significant acceleration of calculations overlarge data sets (naturally implementing the Single Instruction Multiple Data approach).

However, R language offers some extensions to fully utilize the potential of a parallel hardware. The base in parallel computations in R is the `parallel` package, the successor of multi-core shared memory computing functions (previously provided by `multicore` package) and the multi-server computing functionality of the `SNOW` package. The `SNOW` package allows to organize the so-called Beowulf-type cluster (i.e. Simple Network of Workstations) with (possibly slow) general purpose interconnect (e.g. direct network sockets), as well as use specific HPC hardware with the Message Passing Interface (MPI) infrastructure. Some extensions of the `SNOW` package include the packages `snowfall` (an abstraction layer made to simplify the programming), and `snowFT` (introducing the fault-tolerant computing).

Using the `parallel` package as a backend, a higher-level abstraction is constructed, as a combination of the `foreach` package (implementing the concept of iterators) and the do package family (`doParallel`, `doMC`,

doMPI, doSNOW, etc.). This combination allows to perform a general iteration for each item in the set without using an explicit loop counter, thereby parallelizing the execution of the loop (which is also useful if the data is received sequentially, and not fully available at the beginning of computations). The do packages allow to organize the backend, including working with HPC hardware.

MPI is the most common standard for data exchange interfaces in parallel programming. In this regard, there are a number of implementations using this standard. It is supported in R via the RMPI package offering access to numerous functions from the MPI API. The RHPC package provides the apply function on the basis of MPI. We also note that low-level parallel programming can be performed using special package wrappers for parallel programming tools and languages outside the R environment (such as C++), including RcppParallel and RcppBlaze, which also include the OpenMP (shared memory multiprocessing) standards.

R also offers some packages to use the existing high-level infrastructure and perform computations with the help of various parallel schedulers (e.g. LSF, SGE, SLURM). Such packages basically deliver the computing tasks to the scheduler, and the scheduler distributes the tasks among the nodes. The batchtools, flowr and clustermq packages allow you to interact in the usual way with a wide variety of high-performance schedulers, including SLURM, TORQUE, Docker Swarm, etc. In batchtools, the multicore and socket mode allow parallelization on local machines, and several computers can be connected via secure channel (SSH) to create an improvised cluster.

The package RSLURM works with the celebrated scheduler SLURM (widely used among HPC resources, including the most powerful machines in the world). The package allows to manage parallel computing resources and tasks. It also distributes the tasks after splitting the parameter space, and collects results in an asynchronous manner, thus, implementing the PSA paradigm.

A number of packages in R allows to perform high-performance computing using GPGPU. Many of them implement algorithms for certain studies. For example: the package gputools provides algorithms for data mining; cudaBayesreg package is used for high-performance statistical analysis; the rgpu package aims to accelerate the analysis of bioinformatics using a graphics processor; the permGPU package is used in statistical genomics. In addition, there are packages that provide an interface for a specific language (OpenCL packages, RCUDA, RviennaCL). To evaluate matrix and vector operations using GPU coprocessors, the packages gmatrix, gpuR may be used. Interim calculations can be stored on the coprocessor and reused with potentially significant performance improvements by minimizing data movement.

There are solutions for organizing distributed computing in R. Basically, these solutions are oriented to distributed data structures for working with Big Data (ddR, datadr, distcomp, DSL, startR). In addition, there are packages RHIPE and rmr, which provide an interface between R and the Hadoop framework widely used to develop and execute distributed programs. Also, the in-database calculations can be performed using the Teradata Aster distributed analytic platform and the toaster package.

However, currently there are no active packets in the repository for DG and VC hardware and paradigm utilization. Although, there were several attempts to create packages based on DG. For example, the multR package could implement the idea of the SNOW package on the grid computing platform. GridR also used the existing DG (based on Condor and OpenScienceGrid technologies). But these packages were never released.

In conclusion, we note that in the CRAN repository there are a lot of solutions both low-level and high-level parallel computing. However, to the best of our knowledge, there are no solutions based on DG and VC in R, and, in particular, for the BOINC platform, standard for DG computing.

III. RBOINC PACKAGE FOR R LANGUAGE

In this section, we describe the package RBOINC implementing the PSA type parallel computing (currently as a prototype). In the implementation, we used remote access to the BOINC server using the BOINC API, which technically means implementation of some specific POST HTTP-requests with special XML structure attached to the request body.

