Design and Implementation of a Service for Cloud HPC Computations

Ruslan Kuchumov[1], Vadim Petrunin[1], Vladimir Korkhov[1](✉), Nikita Balashov[2], Nikolay Kutovskiy[2], and Ivan Sokolov[2]

[1] Saint Petersburg State University, 7/9 Universitetskaya nab., St. Petersburg 199034, Russia
kuchumovri@gmail.com, petrunin-vn@yandex.ru, v.korkhov@spbu.ru

[2] Joint Institute for Nuclear Research, Dubna, Russia
{balashov,kut,isokolov}@jinr.ru

Abstract. Cloud computing became a routine tool for scientists in many domains. In order to speed up an achievement of scientific results a cloud service for execution of distributed applications was developed. It obliviates users from manually creating virtual cluster environment or using batch scheduler and allows them only to specify input parameters to perform their computations. This service, in turn, deploys virtual cluster, executes supplied job and uploads its results to user's cloud storage. It consists of several components and implements flexible and modular architecture which allows to add on one side more applications and on another side various types of resources as a computational backends as well as to increase a utilization of cloud idle resources.

Keywords: Cloud computing · High performance computing
Software as a service

1 Introduction

At the moment a great number of large scale data processing centers are created worldwide including different scientific and commercial organizations. The majority of them deploy their own private cloud environments for hosting services and performing computations. The advantages of using cloud computing has been discussed many times, for example, paper [1] gives a good survey on this topic. From our point of view, the main benefit of using cloud computing is its flexible architecture and the ability to reduce the maintenance cost of computational infrastructure.

The growth of cloud computing has also led to the changes in scientific computations: it enabled scientists and researchers to launch high-performance applications without having computing infrastructure at their disposal. One of key factors in this case may be cloud computing pay-as-you-go financial model, when the user pays only for the resources he uses. Through the use of this model, renting virtual cloud resources only to execute a specific application is often more beneficial than to purchase, maintain and upgrade hardware at site. In addition to lower cost, clouds provide flexibility to choose both hardware and software components, which is not usually the case for traditional physical infrastructures where horizontal and vertical scaling are done by upgrading the hardware. So, the main advantages of using cloud infrastructure for scientific computing are flexibility, low cost of maintenance, and scalability.

Distributed computing paradigms can be characterized into high-performance computing (HPC), high-throughput computing (HTC) and many-task computing (MTC) which are usually distinguishes by the measure of computational tasks interconnect and task execution time [2]. In HPC applications the tasks are tightly-coupled and require large amount of computing power over a short period of time (hours or days). On contrary, HTC targets long running applications (for month or years) consisting of loosely-coupled tasks. MTC paradigm bridges gap between HTC and HPC. MTC applications can be distinguished by a very large number of tasks with relatively short per task execution time and they usually rely on disk I/O throughput rather than network throughput.

In previous research [3,4] we examined how flexible configuration of virtualized computing and networking resources can influence application performance and enable multi-tenancy with minimal mutual impact of simultaneously running parallel applications; in [5] we focused on Hadoop deployment and execution in virtual container-based clusters and investigated the dependency of Hadoop benchmarking suite performance on resource restrictions and other simultaneously running applications.

In this paper we are focusing on HPC applications for scientific computations where the most common workflow consists of the following stages. At first, a group of scientists needs to perform computations, for example, to do modelling, then the application for this purpose is being developed by their own effort or by separate team of developers, then this application is being ported to and executed in cluster environment. The same version of a application is usually executed many times with different input parameters or input datasets. When operating in a cloud environment, for each user or an application a new virtual cluster has to be created and configured either by users themself or by system administrators. In both cases it leads to delays in the achievement of scientific results and unnecessary complications.

What makes matter worse, is the problem that these scientists do not have and shouldn't have enough experience or knowledge on how to prepare their task for execution, how to operate cluster schedulers using command line interface and especially on how to configure a virtual cluster. Instead of all these concerns, they would prefer to focus on the problems in their scientific field and leave these that to system administrators.

