Running Many-Task Applications Across Multiple Resources with Everest Platform

Oleg Sukhoroslov(✉), Vladimir Voloshinov, and Sergey Smirnov

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

Abstract. Distributed computing systems are widely used for the execution of loosely coupled many-task applications, such as parameter sweeps, workflows, distributed optimization. These applications consist of a potentially large number of computational tasks that can be executed more or less independently. Since the application users often have an access to multiple computing resources, it is important to provide a convenient and efficient environment for execution of applications across the user-defined heterogeneous resource pools. The paper discusses the related challenges and presents an approach for solving them based on Everest, a web-based distributed computing platform. The presented solution supports reliable and efficient execution of many-task applications, while taking into account resource performance, adapting to queuing delays and providing a mechanism for communication between tasks.

Keywords: Distributed computing · Many-task applications · Parameter sweep · Resource pool · Scheduling · Messaging

1 Introduction

Many-task applications [13] are loosely-coupled parallel applications consisting of potentially large number of computational tasks that can be executed more or less independently. Multiple classes of such applications are widely used in science and technology. Bag-of-tasks applications [6], such as parameter sweeps, Monte Carlo simulations, image rendering, have no dependencies between the tasks. Workflows [24], which are used for automation of complex computational and data processing pipelines, consist of multiple tasks with control or data dependencies between them. There is also a special class of many-task applications that require cooperation between the running tasks. For example, the distributed implementation of the branch-and-bound method [15] requires the exchange of incumbent values between the tasks concurrently processing different subtrees.

While many-task applications are naturally suited for execution on distributed computing resources, there exists a number of challenges related to the

V. Voevodin and S. Sobolev (Eds.): RuSCDays 2020, CCIS 1331, pp. 634–646, 2020.
https://doi.org/10.1007/978-3-030-64616-5_54

efficient execution of such applications such as management of a large number of tasks, accounting for dependencies between tasks, implementing coordination and data exchange, use of multiple independent computing resources, task scheduling, accounting for local resource policies and dealing with failures. Some of these challenges were addressed by existing solutions such as meta-schedulers [9,25], user-level frameworks [5,12], workflow management systems [24], grid portals [10,26] and science gateways [3,11]. However, there is still a lack of convenient tools and environments that do not require a considerable effort to master and use them, thereby allowing users to focus on problems being solved.

In particular, this paper considers the following use case. A user has accounts on multiple computing resources, such as an institution cluster and supercomputing centers. The user wants to run some many-task application by leveraging all available resources in a most efficient manner to obtain the results as quickly as possible. The desired solution should support reliable execution of long-running computations spanning multiple resources, while taking into account resource characteristics and policies. In addition, it should not require complex deployment and configuration, while allowing the users to adapt and run their applications with a minimal effort via a convenient user interface. However, there is a lack of solutions that meet all these requirements, which hinders their adoption by users with less technical background in distributed computing.

In this paper, we present an approach for addressing the aforementioned issues and requirements using Everest [1,20], a web-based distributed computing platform. Everest implements the Platform as a Service (PaaS) model by providing its functionality via remote user and programming interfaces. The platform allows the users to attach their computing resources, publish applications and run them on arbitrary combinations of resources. These features enable Everest to serve multiple distinct groups of users while satisfying the above requirements.

In particular, the paper describes the recent platform improvements and features related to the use of multiple resources, scheduling and execution of many-task applications. The major contributions are the resource pool mechanism (Sect. 2), task scheduling improvements (Sect. 3) and a generic mechanism for communication between tasks (Sect. 5). An experimental evaluation of presented solutions on several application cases is also included (Sect. 6).

2 Many-Task Applications and Resource Pools

Everest provides several tools for execution of many-tasks applications. A general-purpose service for execution of bag-of-tasks applications [27] enables users to describe parametrized computations using a declarative format. Execution of workflows is supported by external orchestration using a scripting language or via a general-purpose service integrated into the platform [22]. Finally, the low level programming interface enables platform clients to dynamically manage a set of tasks within a job by sending commands and receiving notifications [23].

Instead of using a dedicated computing infrastructure, Everest performs the execution of application tasks 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 platform also supports integration with grid infrastructures [16], desktop grids [21] and clouds [28].