In the RBOINC package, there are currently two functions, rboinc_batch() and rboinc_submit(). The rboinc_batch() function prepares the job packages for subsequent submission to the BOINC server. The function must receive the following mandatory arguments to the input: f - function for further calculations, params - a set of parameters in the so-called data.frame structure (widely used as an analogue of relational tables). An optional workunits parameter (with default value 1) can also be specified, which is related to the number of workunits rboinc_batch() function is to produce.

```
rboinc_batch(f, params, workunits = 1)
```

At runtime, the rboinc_batch() function saves the parameter set params, the function f:

```
saveRDS(params, file = "params.RDS")
saveRDS(f, file = "f.RDS")
```

The data is saved into a binary format RDS widely used in R scripts. Note that every node receives the whole set of parameters.

Then the params parameter set is divided into parts according to the required number of workunits by filling a specific template (with the help of whisker package) to produce the required batch submission request to the BOINC server as demonstrated by the following code:

```
start=1
end=nchunk
```


Published By:
Blue Eyes Intelligence Engineeri
& Sciences Publication

```
  r=readLines("r.txt")
  for (i in 1:workunits) {
  r_task<-whisker.render(r,list(pkgs = pkgs,
  start = start,end = end,i = i))
  writeLines(r_task,
file.path(getwd(),"tasks",
  paste0('task',i,'.R')))
  start=start + nchunk
  end=min((nchunk + start),
nrow(params))}
```

Each workunit package contains a function `f` and a set of parameters `params`. In addition, it includes the procedure wrapper for BOINC clients, enabling them calculate the function `mcmapply` for the function `f` with a specified range of parameters and save the result of the calculation. Finally, the batch of such workunits is saved to the current working directory, each workunit having unique sequential name within a batch.

The `rboinc_submit()` function sends ready workunit batches to the BOINC server. The following arguments must be specified: `app_name`, which is the name of the BOINC application (set up by BOINC project administrator), `auth` is the BOINC user's identifier (the user should have suffifient privileges to use the remote procedure calling mechanism for task submission), `url_proj` is the BOINC project's URL address (in the format: http://boinc-server/proj).

```
boinc_submit (app_name, auth, url_proj)
```

Next, the function reads the XML request template.

```
req <- readLines("req.txt")
```

The template is filled with the received arguments and information from the workunit batch in the current working directory. Finally, the appropriate POST request is created and sent to the BOINC server with the help of `httr` package.

```
  for (j in list.files())
  {
  num_b = num_b + 1
  r_post <- whisker.render(req,
  list(app_name = app_name,auth = auth,
  name = paste0('batch_',num_b),
  url = tools::file_path_as_absolute(j),
  size = file.info(j)$size,
  md5 = q<-as.vector(tools::md5sum(j))))
  print(POST(paste0(url_proj,'/submit_rpc_handler.php'),
  body=list(request=r_post)))
  }
```

In the nearest future it is planned to finalize the prototype with new functions, in particular, it is necessary to receive the results of calculations from the BOINC server (which could currently be downloaded manually). To do this, we also need to use remote access to the BOINC server. Finally, the results need to be merged in appropriate order and returned to the R user.

IV. HYBRID COMPUTING

To minimize the effect of resources volatility and guarantee reasonable time of response from the DC system, it would be preferable to include reliable resources, such as HPC, in the computing resources pool. To implement this key feature, a further extension of the RBOINC package towards usage of hybrid infrastructures including standalone clusters, grids and clouds is required. An integration of RBOINC with Everest platform would allow to perform the suggested evolution. Future work will investigate this opportunity by implementing remote job submission from RBOINC to Everest via the REST API provided by the latter. To initiate this work, we discuss the capabilities of Everest platform below.

Everest [15] is a web-based Platform-as-a-Service that equips researchers with programming interfaces and tools to perform distributed computing on heterogeneous pool of resources provided by the users. The platform is available online to all interested users [5]. A typical Everest application has a number of *inputs* that are implemented as requests from an application, and a number of *outputs* that deliver the result of computation of some request. Such applications may be combined in software-defined way. Each application request is considered as a *job* consisting of one or more computational *tasks* handled by the Everest platform on the resources specified by a user. A specific configuration of the application and platform setup may be published as a *template* for further use by other researchers. A ready to use application is published via RESTful web service for further incorporation into a workflow, and such a publishing includes a ready web form for application management, as well as sophisticated access control.

Many-task applications may be implemented via software-defined management of new task instances implemented as dynamic task allocation with the help of Everest API. Another option is to use the generic application [18] for running a large number of independent parametrized tasks, i.e. parameter sweep experiments, available in the platform.