To solve these problems a cloud service denoted as IdleUtilizer for helping users to perform HPC computations in a cloud environment was created. It provides users with a web interface that allows them to specify input parameters of their application, submit it for execution and receive its output when it's finished. When an application is submitted, this service will deploy a new virtual cluster in a cloud, configure it, prepare and execute user's application and then upload its results to user shared cloud data storage. The details of cluster configuration, its deployment and bootstrapping process are all hidden from the user. System administrator help is only may be required to add a new application to the system and for service maintenance.

Because of service's flexible architecture, IdleUtilizer provides users with a common interface for executing their job on different computational backends. Among them are different cloud providers (e.g. OpenNebula and OpenStack) and batch schedulers (e.g. Slurm) of a physical cluster.

2 Related Works

There are several works attempting to create web interfaces for batch schedulers, for example, there are implementations for HTCondor scheduler [6] or PBS scheduler [7]. The key difference is that our approach can be described as a common denominator across different schedulers where the user has ability only to specify input parameters of his application and where to perform computations, while scheduler-specific parameters, such as job file or input and output paths, are hidden from the user.

There are also systems for managing virtual clusters. "Virtual Cluster as a Service" [8] allows users to create a virtual cluster in OpenNebula cloud with preconfigured images, and to submit their jobs to this cluster. When user job is completed he or she may restart it with other input parameters or delete the cluster. "Dynamic Virtual Cluster" [9] is a similar system that allows to create a virtual cluster and then submit users jobs, but instead of accessing a cloud provides, it uses Xen hypervisor directly and deploys Moab scheduler for launching user jobs. In "Virtual Organization Cluster" [10] the system automatically creates a virtual cluster exclusively to execute the jobs coming from the grid. Similar to these approaches, we also create a virtual cluster for executing user jobs, but we also offer some flexibility in turns of choosing computational backends. Users can decide where to execute their tasks, he or she can choose different cloud providers or to execute the job in a physical scheduler. Related issues of adaptation and deployment of a task management system for a private cloud infrastructure and modelling of message passing middleware in cloud computing environments is considered in [11,12].

There are also works with similar goals as ours. HPCaaS [13] and Uncinus [14] allow users to run their applications in a cloud based on SaaS model. While the purpose is the same, its implementation differs. In these works, users must create virtual machine image for each application which would be used for deploying a cluster. This would not work in our approach as we are using physical clusters along with virtual and rely on batch schedulers to distribute users jobs. Besides that we could not find any similar systems available for deployment.

3 Design and Implementation

Figure 1 shows high level overview of Idle-Utilizer service architecture. It consists of three main components. At first, there is a web interface that is used by end-users for configuring and submitting jobs and monitoring their statuses. Web interface communicates only with the seconds component, idle-utilizer service. The main responsibilities of this service are to configure computational backend environment, to prepare, execute and monitor user's job, and to send the results back to the user. The third component is computational backed. In case the users prefers to do computation in a cloud, the cloud provider is accessed for deploying virtual cluster specifically for executing his job, and in case of the batch scheduler backend, the job would be created and submitted to the scheduler.

