Using Resources of Supercomputing Centers with Everest Platform

Sergey Smirnov, Oleg Sukhoroslov ✉ Vladimir Voloshinov

Institute for Information Transmission Problems of the Russian Academy of Sciences, Moscow, Russia
sasmir@gmail.com, sukhoroslov@iitp.ru, vv_voloshinov@iitp.ru

Abstract. High-performance computing plays an increasingly important role in modern science and technology. However, the lack of convenient interfaces and automation tools greatly complicates the widespread use of HPC resources among scientists. The paper presents an approach to solving these problems relying on Everest, a web-based distributed computing platform. The platform enables convenient access to HPC resources by means of domain-specific computational web services, development and execution of many-task applications, and pooling of multiple resources for running distributed computations. The paper describes the improvements that have been made to the platform based on the experience of integration with resources of supercomputing centers. The use of HPC resources via Everest is demonstrated on the example of loosely coupled many-task application for solving global optimization problems.

Keywords: High-performance computing · Clusters · Many-task applications · Distributed computing · Web services · Global optimization

1 Introduction

Computational methods are now widely used for solving complex scientific and engineering problems. These methods often require a large amount of computations and the use of high-performance computing (HPC) resources. Such resources can be provided by clusters operated on-premises, supercomputing centers, distributed computing infrastructures or clouds. Supercomputing centers represent an important source of HPC resources. In contrast to on-premises clusters, supercomputers have significantly more resources. However these resources are usually shared among many users and projects, which can lead to queues and high wait times. In contrast to distributed computing infrastructures, supercomputers support efficient execution of tightly coupled parallel applications. In comparison to public clouds, supercomputers use specialized hardware and are generally free for scientific projects.

The wide use of HPC resources among scientists is complicated due to a number of problems. There is a lack of convenient interfaces for running computations on such resources. Typically this procedure involves copying of input data, compilation of a program, preparing a job script, submission of a job via a

batch system, checking the job state and collecting the job results. All these steps should be performed via a command line environment. Such environments and batch systems are unfamiliar and too low-level for many researchers, requiring a considerable effort to master and use them instead of focusing on a problem being solved. This can demotivate researchers with less technical background. At the same time, there is a lack of tools for automation of routine activities that can be useful even for advanced users. Such tools are necessary for execution of single jobs, many-task applications such as parameter sweeps or complex workflows involving multiple jobs with dependencies. Also there is a need for tools supporting reliable execution of long-running computations or massive computations spanning multiple resources. In order to be efficient, such tools should take into account various characteristics and policies of HPC resources.

In this paper, we present an approach for solving the aforementioned problems using Everest [1, 18], a web-based distributed computing platform. A distinguishing feature of Everest is the ability to serve multiple distinct groups of users and projects by implementing the Platform as a Service (PaaS) cloud computing model. The platform is not tied to a single computing resource and allows the users to attach their resources and bind them to the applications hosted by the platform. These features make it possible to use the publicly available platform instance without having to install it on-premises. The use of Everest enables convenient access to HPC resources by means of domain-specific computational web services. It also allows one to seamlessly combine multiple resources of different types for running massive distributed computations.

The paper is organized as follows. Section 2 discusses the related work and compares it with the presented approach. Section 3 provides an overview of the Everest platform and its relevant features. Section 4 describes the integration of HPC resources with Everest including improvements of platform components. Section 5 demonstrates the use of Everest for running a loosely coupled many-task application for solving global optimization problems on HPC resources. Section 6 concludes the paper and discusses the future work.

2 Related Work

The use of web technologies for building convenient interfaces to HPC systems has been exploited since the emergence of the World Wide Web. For example [7] describes a user-friendly web interface for reconstruction of tomography data in a clinical environment backed by the Cray T3D massively parallel computer. In [6] authors describe a Web/Java based framework for remote submission of parallel applications on HPC clusters.

The emergence of grid computing [8] and the web portal technologies enabled the development of grid portals [21] and science gateways [5] facilitating the access to distributed computing resources and offering additional services such as collaborative capabilities. One of the early examples is the NPACI HotPage portal [20] consisted of a set of services for accessing the grid and individual HPC resources via a web browser. In [14] authors describe Gateway, a compu-

tational web portal system implementing a set of generic core services such as user management, security, job submission, job monitoring, file transfer, etc., that were used to build web interfaces for running applications from different domains on HPC systems. A number of similar frameworks for development of computational portals were proposed [26], which facilitated the development of many specialized portals in different domains of computational science.