The Everest platform allows to flexibly attach external computational resources in a heterogeneous way. To this aim, integration with standalone machines and clusters is done with the help of a specific program called *agent* [13]. The agent is executed on the computational resource controlling the computation of tasks, staging input files, performing monitoring and reporting activities, and giving back the results as well. The platform also supports integration with grid infrastructures [13] and clouds [19]. Everest users can flexibly bind the attached resources to applications. In particular, a user can specify multiple resources, possibly of different type, for running an application [13]. In this case the platform performs dynamic scheduling of application tasks across the specified resource pool. At that, Everest could provide necessary flexibility to serve as a hybrid computing backend for PSA-type applications in R language. We leave a deeper investigation of particular implementation of RBOINC package over Everest for future research.

Published By:
Blue Eyes Intelligence Engineeri
& Sciences Publication

V. PARAMETER SWEEP-TYPE EXPERIMENTS IN R: A CASE STUDY

The case study focuses on Gene Regulatory Network (GRN) expansion that is the task of identification of genes related to a GRN, a frequent task in the biological practice. In fact, a biologist working on a research topic has prior knowledge about relevant genes and their relationships that can be formulated as a reasonable hypothesis about putative interesting networks (a similar task for pathways is addressed by the recently proposed Clic project (www.gene-clic.org)). The regulation of genes can be modelled by a GRN whose nodes are the genes and the connections (either facilitatory or inhibitory) represent the causal relationships between genes' transcription abundance. A GRN does not capture the details of the whole set of events that connect the transcription of two genes, however the correct description of a GRN allows either predicting the behavior of the system or manipulating it in the direction of a specific aim. The high-throughput technologies for gene expression measure, such as microarray and RNA-seq experiments, have given the chance to reverse engineering GRNs. Various bioinformatic approaches have been proposed in order to identify gene networks [8]. The task of GRN expansion that is the identification of genes related with the genes of the GRN, has received less attention than networks reconstruction, although it is a frequent task in the biological practice.

In order to expand GRNs, Asnicar and co-authors have proposed NES^{2}RA [2] and already applied it to *Arabidopsis thaliana*, *E. coli* and *Vitis vinifera* data. The method can be described as a parameter sweep over the application of a causal relationship discovery algorithm (PC-algorithm) on subsets of variables. The group that proposed the method plans to apply it to several whole genomes. In this context the migration of NES^{2}RA to the technology proposed in this paper can facilitate its application to the ever-growing number of expression datasets available for different organisms. The foreseen application and the migration will contribute to mitigate the main drawbacks of the current implementation of NES^{2}RA (namely the gene@home BOINC project described below) that includes 1) the rather long elapsed time in order to have the results and 2) the parameter tuning necessary to apply it to new datasets. In order to mitigate the high elapsed-time of NES^{2}RA and provide in real time a biologist with a list of genes that expand a given GRN, a viable solution is to pre-compute the expansion of each gene of a genome and compose the results on-the-fly at query time. The composition can be attained with a ranking aggregation techniques applied to the single expansion lists of the genes of the GRN. The success condition of the system is that the aggregated list and the list obtained by NES^{2}RA on the original GRN should have comparable or better quality. To this end preliminary expansions of *Vitis vinifera* genes showed encouraging results. A systematic application of this approach to several genomes can establish a precious resource that biologists can browse in real time with a potentially high impact in several different areas of biological research and biotechnology development. The application of NES^{2}RA to one-gene-at-a-time gene network expansions (OneGenE for short) to several data-sets attracts many volunteers and consequently arises the need to guarantee a computational grid as efficient as possible. In depth analysis of the relationships between the parameters and the behaviour of the grid is necessary and the migration to RBOINC can achieve this goal as shown below.

A. The gene@home project

Since year 2014 CNR-IMEM, in collaboration with University of Trento, has run TN-Grid, a DG infrastructure based on the BOINC platform. TN-Grid relies (May 2018) on 1692 registered users and 26130 computers almost a third of which participated recently (522 users and 8061 computers). TN-Grid has hosted gene@home (gene.disi.unitn.it), a project developed with Edmund Mach Foundation that has the goal to expand GRNs with putative causal relationships with genes discovered by analyzing gene expression data. More in detail, the gene@home BOINC project hosts an implementation of the NES 2 RA algorithm [2], based on PSA and on stratified subsetting of variables fed to the PC algorithm [14], using an iterated version of it (called PC-IM) [11] that is used for expanding GRNs. The output of a single run of PC-IM is a ranked list of putative interactions between couple of genes. The outputs of runs with various parameters are integrated using ranking aggregation techniques.