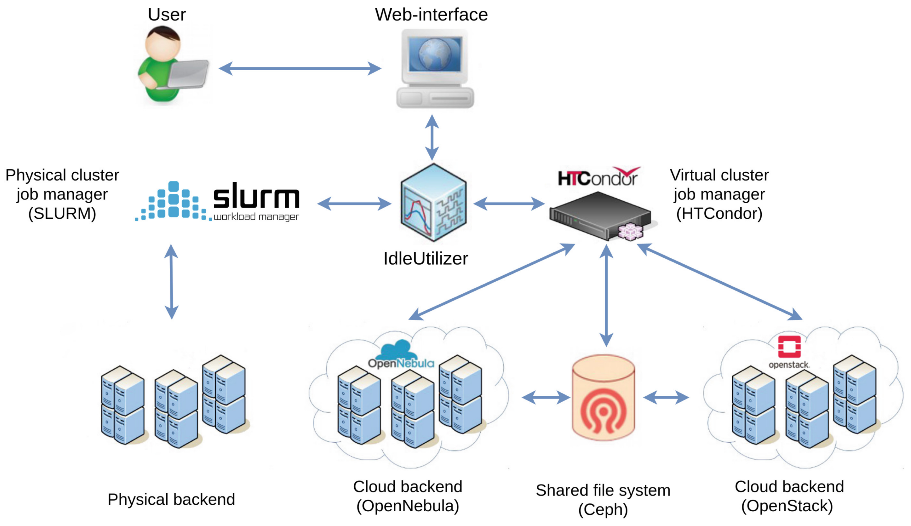
User
Web-interface
Physical cluster job manager (SLURM)
slurm workload manager
IdleUtilizer
HTCondor
Virtual cluster job manager (HTCondor)
OpenNebula
openstack
Physical backend
Cloud backend (OpenNebula)
Shared file system (Ceph)
Cloud backend (OpenStack)

Fig. 1. Idle-Utilizer high-level architecture.

The web interface component and users scenarios are covered in [15]. From IdleUtilizer point of view, the interactions are done through the request protocol defined over XML-RPC (remote procedure calls). Among the supported procedures are mainly the ones for submitting a new request, obtaining its state and deleting it. When submitted, request specifies such parameters as user job resources demand (number of nodes, CPUs, memory and time), task template and job input parameters, it also specifies which computational backend to use and where and how to upload job results. As request processing advances, additional information such as cluster configuration and batch scheduler status are stored in the request and is available for RPC client.

The Idle-Utilizer service is a core component as it is responsible for handling RPC request, orchestrating virtual cluster environment, submitting and monitoring jobs statuses in batch scheduler and uploading their results to user shared directory. This service has its own database for holding active requests. With a specific interval (2 s) it updates each request by changing its state as depicted in Fig. 2 In case an error occurs in any of the transitions between states, its message is stored in the database and can be reported to the user. Before and after each stage and on error custom scripts (so called hooks) can be executed e.g. for benchmarking, creating job-specific files or error reporting, but they usually hidden from end-users.

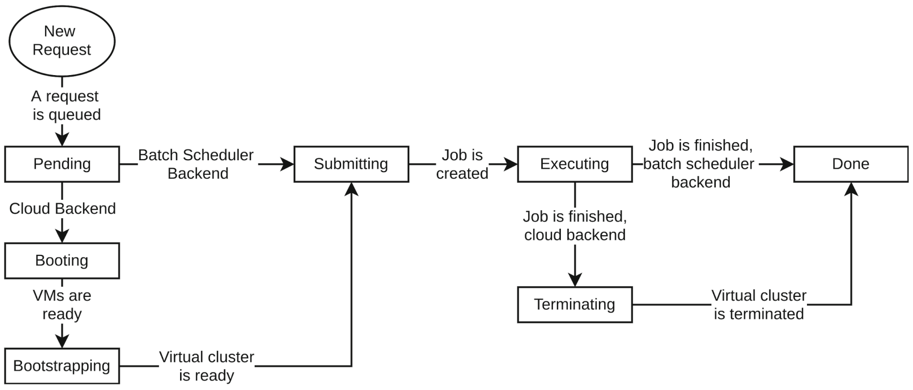

Fig. 2. Request states transitions.

As soon as the request is received it starts transitioning from pending to done (Fig. 2). In pending state virtual machines are requested to spawn, or in case of physical cluster backend, it transitions to submitting state. When virtual machines have started (booting stage), they autostart HTCondor software and add themselves into HTCondor pool (bootstrapping stage). During submitting stage (Fig. 3), job file is created and submitted to the scheduler. When the job is finished (after executing stage), virtual cluster is terminated and the request is considered to be done.