Everest users can flexibly bind their resources to applications. In particular, a user can specify multiple resources, possibly of different type, for running an application, which is especially important for many-tasks applications. In this case, the platform performs a dynamic scheduling of application tasks across the specified resources However, when submitting such jobs, the user had to manually form a list of used resources. In addition, there was no way to pass the user preferences for the use of individual resources. This complicated the use of Everest for running applications on multiple resources.

To solve the aforementioned problems, a concept of resource pool has been introduced and implemented in Everest. A resource pool is a virtual resource comprised of several regular resources. The user can configure the set of resources included in the pool, as well as the priorities of individual resources. The priorities may correspond to user preferences, relative performance or cost of the corresponding resources. These priorities are used by the platform during the task scheduling as described in Sect. 3. The user can also temporary forbid running tasks on a resource by setting its priority to zero. The changes of pool configuration made during the application execution are automatically picked up by the scheduler allowing the user to manually control the execution.

The resource pool can be used in the same way as regular resources when starting jobs and configuring applications. The pool owner can also configure a list of users and groups that are allowed to use the pool. This enabled Everest users to flexibly configure and conveniently use personal and collective computing infrastructures consisting of several resources for their computations.

3 Task Scheduling

Everest implements a flexible multi-level approach for scheduling jobs, corresponding to application runs, and individual tasks with jobs. The global *job scheduler*, which is periodically invoked with information about current jobs and resource states, fairly distributes the available resource slots among the jobs. Each job encapsulates an *application-level task scheduler*, which is invoked to select the tasks for running on the resources offered by the job scheduler. Finally, a complex resource, such as the one representing a grid or a cloud infrastructure, implements an internal *resource-level task scheduler* that is used to distribute the tasks assigned to the resource among a pool of dynamically allocated machines.

The same approach has been used to implement the scheduling of tasks assigned to resource pools. If a pool is specified as a resource for running a job, then the platform distributes the job tasks among the pool resources using the

newly developed resource-level scheduler associated with the pool. By default, this scheduler assigns the tasks to resources with idle slots in the order of user-defined priorities specified in the pool configuration.

However, the relative resource performance often depends on an application, and the provided priorities may not allow for efficient task distribution. Also, for non-dedicated resources shared among many users, such as supercomputers, the presence of idle slots does not guarantee that the scheduled task will start immediately. In practice, the tasks can be arbitrary delayed in the resource queues and blocked by the pending reservations, which complicates the task scheduling and may negatively impact the application execution time. These issues are addressed by the following task scheduling improvements.

The relative performance of resources for a given many-task application can be taken into account in runtime by collecting the execution times of completed tasks. Since these tasks are different, this is not a real benchmarking. However, many-task applications are often comprised of one or several groups of tasks with similar characteristics, which justifies the use of this approach. The mean of execution times of completed tasks on a given resource is used as an estimate of a task execution time during the task scheduling. The mean is computed on a subset of recent measurements (10 latest values currently) to better account for the applications that consist of several "waves" of tasks, which characteristics differ between the waves, and for the variation of resource performance in time.

Another possible approach is to run an application-specific benchmark on all resources prior to the application execution and use the results to compute the resource priorities. However this approach requires more time and effort from the user, while not solving completely the problem of task execution time estimation and not taking into account the variation of resource performance in time. A more promising approach, which can be used to improve the current implementation, is to use the information on task execution collected during the previous application runs, as was previously demonstrated for workflows in [22].

To account for the wait times in resource queues, a similar approach is used to estimate the task wait time on a given resource during the scheduling. The wait times of completed tasks are collected during the application execution. Note that this information is not specific for a given application and is collected and shared across all jobs using the particular resource. A mean of recent measurements (10 latest values currently) is used as an estimate of a task wait time on a given resource during the task scheduling.

The described estimates for task execution and wait times are used during the task scheduling to compute the estimated task finish time for each resource in the pool. A task is assigned to a resource that provides the earliest task finish time. In the beginning of application execution, when these estimates are not available, the user-specified resource priorities are used as described above.

The presented approach allows to take into account the heterogeneous and dynamic characteristics of resources within a pool by using the available information and adapting to changes. However, after being scheduled, a task may still stuck in the resource queue or run extremely slowly on a malfunctioning

machine, thereby severely delaying the application finish time. The so called "long tail" or "stragglers" problem is often observed for many-task applications, when the finish time is determined by the "slow" resources computing the several remaining tasks. While the previously described approach can partially alleviate this problem, it cannot eliminate it completely due to the dynamic nature of the execution environment. Therefore an additional mechanism is needed to reschedule the stuck tasks if it will improve the application finish time.