While providing convenient web interfaces to HPC systems or specific computational packages, the first generation systems had the following drawbacks. First, it was difficult to combine multiple packages, run multi-step workflows and other many-task applications [15] typically found in science. This has lead to development of web based environments supporting the description and execution of user-defined workflows [10, 4, 3]. Second, the extension of portal functionality, e.g. publishing new applications, was difficult and limited to administrators only. While more recent systems [12, 4] introduced standard tools for application development, the deployment is still limited to privileged users. Third, classic portals were tightly coupled with particular HPC systems or a limited set of resources configured by administrators. Current science gateways support different types of resources ranging from supercomputers to clouds, but still not allow users to attach and access their own resources and accounts via the gateway.

In this paper we use Everest [18], a web-based distributed computing platform that addresses the mentioned drawbacks.

3 Everest Platform

Everest [18, 1] is a web-based distributed computing platform. It provides users with tools to publish and share computing applications as web services. The platform also manages the execution of applications on external computing resources attached by the users. In contrast to traditional distributed computing platforms, Everest implements the PaaS model by providing its functionality via remote interfaces. A single instance of the platform can be accessed by many users in order to create, run and share applications with each other.

Everest supports the development and execution of applications following a common model. An application has a number of *inputs* that constitute a valid request to the application and a number of *outputs* that constitute a result of computation corresponding to some request. Upon each request Everest creates a new *job* consisting of one or more *tasks* generated by the application from the job inputs. The tasks are executed by the platform on computing resources specified by a user. The dependencies between tasks are currently managed internally by the applications. The results of completed tasks are passed to the application and are used to produce the job outputs or new tasks if needed. The job is completed when all its tasks are completed. The described model is generic enough to support a wide range of computing applications.

Everest users can publish arbitrary applications with a command-line interface via the platform's web interface. Upon the invocation of the application the platform produces a single task corresponding to a specific command run.

It is possible to dynamically add new tasks or invoke other applications from a running application via the Everest API. This allows users to create and publish complex many-task applications with dependencies between tasks, such as workflows. Everest also includes a general-purpose application for running a large number of independent parametrized tasks such as parameter sweeps [23]. Each application is automatically published as a RESTful web service. This enables programmatic access to applications, integration with third-party tools and composition of applications into workflows. The platform also implements a web interface for running the applications via a web browser. The application owner can manage the list of users allowed to run the application.

Instead of using a dedicated computing infrastructure, Everest performs the execution of applications on external resources attached by users. The platform implements integration with standalone machines and clusters through a developed *agent* [16]. The agent runs on a resource and acts as a mediator between it and Everest enabling the platform to submit and manage tasks on the resource. The agent performs routine actions related to staging of input files, submitting a task, monitoring a task state and collecting the task results. The platform also supports integration with grid infrastructures [16], desktop grids [19] and clouds [24]. 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 [16]. In this case the platform performs dynamic scheduling of application tasks across the specified resource pool.

4 Integration of HPC Resources with Everest

In this section, we describe the improvements that have been made in Everest to support the efficient use of HPC resources via the platform. These developments are based on the experience of integration with several supercomputing centers in Russia including the Data Processing Center of NRC Kurchatov Institute (NRC KI) and the Supercomputer Simulation Laboratory of South Ural State University (SSL SUSU).

4.1 Agent Improvements

The integration of computing resource with Everest is achieved by means of the Everest agent [16]. The main functions of the agent are execution of tasks on the resource and provision of information about characteristics and current status of the resource. The agent supports interaction with various types of resource managers through the extensible adapter mechanism. The adapter receives generic resource requests and translates these requests into commands specific to a particular resource manager. There are three adapters have been developed for HPC clusters that support integration with TORQUE, Slurm and Sun Grid Engine.

In case of a cluster, the agent should by deployed by a user on the submission node. Since the agent has minimal dependencies and system requirements, it can be quickly deployed by the users without special skills and superuser privileges.

We have not found any serious problems when launching the agent on the used supercomputing centers. Basically, the agent was able to operate in environments with old Python versions and hard network restrictions.