The application running the PC-IM on the BOINC platform has the following main parameters: *iter* (number of iterations), *tsize* (the subset dimension, i.e. the tile size) and *alpha* (α, the statistical significance level). These parameters have to be carefully chosen by balancing the execution speed of the application, the accuracy of the results and the statistical errors. This kind of choices has to be completely rethought if the input dataset changes. Additionally, parameter *npc*, the number of PCs that are inserted in a single workunit (the BOINC unit of work that will be distributed to the volunteers), and *cutoff*, a way of shortening the size of the output file by removing the last (less significant) elements found in the ranked output list, have to be selected in order to optimize server and client performance and the network bandwidth.

Optimal choice of PC-IM and BOINC parameters based on PSA-type preprocessing is critical in the forthcoming OneGenE project. The main idea of OneGenE is to eliminate the need of repeated calculations of GRN expansions list by creating a public database containing the single expansions for each gene of an organism. Once created, the gene expansion list may be used to produce the final expansion list of any local gene network. This is in sharp contrast to the previous gene network expansion methods, where results were affected by the knowledge of the LGN inserted in each experiment, without a possibility to use previously obtained results. With this set of parameters: (*alpha* = 0.05, *tsize* = 1000, *iter* = 2000) an expansion of any single gene in the *Vitis vinifera* Vespucci dataset (28013 genes) is made up up of ~ 1636M PCs, each of them needs ~ 2.34s of computational time for each core of our reference machine (see *exp_id* = 4, *id* = 1, in Table 2). Due to large computational time, it is crucial to set up optimal parameter values at the early stages of the computational experiment.

B. Benchmarking gene@home

One of the key parameters of a BOINC workunit is an estimate of the compute

time, which allows server to perform efficient scheduling, as well as to assign the so-called credits (virtual reward of a volunteer) in a fair way. If a volunteer participates in many DG projects, appropriate compute time estimate allows the BOINC client to better choose its scheduling parameters (cache, priority etc.). The calculation is usually done by running a small number of randomly generated PC on a benchmark machine, getting the execution time and calculating the needed flops with the formula *pc_time * host_flops * scale_factor.*

Table 1. Structure of table benchmark

Column	Type	Null	Default
id	int(11)	No	
exp_id	int(11)	No	
pc_tsize	int(11)	No	
pc_alpha	decimal(4,3)	No	
pc_time	float	Yes	NULL
host_flops	double	No	
host_iops	double	No	

While Table 1 demonstrates the parameters of a workunit benchmark, Table 2 delivers the results of benchmarking for various system parameters. The host name field has been removed from this table, for more readability, the reference benchmark machine is an Intel 4770k workstation running Linux. The pc_alpha (α), pc_tsize (*tsize*) parameters and the choice of the input dataset (*exp_id*) are related to the execution time (pc_time), whereas the number of floating point operations (host_flops) and input/output operations (host_iops) are defined by *exp_id*. For *exp_id* 3 (*Escherichia coli*, Colombos gene expression dataset, 2470 columns x 4065 rows) there is a dramatic increase of the execution time by moving the *tsize* from 100 to 1000. Other combinations of the application parameters yield very different times.

Table 2. Contents of table benchmark

id	exp_id	pc_tsize	pc_alpha	pc_time	host_flops	host_iops
13	3	100	0.050	2.4081	4368114902	16780523829
14	3	200	0.050	10.6001	4366259032	16818293110
15	3	100	0.010	0.9404	4368114902	16780523829
16	3	200	0.010	4.0475	4368114902	16780523829
19	3	500	0.050	180.161	4357320988	16864725110
20	3	1000	0.050	609.37	4357320988	16864725110
1	4	1000	0.050	2.3407	4308551041	16785697288
21	4	1000	0.010	1.3426	4357320988	16864725110
2	4	2000	0.050	9.7945	4308551041	16785697288
7	4	500	0.050	0.8325	4308551041	16785697288
5	5	200	0.050	0.4698	4308551041	16785697288
6	5	500	0.050	0.7485	4308551041	16785697288
3	6	200	0.050	0.4391	4308551041	16785697288
4	6	500	0.050	0.5849	4308551041	16785697288
10	7	1000	0.050	0.9888	4362642280	16828186735
11	7	1500	0.050	1.9549	4362642280	16828186735
12	7	2000	0.050	3.593	4362642280	16828186735
8	7	100	0.050	0.4086	4362642280	16828186735
9	7	500	0.050	0.5214	4362642280	16828186735
23	8	1000	0.050	0.9839	4366259032	16818293110
17	9	1000	0.050	5.6254	4344975227	16754117890
18	9	1000	0.010	2.773	4344975227	16754117890
22	9	2000	0.050	21.4383	4357320988	16864725110
25	9	500	0.050	1.258	4374074767	16809680358
24	10	1000	0.050	15.8173	4374074767	16809680358