To address this issue, a task migration mechanism has been implemented inside the pool task scheduler. It periodically checks all tasks in a pool and cancels a task if it is waiting on a resource and there is another idle resource with a better estimate of the task finish time. The cancelled tasks are then rescheduled using the previously described approach. The migration of already running tasks is not enabled by default since it can lead to a bouncing of long-running tasks between resources. A user can optionally enable it when running an application with uniformly sized tasks, to protect against the slowly running machines. Besides, the task execution and wait time statistics are updated for the original resource if the cancelled task experienced longer wait or execution times than was expected, to better account for a sudden resource degradation. Finally, in the case of resource failure, Everest performs the rescheduling of tasks assigned to the resource, which can be considered as a special migration case.

The adaptive task scheduling and migration have been previously studied and implemented in the context of grid computing [4,9]. However, while the previous work was mostly focused on detecting the performance degradation and on opportunistic migration, the presented approach takes into account the resource queuing delays and leverages the recent execution history. Another approach widely used in grids is "pilot jobs" [14], which dynamically deploys the execution agents on the resource nodes as regular jobs and then directly assigns tasks to the agents bypassing the resource queues. While this approach allows to avoid the queuing delays, it requires a direct connection from the resource nodes to the external scheduler, which is often restricted on shared HPC resources and complicates the deployment. Also, this approach has an inevitable trade-off between stopping an idle agent and wasting the occupied resources. Another remedy for varying queuing delays is to simultaneously submit the same task to multiple resources [7], however this would complicate the task dispatching and introduce an additional load on the resource queues.

4 Supporting Communication Between Tasks

As described in Sect. 2, Everest supports execution of different types of many-task applications. In many cases, such as parameter sweep experiments, these tasks are executed completely independently of each other. In the case of workflows, the dependencies between the tasks correspond only to the use of the results of one task when starting another task. Therefore, there is no communication between the running tasks in the aforementioned cases.

However, there exist important cases of many-task applications that require coordination and data exchange between the tasks in the process of their execution. These are not only traditional tightly coupled parallel applications, such as MPI programs, but also loosely coupled applications, which permit non-simultaneous execution of tasks and have moderate communication requirements. The former applications are suited for running inside a single HPC system and their support in Everest has been described in previous work [17]. Here we focus on the latter applications that allow execution across multiple resources.

A typical example of a loosely coupled many-task application is a parallel version of the branch and bound method for solving optimization problems, in which the tasks process different subtrees of the whole search tree. In this case, it is necessary to implement the exchange of the best found solutions (incubment values) between the tasks to speed up the search process. Note that the tasks need not to be started and executed simultaneously, they can start upon the resource availability and can also fail independently. However, upon the startup a task should be able to obtain the current best incumbent value. This calls for a reliable storage of such values independent of the tasks.

The required interaction between the tasks can be implemented by the application developer in ad-hoc manner. However, this involves a solution of a number of difficult technical problems. For example, when the tasks are executed on multiple resources, it may not be possible to establish a direct connection between the tasks. Also, as was exemplified before, it may be necessary to reliably store and forward the previously transmitted data to the newly started tasks. Solving such problems requires the implementation and deployment of a rather complex separate component for organizing the data exchange between the tasks.

To simplify the implementation of the considered class of applications on Everest, a general-purpose mechanism for communication between the tasks was implemented at the platform level. The implemented mechanism is based on a two-way message exchange between the tasks and the platform through the Everest agent running on the resource and managing local tasks. When the task starts, the agent passes through the environment variable the port number of the local socket, by connecting to which the task can send and receive messages. The agent forwards the messages received from the task including the task identifier to the platform via the WebSocket connection. Similarly, the platform can send the messages back to the running tasks through the corresponding agents.

Using the messaging support, the following lightweight model for communication between tasks based on shared variables is implemented. Tasks can write and read the values of arbitrary named variables by sending special messages *VAR_SET* and *VAR_GET*. The current values of the variables are stored on the platform side and represent an analogue of the shared memory available to all tasks within a job. The variables belonging to different jobs are stored separately, such that the tasks of different jobs cannot share the data. When the value of a variable is changed, the new value is sent to all tasks of the job. Thus, this mechanism combines the shared memory and publish-subscribe models.