However, while setting up the agent on the HPC4 supercomputer at NRC KI we have found that the agent can not submit more than 64 concurrent jobs. It was due to the limit on the maximum number of jobs per user imposed by the administrators via the Slurm manager. Thus the basic Slurm adapter running a single job per Everest task was not able to fully utilize the user quota on this system. In case of loosely coupled many-task applications with single-core tasks, such as described in Section 5, it was possible to utilize only 64 cores at once.

Two possible approaches to solving this problem were considered that relied on advanced features of Slurm: job arrays and complex many-task job scripts. Job arrays is a mechanism for managing collections of similar jobs in Slurm. It allows to submit and manage such jobs faster. However every job in the array is still treated by Slurm as a single job. So it does not allow to overcome the imposed jobs per user limit. Another option is to create complex Slurm jobs consisting of multiple tasks started with `srun` command inside a job script submitted with `sbatch`. Using this feature a set of Everest tasks can be submitted as a single Slurm job requesting resources needed to run all these tasks. In the job script a new Slurm task is created for every Everest task by calling the `srun` command.

The second approach was implemented in the new advanced Slurm adapter. The tasks received by the adapter from the agent are grouped by their Everest job ID and are accumulated. If a specified number of tasks belonging to the same job has accumulated, or no new tasks have arrived within the specified time, a complex job containing accumulated tasks is submitted to Slurm.

Below is an example of a job script generated by the adapter for two tasks:

```
#!/bin/bash
#SBATCH -D TASK_DIR/job1-2
#SBATCH -e srstderr
#SBATCH -o srstdout
#SBATCH -p hpc4-3d
#SBATCH -c 6
#SBATCH -n 2
( srun --exclusive -D TASK_DIR/job1-2 --output TASK_DIR/job1-2/stdout \
    --error TASK_DIR/job1-2/stderr -n1 -N1 bash TASK_DIR/job1-2/jobfile
  _ECODE=$?
  echo errorcode 0 $(ps -o etime= -p "$$") $_ECODE
  exit $_ECODE ) &
( srun --exclusive -D TASK_DIR/job1-1 --output TASK_DIR/job1-1/stdout \
    --error TASK_DIR/job1-1/stderr -n1 -N1 bash TASK_DIR/job1-1/jobfile
  _ECODE=$?
  echo errorcode 1 $(ps -o etime= -p "$$") $_ECODE
  exit $_ECODE ) &
wait
```

The `-exclusive` switch provides a uniform distribution of task processes to cluster nodes that have been allocated by Slurm for the job. To track the com-

pletion of a task, immediately after the `srun` call is completed, its return code is printed along with the sequence number of the task. The `srun` call is grouped with these operations using the parentheses operator, so that the group of commands is executed in a separate bash subshell. The ampersand operator allows tasks to be started concurrently while the `wait` call blocks further execution until all `srun` calls are completed.

Checking the state of tasks running inside a complex Slurm job is carried out in two stages. First, the adapter obtains the status of the job via the `scontrol` command. If the job is running, the state of individual tasks is checked by reading the `srstdout` file, where the job script prints the return codes of completed tasks. Thus the agent can process completed tasks before the whole job has completed.

While Everest supports the cancellation of jobs and tasks, in the described implementation it is difficult to stop the individual tasks. Therefore the cancellation is done for the entire Slurm job containing the tasks being canceled.

There are a number of other improvements that has been made in the agent and the Slurm adapter, such as automatic tasks directory cleanup, handling of job submission failures and propagation of environment variables.

The work on integrating the Tornado supercomputer [11] at SSL SUSU is ongoing right now. Because this system is quite overloaded, the calls to Slurm command line utilities can fail with a timeout. This is a new challenge because some rework of the agent's internals is needed to support such timeouts.

4.2 Supporting Advanced Resource Requirements

A significant limitation of Everest related to running parallel applications on HPC resources was the lack of explicit support for resource requirements. By default, all Everest tasks submitted via the agent on a resource were run as single-core jobs in a predefined queue. It was possible to change the number of CPU cores per task, but only on the level of the agent and for all tasks submitted via it. The execution of a parallel application with specific requirements for the number of nodes, processes per node and system resources required the development of an auxiliary script, which complicated publishing of such applications in Everest.

The aforementioned limitation was removed by enabling the Everest users to specify the application resource requirements. A number of new parameters were added to the configuration of Everest application such as the number of CPU cores per node (default is 1), the maximum number of CPU cores per node, the number of nodes (default is 1), the maximum number of nodes, the total amount of memory, the amount of memory per core and the wall clock time limit.