The computational backend is specified in users' request. Right now OpenNebula cloud and Slurm batch scheduler are supported. If batch scheduler backend is specified, request handling process is straightforward as IdleUtilizer mainly serves as an interface for a scheduler.

In case of a cloud backend, a new virtual cluster is deployed for executing user job. Each node of a virtual cluster is created from the same template and the same image, they only may differ by the amount of memory and the number of CPU cores. HTCondor scheduler software is installed in virtutal machine image and it is configured to automatically register itself in the pool of HTCondor scheduler. HTCondor scheduler is only used for assigning jobs for execution to a virtual cluster. Although the architecture can be simplified by omitting HTCondor scheduler, we decided to keep it for its simplicity of managing user jobs compared to manual approach and for consistency with physical cluster environment.

```
{
        'status': 'submitting',
        'status_updated_at': 1512462493,
        'resources': {
                'nodes': 2, 'memory': 4096, 'cpu': 4, 'time': 100},
        'scheduler': {
                'name': 'htcondor',
                'hosts': [ 'host-100', 'host-101' ]},
        'backend': {
                'name': 'opennebula',
                'owner': {'gid': 1, 'uid': 196},
                'hosts': [
                        { 'id': 100, 'ip': '10.10.0.100' },
                        { 'id': 101, 'ip': '10.10.0.101' }]},
        'template': '....',
        'user_data': {
                'key1': 'value1',
                # Job parametres
        },
}
```

Fig. 3. An example of request fields at submitting stage.

Similar to physical cluster environment, the nodes of virtual cluster have shared file system mounted for executing distributed applications and exchanging job input and output files.

Before the job is executed, user's custom scripts in scheduler manager node can be executed for obtaining input files either by generating them from job request value or by downloading them from remote location. When the job is finished its output is uploaded to user's cloud storage directory.

4 Experiments

In our experiment we used NAS Parallel Benchmarks (NPB) which consists of several widely used set of programs designed to evaluate the performance of HPC systems. The benchmarks is a set of five kernels program and three pseudo-applications that mimic scientific computations. Problem sizes in NPB are predefined and indicated as different classes. We used MG, FT, CG and IS kernels of A, B and C classes:

– IS (Integer Sort): sorts small integers using the bucket sort. This test is CPU and Memory intensive but with low inter-node communications.
– CG (Conjugate Gradient): calculates conjugate gradient method. This test is both memory- and communication-intensive.

- MG (Multi-Grid): solves discrete Poisson equation using multi-grid method. This test is both memory- and communication-intensive.
- FT (Fourier Transform): solves three-dimensional partial differential equation using the fast Fourier transform. This benchmarks tests inter-node all-to-all communications.

For a testbed we used OpenNebula cloud with KVM hypervisor provided by JINR [16].

Experiments showcasing low overhead of virtualization for HPC applications have been done many times by different authors [8,17]. Instead, in experiments, we tried to justify the need for automatic virtual cluster configuration depending on job input parameters. There are two values that should be minimized: the size of virtual cluster (the number of virtual machines and their memory requirements) and job execution time. Figure 4 shows that for each benchmark the minimal amount of memory can be set without sacrificing the performance. Also Fig. 5 shows that by increasing the number of virtual machines from 4 to 8 in some cases (especially in cg and mg) yields the performance increase significantly less in comparison with 2 to 4. In other experiments (Fig. 6) we have throttled network bandwidth to show that in some hardware configurations there may be peaks of performance at the certain number of nodes, and when this number increases further, job performance degrade drastically. This can be noticed with network bandwidth of 50 Mbits/s, when the optimal number of nudes is 2 or 4.