The drawbacks of TN-Grid that were apparent while running the gene@home project proved to be 1) the high elapsed time in order to have the results, 2) the parameter tuning for new datasets, 3) the limitation to the PC-algorithm and 4) absence of non-volatile resources, such as HPC. In fact, given a dataset and a GRN to expand, the system distributes several subtasks, and the slowest subtask to be completed determines the overall computation time. Dimension and runtime of the subtasks is influenced by the nature of the dataset, by the availability of volunteers and by the setting of various parameters: a) functional parameters of the algorithm, b) technical parameters of its implementation, and crucially c) grid-related parameters of the application and of the framework whose values depend on the computational power. At that, migration to RBOINC can help because it permits conveniently to test and balance the value of the parameters. Moreover, it can easily permit the application to other discovery algorithms and to include HPC in the computation when needed, for example when the data cannot be legally distributed on a grid for privacy issues.

VI. CONCLUSION AND DISCUSSION

In this paper we introduced the prototype of RBOINC package for R language equipping researchers with a simple way to perform parameter sweep-type experiments over Desktop Grid infrastructure based on the celebrated BOINC software. We discussed some applications naturally implementing this type of experiments, including the causal relation discovery, and our further plans extend to practical evaluation of the proposed package in real life experiments which, however, we leave for future research.

REFERENCES

1. D.P. Anderson "BOINC: A system for public-resource computing and storage" in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, IEEE Computer Society, Washington, DC, USA, 2004, pp. 4-10, available: https://doi.org/10.1109/GRID.2004.14
2. F. Asnicar, L. Masera, E. Coller, C. Gallo, N. Sella, T. Tolio, P. Morettin, L. Erculiani, F. Galante, S. Semeniuta, G. Malacarne, K. Engelen, A. Ar-gentini, V. Cavecchia, C. Moser, E. Blanzieri "Nessra: Network expan-sion by stratified variable subsetting and ranking aggregation" in *The InternationalJournal of High Performance Computing Applications* issue 32(3), 2018, pp. 380–392, available: https://doi.org/10.1177/1094342016662508
3. N. Faujdar, S. Saraswat "A Roadmap of Parallel Sorting Algorithms using GPUComputing" in *2017 IEEE INTERNATIONAL CONFERENCE ON COMPUTING, COMMUNICATION AND AUTOMATION (ICCCA)*. pp. 736–741.
4. W. Gan, J.C.W. Lin, H.C. Chao, J. Zhan "Data mining in distributed environment: a survey" in *WILEY INTERDISCIPLINARY REVIEWS-DATA MINING AND KNOWLEDGE DISCOVERY* issue 7(6), 2017, available: https://doi.org/10.1002/widm.1216
5. IITP RAS: Everest [Online]. Available: http://everest.distcomp.org/
6. E. Ivashko "Enterprise Desktop Grids" in Proceedings of the Second InternationalConference BOINC-based High Performance Computing: Fundamental Researchand Development (BOINC:FAST 2015) vol. 1502, 2015, pp. 16–21, available: http://ceur-ws.org/Vol-1502/paper2.pdf
7. M.H. Maathuis, D. Colombo, M. Kalisch, P. B ̈uhlmann"Predicting causal effectsin large-scale systems from observational data" in Nature Methods issue 7, 2010, pp.247, available: https://doi.org/10.1038/nmeth0410-247
8. D. Marbach, J.C. Costello, R. K ̈uffne, N.M. Vega, R.J. Prill, D.M. Camacho, K.R. Allison, The DREAM5 Consortium, M. Kellis, J.J.