The application developer can specify the values of these parameters and optionally allow the users to override some of them when running the application. This approach is flexible enough to support the various use cases. The default values correspond to the previously supported case of a single-core task. Since for many applications the resource requirements depend on the input data, it is desirable to allow the users to tune these parameters according to their problems. A possible future improvement would be to allow application developers to provide performance models that can be used to automatically compute runtime

settings for a given problem. In cases where the problems have a fixed size or resources are limited, it makes sense to use only the predefined runtime settings.

The platform and the agent were modified to take into account the described resource requirements during the task scheduling and execution. The requirements are embedded in each task and are examined during the resource selection. In order to do this, the agent was modified to provide the additional information about the resource, such as number of nodes, cores and memory per node, available queues with their limits and states. Everest scheduler matches this information with the task requirements in order to select a suitable resource and a queue for the task execution. After the resource is selected, the final resource requirements of the task are computed and passed along with the task to the remote agent. The agent was modified to take into account these requirements during the submission of the task to a local resource manager. Currently, this functionality is implemented in the advanced Slurm adapter, which translates the task requirements to the corresponding Slurm options.

5 Experimental Evaluation Using Global Optimization

In this section, an application use case is presented which demonstrates the use of HPC resources via Everest for running loosely coupled many-task applications.

For many years the branch-and-bound (B&B) algorithm for discrete and global optimization has been considered as a challenge for parallel computing [2]. Most of known implementations for distributed computing environments follow the so called fine-grained approach and low-level parallelization. Here the parallel search tree traversal is coordinated by a master process delegating searching in the subtrees to a number of slave processes. To achieve a good load balancing the master performs dynamic redistribution of the work among slaves which implies intensive two-way data exchanges between the processes. The master also keeps track of a global incumbent value. This approach has drawbacks: the complexity of implementation; the need for low level communication between B&B solvers; usually it requires a homogeneous computing environment, e.g. cluster.

DDBNB (Domain Decomposition B&B) is an Everest application which implements an alternative approach based on a coarse-grained parallelism and preliminary decomposition of a feasible domain of the problem by some heuristic rules. Subproblems obtained via decomposition are solved by a pool of standalone B&B solvers running in parallel. The incumbent values found by each solver are intercepted and delivered to other solvers in order to speed up the traversal of B&B search tree. Current implementation of DDBNB[1] is based on SCIP and CBC open source solvers. Incumbent values exchange is based on a special messaging service implemented in Everest. More details can be found in [25, 17]. DDBNB is an example of loosely coupled many-task applications [15].

Initially DDBNB has been tested by solving a well known Travelling Salesman Problem formulated as a mixed-integer linear problem [25, 17]. The experiments

[1] https://github.com/distcomp/ddbnb

conducted in a heterogeneous distributed environment with standalone servers and virtual machines (up to 28 CPU cores in total) have demonstrated promising results. The described integration with HPC resources allowed to try DDBNB for solving hard global optimization problems requiring more computing resources.

Two well known problems in combinatorial geometry, which may be treated as global optimization problems, have been considered: the so called Tammes[2] and Thomson[3] problems. Both problems in their original form concern the arrangement of N points $(x_i, i{=}1{:}N)$ on a unit sphere in $\mathbb{R}^3$:
(Tammes) to maximize the minimal distance between any pair of points x_i, x_j

$$\begin{gathered} y \to \max_{y\in\mathbb{R}, x_i\in\mathbb{R}^3,\ i=1:N} \quad s.t.: \\ y \leqslant \|x_i - x_j\|, (1{\leqslant}i{<}j{\leqslant}N)\,, \|x_i\|{=}1\ \ (i{=}1{:}N)\,; \end{gathered}$$

(Thomson) to minimize electrostatic Coulomb energy of unit charges put in x_i:

$$\sum_{1\leqslant i<j\leqslant N} \frac{1}{\|x_i - x_j\|} \to \min_{x_i\in\mathbb{R}^3,\ i=1:N} \quad s.t.:\ \|x_i\|{=}1\ \ (i{=}1{:}N)\,.$$

Both problems are the well known challenges for computer science. It is hard to proof the global optimality of a given set of points $\{x_i,\ i{=}1{:}N\}$ even for small values of N, e.g. Tammes problem has computer-assisted proof for $N{\leqslant}14$ [13]. This proof substantially relies on the problem's specifics and is based on an enumeration of millions of the so called irreducible contact graphs.