In addition to the basic operations for reading and writing the values of the variables, the conditional write operations *VAR_SET_MI* and *VAR_SET_MD* are implemented. These operations atomically change the value of the variable only if the passed value is greater or less than the current one. These operations, guaranteeing a monotonous increase or decrease of the variable values in the presence of concurrent updates, are used to exchange the incumbent values between the tasks in a distributed implementation of the branch and bound method [15]. In the future, the set of supported operations can be expanded. For example, an atomic compare and swap operation can be implemented in a similar way.

5 Experimental Evaluation and Applications

In this section we present experiments performed to evaluate the described results and demonstrate the use of Everest for running many-task applications across a pool of multiple HPC resources.

The following resources were used in the experiments:

- **HSE:** Supercomputing complex of NRU Higher School of Economics,
- **HPC4:** HPC-4 cluster at NRC Kurchatov Institute,
- **Lomonosov:** Lomonosov-1 supercomputer at Lomonosov Moscow State University,
- **Govorun:** Govorun supercomputer at Joint Institute for Nuclear Research.

5.1 Parameter Sweep Applications

The computational workload was represented by two real-world parameter sweep applications consisting of a large number of independent tasks. The first application is from the life sciences domain and represents the virtual screening of 1000 ligand molecules against the same protein using the molecular docking program Autodock Vina. The second application is from the geophysics domain and consists of 670 tasks performing tabulation of a complicated multidimensional function for solving an inverse problem. Both applications were run via the generic Parameter Sweep service [27] implemented on Everest. The experiments for each application were run in succession with a minimal interval between the runs in order to have the similar conditions on the resources. The number of simultaneously running tasks on each resource was limited as follows: HSE - 88, HPC4 - 64, Lomonosov - 32, Govorun - 32.

Figure 1 (top) displays the number of running tasks across the used resources during the execution of the virtual screening application without using the new resource pool functionality and task scheduler. The platform managed to reach the specified limits on allocated resources given their availability. The periodic drops of utilization are due to the completion of task batches and the queuing delays. However, it can be seen that the application run time suffers from the mentioned “tail problem” - the last batch of tasks was processed very slowly on Govorun. Actually this resource was the fastest one in terms of the task

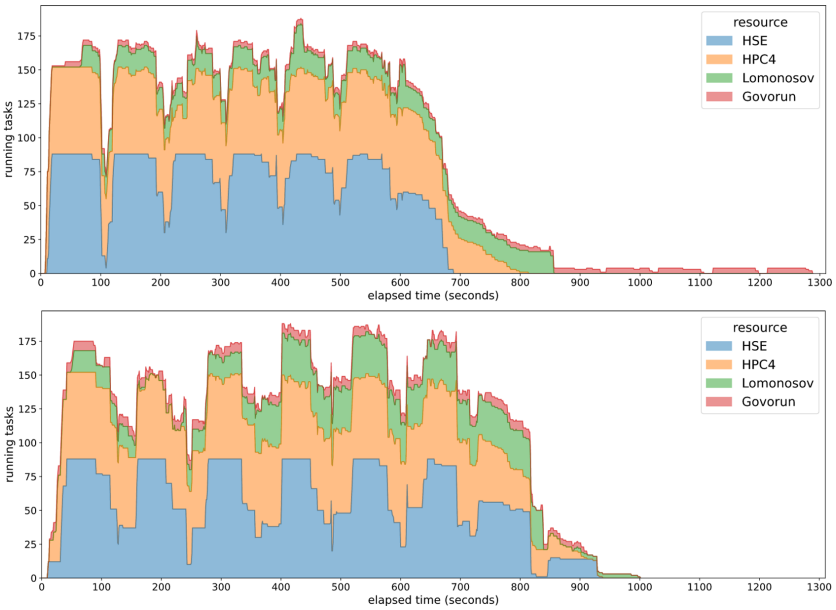

Fig. 1. Executions of the virtual screening application (top - no optimizations, bottom - with optimizations)

execution time, but the task wait times were very high in comparison to the other resources, which explains the observed slow progress.

Figure 1 (bottom) displays the execution of the same application using the new resource pool scheduler. The "tail" has been largely eliminated by both taking into account the collected task wait and execution times during the task scheduling and by migrating 14 tasks from Govorun to HSE (the second fastest resource) in the end of the computations. These optimizations reduced the application execution time from 1288 to 1001 s.