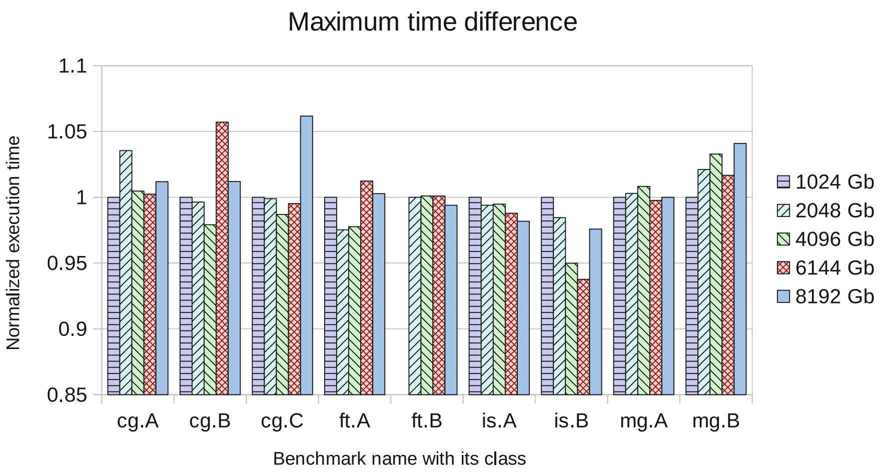

Fig. 4. The maximum of normalized benchmark execution time for different amounts of memory. Execution time with 1024 Gb considered as 1.

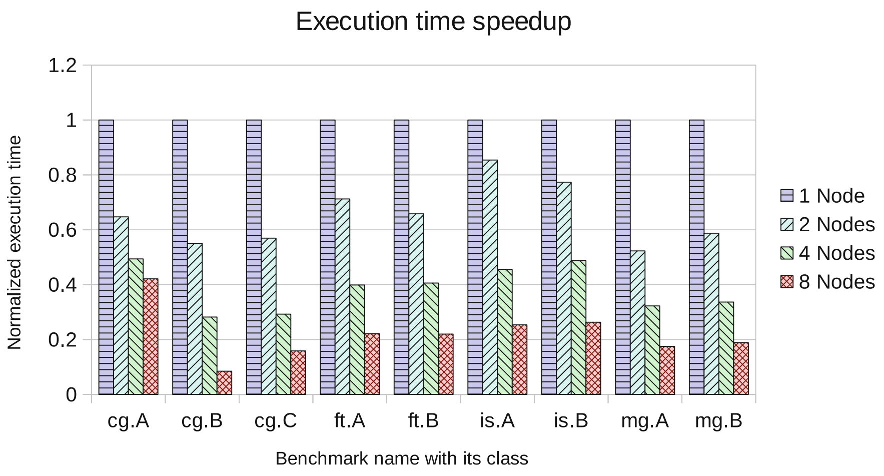

Fig. 5. Execution time speedup of different number of nodes.

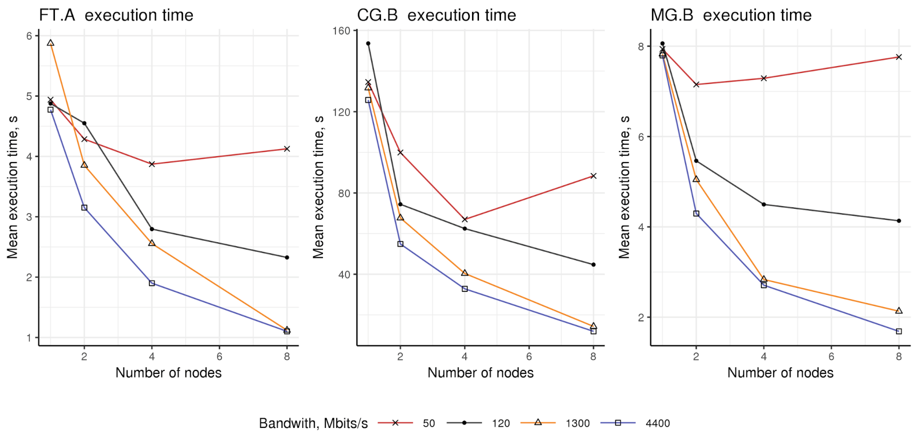

Fig. 6. Execution time with throttled network bandwidth.