DDBNB application enables one to try another, rather general approach. It is based on the implementation of the B&B algorithm for mathematical programming problems with polynomial in constraints and objective function in the SCIP solver [9, 22]. Both problems may be reduced to the proper form:
Tammes - minimize the maximum of scalar products $x_i^{\mathsf{T}} x_j$ for any $i, j{:}1{\leqslant}i{<}j{\leqslant}N$:

$$\begin{gathered} z \to \min_{x_i, z} \quad s.t.: \\ x_i^{\mathsf{T}} x_j {=} \sum_{d=1:3} x_{i,d} x_{j,d} \leqslant z, (1{\leqslant}i{<}j{\leqslant}N)\,; \\ \|x_i\|^2 = \sum_{d=1:3} (x_{i,d})^2 = 1, z{\in}\mathbb{R}, x_i{\in}\mathbb{R}^3,\ i{=}1{:}N. \end{gathered} \tag{1}$$

Thomson - via the auxiliary variables for the values of Coulomb energy:

$$\begin{gathered} \sum_{i,j=1,i<j}^{N} z_{ij} \to \min_{x_i, z_{ij}} \quad s.t.: \\ z_{ij}^2 \,(x_i - x_j)^{\mathsf{T}} (x_i - x_j) = 1\ \ (1{\leqslant}i{<}j{\leqslant}N)\,; \\ x_i^{\mathsf{T}} x_i = 1\ \ (i{=}1{:}N)\,,\ x_i \in \mathbb{R}^3, z_{ij}{\in}\mathbb{R}. \end{gathered} \tag{2}$$

It is worth to mention that in addition to major constraints, some auxiliary constraints have been added to reduce the number of redundant solutions that

[2] http://neilsloane.com/packings/
[3] http://www-wales.ch.cam.ac.uk/∼wales/CCD/Thomson/table.html

might be obtained by 3D rotations, mirror transformations and renumbering of points. For example, the first point is always fixed as $x_1{=}(0,0,1)$.

The first attempts to solve the Tammes problem by a standalone single-threaded SCIP process were failed even for $N{=}8$. The solver running with default settings quickly occupies a lot of memory (29 GB in 45 minutes) to store the search tree data. By that time, the difference between the lower bound and the incumbent value was more than 90%, i.e. the B&B was far from completion.

The further optimizations are based on the understanding of how SCIP handles non-convex polynomial constraints [9, 22]. Because any polynomials may be converted to a sum of bilinear summands by introducing additional variables and constraints, it is enough to explain how SCIP handles a bilinear function in constraints. SCIP uses the so called McCormik envelopes which give convex lower bound and concave upper bound (both are piecewise linear) for a bilinear function on a rectangle (Fig. 1).

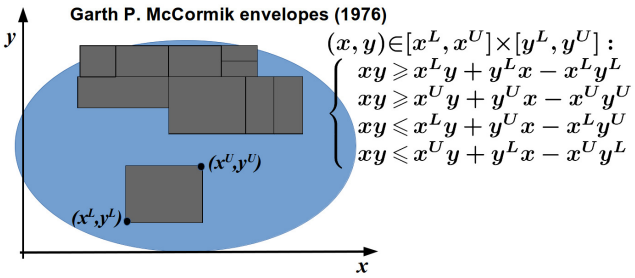

Fig. 1. McCormik convex lower bound and concave upper bound for bilinear function.

The smaller the diameter of the rectangle the more is the accuracy of convex approximations and the better is the lower bound for global optimum value given by solving of relaxed convex subproblems. Thus the greater the number of "small" rectangles, the higher the accuracy of the B&B algorithm. As a rough approximation, assume that the memory consumption is proportional to the current number of rectangles overlapping the feasible domain in problems (1) and (2). Assume also that the number of rectangles is proportional to the multi-dimension volume of the domain. Let's divide the domain, e.g. into two equal subdomains with twice as less volume (Fig. 2), and solve these subproblems in parallel by using different SCIP processes and exchanging incumbents between the solvers. One can hope that in this case each solver process will require twice as less memory. This approach allows to solve the problem in parallel while reducing the system requirements of individual tasks.