The execution of the geophysics application (see Fig. 2) has been similarly improved. In this case, the "tail" was caused by several factors - the long processing of a batch of tasks by the slowest resource (HPC4) and the high wait times on Govorun and Lomonosov. The improved task scheduling and migration of tasks from these resources to HSE helped to eliminate the problem and reduce the application execution time from 2987 to 2514 s.

The following subsections present other application use cases that are currently leveraging the described implementation to perform computations on the pools of HPC resources.

5.2 Coarse-Grained Parallelization of Branch-and-Bound

The first use case, where the integration of muliple resources via Everest can help to solve the hard optimization problems, concerns the so called coarse-grained

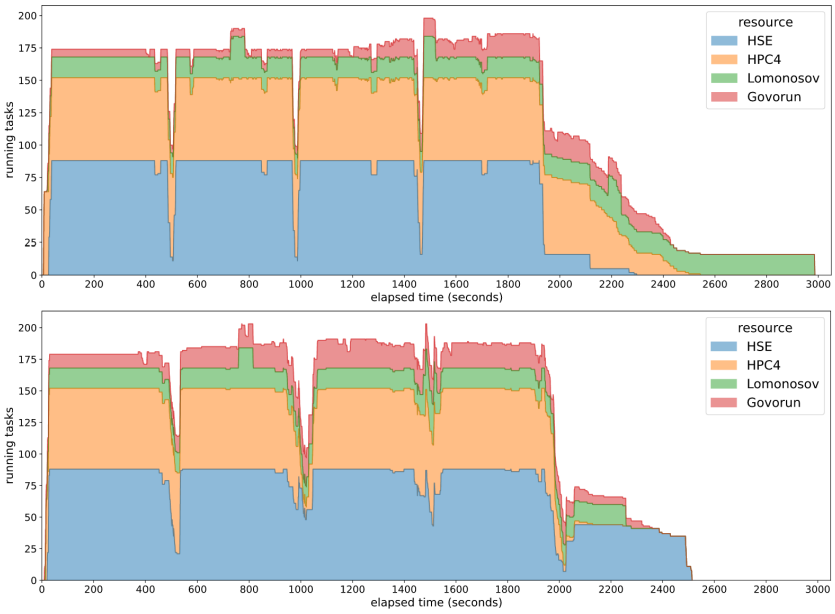

Fig. 2. Executions of the geophysics application (top - no optimizations, bottom - with optimizations)

parallelization of the branch-and-bound (BnB) algorithm [15,18,29]. Here the original problem is decomposed into sub-problems by some decomposition of its feasible domain. Then these sub-problems can be solved in parallel by a pool of existing general-purpose BnB-solvers running on multiple machines and exchanging the incumbent values they found via the communication mechanism described in Sect. 4. In addition to the decomposition, the concurrent approach can be used where the same sub-problems are being solved by multiple BnB-solvers with different algorithm options [18]. The developed Everest application *DDBNB*[1] (Domian Decomposition BnB) enables to combine the "decomposition" and "concurrent" modes of operation. The current implementation is based on SCIP solver [8].

Figure 3 displays the execution of tasks during a *DDBNB* run on two resources for solving the following instance of the Traveling Salesman Problem (TSP) from the TSPLIB[2] collection: *ch150* (150 cities), 64 subsets of the feasible domain and 7 sets of the solver options (448 tasks in total). The used limits on the number of simultaneously running tasks: HSE - 88 tasks, HPC4 - 48 tasks. On the top graph, each task corresponds to a horizontal line from the task submission time to the task completion time. The tasks are colored according to the used resource, the intense color corresponds to the actual task

[1] https://everest.distcomp.org/apps/ssmir/DDBNB.
[2] http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp.

execution. It can be seen that the task execution times vary greatly due to the different complexity of the corresponding sub-problems.

5.3 Balanced Identification of Mathematical Models

The second use case relates to the method of balanced identification with regularization of mathematical models by experimental datasets. This method is also called the SvF-technology [19] and is based on the bi-level optimization with a set of independent mathematical programming problems to be solved at the lower level. All these problems (can be hundreds in practice) can be solved by the general-purpose solvers in parallel, and Everest provides the capabilities to do this on multiple resources. Currently the following solvers are supported: IPOPT [30], to find a local optimum in nonlinear optimization problems; SCIP if a global optimum is needed. The current implementation of SvF-technology is based on the Everest application *SSOP*[3] (Solve Set of Optimization Problems), which solves a batch of independent problems in parallel by using the available computing resources.