Collins, G. Stolovitzky"Wisdom of crowds for robust gene network inference" in *Nature Methods* issue 9, 2012, pp. 796, available:http://dx.doi.org/10.1038/nmeth.2016
9. NationalScienceFoundationAdvisoryCommitteeforCyberin-frastructure:Taskforceongrandchallenges,finalreport(2011) [Online]. Available: http://www.nsf.gov/od/oci/taskforces/TaskForceReportCampusBridging.pdf
10. M. Nouman Durrani, J.A. Shamsi "Volunteer computing: requirements, chal-lenges, and solutions" in *Journal of Network and Computer Applications* issue 39, 2014,pp.369–380, available: https://doi.org/10.1016/j.jnca.2013.07.006
11. A. Rumyantsev, A. Eparskaya, E. Blanzieri, V. Cavecchia "On Distributed RComputations over BOINC" in *Proceedings of the Third International Confer-ence BOINC:FAST 2017* vol. 1973, 2017, pp. 108–113, available: http://ceur-ws.org/Vol-1973/paper14.pdf
12. N. Sizochenko, A. Gajewicz, J. Leszczynski, T. Puzyn "Causation or only corre-lation? application of causal inference graphs for evaluating causality in nano-qsarmodels" in *Nanoscale* issue 8, 2016, pp. 7203–7208, available: https://doi.org/10.1039/C5NR08279J
13. S. Smirnov, O. Sukhoroslov, S. Volkov "Integration and combined use of dis-tributed computing resources with Everest" in *Procedia Computer Science* issue 101, 2016, pp. 359–368
14. P. Spirtes, C. Glymour "An algorithm for fast recovery of sparsecausal graphs" in *Social Science Computer Review* issue 9(1), 1991, pp. 62–72, available: https://doi.org/10.1177/089443939100900106
15. O. Sukhoroslov, S. Volkov, A. Afanasiev "A web-based platform for publicationand distributed execution of computing applications" in *Parallel and DistributedComputing (ISPDC)*, 14th International Symposium on, 2015, pp. 175–184, Available:https://doi.org/10.1109/ISPDC.2015.27
16. O. Sukhoroslov "Integration of Everest platform with BOINC-based desktopgrids" in *Proceedings of the Third International Conference BOINC:FAST2017* vol.1973, 2017, pp. 102–107, available: http://ceur-ws.org/Vol-1973/paper14.pdf
17. H. Sutter "The free lunch is over: A fundamental turn toward concurrency insoftware" in *Dr. Dobb's Journal*, 2005, pp. 1–9
18. S. Volkov, O. Sukhoroslov "A generic web service for running parameter sweep experiments in distributed computing environment" in *Procedia Computer Science* issue 66, 2015, pp. 477–486
19. S. Volkov, O. Sukhoroslov "Simplifying the use of clouds for scientific computing with Everest" in *Procedia Computer Science* issue 119, 2017, pp. 112–120
20. O. Zaikin, M. Manzyuk, S. Kochemazov, I. Bychkov, A. Semenov "A volunteer-computing-based grid architecture incorporating idle resources of computationalclusters" in *Numerical Analysis and ItsApplications*, I. Dimov, I. Farag´o, L. Vulkov ´Ed. Cham: Springer International Publishing, 2017, pp. 769–776

AUTHORS PROFILE

Alexander Rumyantsev received his Ph.D. from Petrozavodsk State University. He is now a senior researcher in the Institute of Applied Mathematical Research of the Karelian Research Centre of the Russian Academy of Sciences. His research interests include Stochastic Processes, Queueing Systems, High-Performance and Distributed Computing, Multi-Core and Many-Core Systems, and Energy Efficiency.

Oleg Sukhoroslov has received his PhD from the Institute for Systems Analysis of Russian Academy of Sciences in 2005. He is actively working in academia and industry as a researcher, software developer, expert and project manager. His research interests include distributed computing, middleware, web services, workflows and job scheduling. Oleg currently holds the position of senior researcher in the Centre for Distributed Computing at the Institute for Information Transmission Problems conducting research on software tools for building and automation of compute/data-intensive scientific workflows.

Anna Eparskaya is a Master student of Petrozavodsk State University actively working in the field of distributed computing and practicing R language development. She has received a scholarship of the Government of Russian Federation for her scientific advances.

Enrico Blanzieri received his Ph.D. from University of Turin, Italy. He is currently associate professor at the Department of Science and Information Engineering at the University of Trento, Italy. His research interests include machine learning, data mining and bioinformatics.

Valter Cavecchia holds an MSc degree in Physics from the University of Trento, Italy. He is working as research assistant at the Institute of Materials for Electronics and Magnetism of the National Research Council of Italy in Trento. His research interests include computer graphics algorithms/techniques and volunteer-based distributed computing.