Consider a possible simple decomposition into subdomains with equal volumes. Let's $K{\leqslant}(N{-}1)$, take vectors $x_k, k{=}2{:}K{+}1$ (beginning from 2, because x_1 is fixed to $(0,0,1)$) and perform decomposition by their first coordinate's sign. Formally, for any K-set of parameters $\{p_k{=}{\pm}1, k{=}1{:}K\}$ the following inequal-

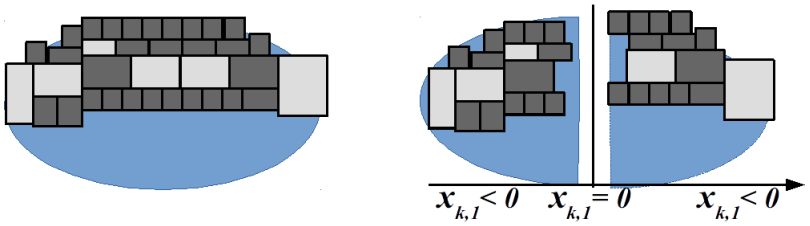

Fig. 2. Decomposition reasoning for global optimization.

ity constraints should be added to problems (1) or (2): $p_k \cdot x_{k+1,1} \leqslant 0, (k = 1{:}K)$. Thus the original problem can be subdivided into 2^K subproblems.

We have run several experiments for solving the Tammes problem using the DDBNB application and the HPC4 cluster at NRC KI. In the experiment with N=8 and K=7 the problem has been decomposed into 128 tasks that were completed within 100 minutes. The task execution trace for this experiment is presented on Fig. 3. Note the large imbalance in the task run times that is due to the chosen decomposition approach. We plan to address this issue in the future by using other decomposition methods producing much more subproblems that may be more balanced. Preliminary experiments were conducted for the Tammes problem with N=9 and K=8. In this case 256 subproblems were generated and the computations were successfully completed in about three days.

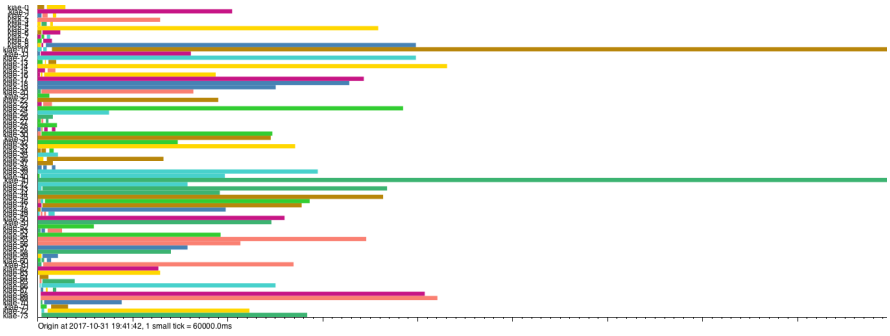

Fig. 3. Trace of solving the Tammes problem N=8 at HPC4 cluster at NRC KI.

6 Conclusion and Future Work

In this paper we have presented the integration of Everest platform with HPC resources of supercomputing centers. The described implementation has been tested by solving hard global optimization problems with the DDBNB Everest

application. The use of HPC resources with DDBNB has provided the significant increment in computing power available for our experiments. A number of improvements has been made in Everest and its agent in order to support the efficient use of HPC resources and overcome the limitations found during the early experiments. The presented approach enables convenient access to HPC resources, execution of many-task applications, and pooling of multiple resources for running distributed computations.

Future work will focus on integration with other supercomputing centers and improving the described implementation. We also plan to conduct the large-scale experiments involving resources of multiple centers via Everest. Regarding the DDBNB application, we plan to investigate the use of alternative decomposition rules producing thousands of subtasks for solving the considered problems.

Acknowledgments

This work is supported by the Russian Science Foundation (project No. 16-11-10352). This work has been carried out using computing resources of the federal collective usage center Complex for Simulation and Data Processing for Mega-science Facilities at NRC "Kurchatov Institute" (ministry subvention under agreement RFMEFI62117X0016), http://ckp.nrcki.ru/.