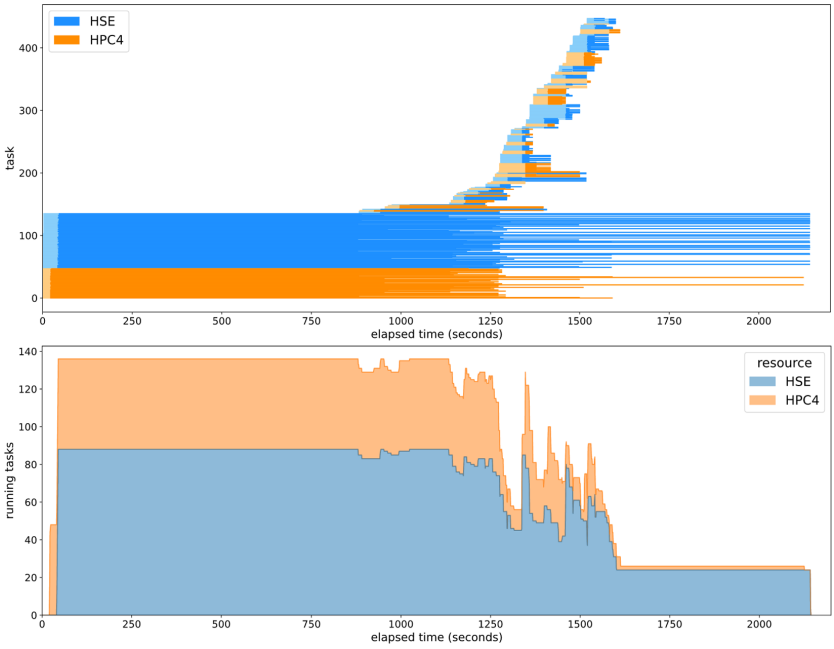

Fig. 3. Execution of *DDBNB* application for solving a TSP instance on HSE and HPC4 clusters (top - task executions in time, bottom - number of running tasks)

[3] https://optmod.distcomp.org/apps/vladimirv/solve-set-opt-probs.

6 Conclusion and Future Work

The paper presented a ready-to-use solution for performing computations across the user-defined resource pools consisting of separate HPC resources. The presented solution supports reliable and efficient execution of many-task applications, while taking into account resource performance, adapting to queuing delays and providing a mechanism for communication between tasks. In contrast to related work, the presented solution is built on a web-based platform and does not require complex deployment and configuration, while allowing the users to adapt and run their applications with a minimal effort via a convenient user interface. The public instance of the platform [1] is available online for all interested users. Future work will focus on improving and extending the presented functionality. We also plan to conduct and report extended large-scale experiments for the mentioned application use cases.

Acknowledgments. This work is supported by the Russian Science Foundation (Project 16-11-10352 - Sects. 4 and 5.2) and the Russian Foundation for Basic Research (Project 20-07-00701 - Sect. 5.3, Project 18-07-00956 - all other sections). This research was supported in part through resources of supercomputer facilities provided by NRU HSE. 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". The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University. Computations were held on the basis of the HybriLIT heterogeneous computing platform (LIT, JINR) [2].

References

1. Everest. http://everest.distcomp.org/
2. Adam, G., et al.: IT-ecosystem of the HybriLIT heterogeneous platform for high-performance computing and training of IT-specialists. Distributed Computing and Grid-technologies in Science and Education. **2267**, 638–644 (2018)
3. Afgan, E., Goecks, J., Baker, D., Coraor, N., Nekrutenko, A., Taylor, J.: Galaxy: a gateway to tools in e-science. In: Yang, X., Wang, L., Jie, W. (eds.) Guide to e-Science, pp. 145–177. Springer, London (2011)
4. Allen, G., Angulo, D., Foster, I., et al.: The Cactus Worm: experiments with dynamic resource discovery and allocation in a grid environment. Int. J. High Perform. Comput. Appl. **15**(4), 345–358 (2001)
5. Casanova, H., Berman, F., Obertelli, G., Wolski, R.: The AppLeS parameter sweep template: User-level middleware for the grid. In: SC 2000: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing, pp. 60–60. IEEE (2000)
6. Cirne, W., Brasileiro, F., Sauve, J., et al.: Grid computing for bag of tasks applications. In: Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment. Citeseer (2003)
7. Dean, J., Barroso, L.A.: The tail at scale. Commun. ACM **56**(2), 74–80 (2013)
8. Gamrath, G., Anderson, D., Bestuzheva, K., et al.: The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020