References

1. Everest. [online], http://everest.distcomp.org/
2. Parallel combinatorial optimization. Ed. El-Ghazali Talbi, vol. 58. John Wiley & Sons (2006)
3. Afanasiev, A., Sukhoroslov, O., Voloshinov, V.: MathCloud: Publication and reuse of scientific applications as restful web services. In: Parallel Computing Technologies, pp. 394–408. Springer (2013)
4. Afgan, E., Goecks, J., Baker, D., Coraor, N., Nekrutenko, A., Taylor, J.: Galaxy: A gateway to tools in e-science. In: Guide to e-Science, pp. 145–177. Springer (2011)
5. Allan, R.N.: Virtual research environments: From portals to science gateways. Elsevier (2009)
6. Chen, Z., Maly, K., Mehrotra, P., Vangala, P.K., Zubair, M.: Web-based framework for distributed computing. Concurrency - Practice and Experience 9(11), 1175–1180 (1997)
7. Formiconi, A., Passeri, A., Guelfi, M., Masoni, M., Pupi, A., Meldolesi, U., Malfetti, P., Calori, L., Guidazzoli, A.: World wide web interface for advanced spect reconstruction algorithms implemented on a remote massively parallel computer. International journal of medical informatics 47(1-2), 125–138 (1997)
8. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a new computing infrastructure. Elsevier (2003)
9. Gleixner, A., Eifler, L., Gally, T., Gamrath, G., Gemander, P., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Miltenberger, M., et al.: The SCIP optimization suite 5.0. Tech. Rep. 17-61, ZIB, Takustr.7, 14195 Berlin (2017)
10. Kacsuk, P.: P-grade portal family for grid infrastructures. Concurrency and Computation: Practice and Experience 23(3), 235–245 (2011)

11. Kostenetskiy, P., Safonov, A.: SUSU supercomputer resources. In: Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). Arkhangelsk, Russia. vol. 1576, pp. 561–573 (2016)
12. McLennan, M., Kennell, R.: Hubzero: a platform for dissemination and collaboration in computational science and engineering. Computing in Science & Engineering 12(2), 48–53 (2010)
13. Musin, O.R., Tarasov, A.S.: The Tammes problem for N=14. Experimental Mathematics 24(4), 460–468 (2015)
14. Pierce, M.E., Youn, C., Fox, G.C.: The gateway computational web portal. Concurrency and Computation: Practice and Experience 14(13-15), 1411–1426 (2002)
15. Raicu, I., Zhang, Z., Wilde, M., Foster, I., Beckman, P., Iskra, K., Clifford, B.: Toward loosely coupled programming on petascale systems. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. p. 22. IEEE Press (2008)
16. Smirnov, S., Sukhoroslov, O., Volkov, S.: Integration and combined use of distributed computing resources with Everest. Procedia Computer Science 101, 359–368 (2016)
17. Smirnov, S., Voloshinov, V.: Implementation of concurrent parallelization of branch-and-bound algorithm in Everest distributed environment. Procedia Computer Science 119, 83–89 (2017)
18. Sukhoroslov, O., Volkov, S., Afanasiev, A.: A web-based platform for publication and distributed execution of computing applications. In: Parallel and Distributed Computing (ISPDC), 2015 14th International Symposium on. pp. 175–184 (June 2015)
19. Sukhoroslov, O.: Integration of Everest platform with BOINC-based desktop grids (2017)
20. Thomas, M., Mock, S., Boisseau, J.: Development of web toolkits for computational science portals: The npaci hotpage. In: High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on. pp. 308–309. IEEE (2000)
21. Thomas, M., Burruss, J., Cinquini, L., Fox, G., Gannon, D., Gilbert, L., Von Laszewski, G., Jackson, K., Middleton, D., Moore, R., et al.: Grid portal architectures for scientific applications. In: Journal of Physics: Conference Series. vol. 16, p. 596. IOP Publishing (2005)
22. Vigerske, S., Gleixner, A.: SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. Optimization Methods and Software pp. 1–31 (2017)
23. Volkov, S., Sukhoroslov, O.: A generic web service for running parameter sweep experiments in distributed computing environment. Procedia Computer Science 66, 477–486 (2015)
24. Volkov, S., Sukhoroslov, O.: Simplifying the use of clouds for scientific computing with Everest. Procedia Computer Science 119, 112–120 (2017)
25. Voloshinov, V., Smirnov, S., Sukhoroslov, O.: Implementation and use of coarse-grained parallel branch-and-bound in Everest distributed environment. Procedia Computer Science 108, 1532–1541 (2017)
26. Yang, X., Martin, T., Mark, H., Mark, C., Ligang, H., Peter, M.: Survey of major tools and technologies for grid-enabled portal development. In: Proc. UK e-Science All Hands Meeting (NeSC 06), University of Cambridge Press (2006)