9. Huedo, E., Montero, R.S., Llorente, I.M.: A framework for adaptive execution in grids. Softw. Pract. Exp. **34**(7), 631–651 (2004)
10. Kacsuk, P.: P-GRADE portal family for grid infrastructures. Concurrency Comput. Practice Exp. **23**(3), 235–245 (2011)
11. McLennan, M., Kennell, R.: HUBzero: a platform for dissemination and collaboration in computational science and engineering. Comput. Sci. Eng. **12**(2), 48–53 (2010)
12. Moscicki, J.T.: Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data. In: 2003 IEEE Nuclear Science Symposium. Conference Record (IEEE Cat. No. 03CH37515), vol. 3, pp. 1617–1620. IEEE (2003)
13. Raicu, I., Foster, I.T., Zhao, Y.: Many-task computing for grids and supercomputers. In: 2008 Workshop on Many-Task Computing on Grids and Supercomputers, pp. 1–11. IEEE (2008)
14. Sfiligoi, I., Bradley, D.C., Holzman, B., Mhashilkar, P., Padhi, S., Wurthwein, F.: The pilot way to grid resources using glideinWMS. In: 2009 WRI World Congress on Computer Science and Information Engineering, vol. 2, pp. 428–432. IEEE (2009)
15. Smirnov, S., Voloshinov, V.: On domain decomposition strategies to parallelize branch-and-bound method for global optimization in Everest distributed environment. Procedia Comput. Sci. **136**, 128–135 (2018)
16. Smirnov, S., Sukhoroslov, O., Volkov, S.: Integration and combined use of distributed computing resources with Everest. Procedia Comput. Sci. **101**, 359–368 (2016)
17. Smirnov, S., Sukhoroslov, O., Voloshinov, V.: Using resources of supercomputing centers with everest platform. In: Voevodin, V., Sobolev, S. (eds.) RuSCDays 2018. CCIS, vol. 965, pp. 687–698. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05807-4_59
18. Smirnov, S., Voloshinov, V.: Implementation of concurrent parallelization of branch-and-bound algorithm in Everest distributed environment. Procedia Comput. Sci. **119**, 83–89 (2017)
19. Sokolov, A., Voloshinov, V.: Balanced identification as an intersection of optimization and distributed computing. arXiv preprint arXiv:1907.13444 (2019)
20. Sukhoroslov, O., Volkov, S., Afanasiev, A.: A web-based platform for publication and distributed execution of computing applications. In: 14th International Symposium on Parallel and Distributed Computing (ISPDC), pp. 175–184, June 2015
21. Sukhoroslov, O.: Integration of Everest platform with BOINC-based desktop grids. In: Third International Conference BOINC:FAST 2017, pp. 102–107 (2017)
22. Sukhoroslov, O.: Supporting Efficient Execution of Workflows on Everest Platform. In: Voevodin, V., Sobolev, S. (eds.) RuSCDays 2019. CCIS, vol. 1129, pp. 713–724. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36592-9_58
23. Sukhoroslov, O.: Supporting efficient execution of many-task applications with Everest. In: Proceedings of GRID 2018, pp. 266–270 (2018)
24. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: Workflows for e-Science: Scientific Workflows for Grids. Springer Publishing Company, Incorporated (2014)
25. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurrency Comput. Practice Exp. **17**(2–4), 323–356 (2005)
26. Thomas, M., Burruss, J., Cinquini, L., et al.: Grid portal architectures for scientific applications. In: Journal of Physics: Conference Series, vol. 16, p. 596. IOP Publishing (2005)

27. Volkov, S., Sukhoroslov, O.: A generic web service for running parameter sweep experiments in distributed computing environment. Procedia Comput. Sci. **66**, 477–486 (2015)
28. Volkov, S., Sukhoroslov, O.: Simplifying the use of clouds for scientific computing with Everest. Procedia Comput. Sci. **119**, 112–120 (2017)
29. Voloshinov, V., Smirnov, S., Sukhoroslov, O.: Implementation and use of coarse-grained parallel branch-and-bound in Everest distributed environment. Procedia Comput. Sci. **108**, 1532–1541 (2017)
30. Wächter, A., Biegler, L.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)