5 Conclusion and Future Works

We have created a software-as-a-service system for launching HPC applications. IdleUtilizer service allows users to execute their jobs by only specifying input parameters via web interface and then to obtain the results via their cloud storage. To achieve this, IdleUtilizer service creates a virtual cluster in a cloud, creates and submits the job for execution on this cluster and, when it's done, transfers results to user. Besides the cloud environment, user jobs can also be submitted for execution to the scheduler of physical HPC cluster.

The primary goal of this service is to simplify the workflow of HPC application execution for scientists. IdleUtilizer service has already been deployed in JINR cloud [16] and used with long Josephson junction modelling application [18].

In the current version, users still have to make decision on a cluster configuration, i.e. he or she needs to know job resource requirements to specify the number of nodes, the amount of memory and CPUs per node. Finding these values for each set of application parametres sometimes may be rather tedious. But, as our experiments have shown, it may help to reduce not only the job execution time, but also job unused resources. To simplify this process we are planning to extend the functionality of IdleUtilizer service so that before every execution, it will analyze the information about previous job executions and then hint user an optimal cluster configuration.

Acknowledgements. Research has been supported by the RFBR grant 16-07-01111.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010). https://doi.org/10.1145/1721654.1721672
2. Shawish, A., Salama, M.: Cloud computing: paradigms and technologies. In: Xhafa, F., Bessis, N. (eds.) Inter-cooperative Collective Intelligence: Techniques and Applications, Studies in Computational Intelligence, vol. 495. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-35016-0_2
3. Korkhov, V., Kobyshev, S., Krosheninnikov, A.: Flexible configuration of application-centric virtualized computing infrastructure. In: Gervasi, O., et al. (eds.) ICCSA 2015. LNCS, vol. 9158, pp. 342–353. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21410-8_27
4. Korkhov, V., Kobyshev, S., Krosheninnikov, A., Degtyarev, A., Bogdanov, A.: Distributed computing infrastructure based on dynamic container clusters. In: Gervasi, O., et al. (eds.) ICCSA 2016. LNCS, vol. 9787, pp. 263–275. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42108-7_20
5. Korkhov, V., Kobyshev, S., Degtyarev, A., Bogdanov, A.: Light-weight cloud-based virtual computing infrastructure for distributed applications and hadoop clusters. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10408, pp. 399–411. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62404-4_29
6. Chapman, C., Goonatilake, C., Emmerich, W., Farrellee, M., Tannenbaum, T., Livny, M., Calleja, M.: Condor Birdbath - web service interface to Condor. In: Cox, S.J., Walker, D.W., (eds.) Proceedings of the UK e-Science All Hands Meeting 2005, EPSRC: Swindon, UK (2005)
7. Ma, G., Lu, P.: PBSWeb: a web-based interface to the portable batch system. In: 12th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), Las Vegas, NV, 6–9 November 2000. ACTA Press, Calgary (2000)
8. Doelitzscher, F., Held, M., Reich, C., Sulistio, A.: ViteraaS: virtual cluster as a service. In: 2011 IEEE Third International Conference on Coud Computing Technology and Science (CloudCom), pp. 652–657. https://doi.org/10.1109/CloudCom.2011.101

9. Emeneker, W., Jackson, D., Butikofer, J., Stanzione, D.: Dynamic virtual clustering with Xen and Moab. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA 2006. LNCS, vol. 4331, pp. 440–451. Springer, Heidelberg (2006). https://doi.org/10.1007/11942634_46
10. Murphy, M.A., Kagey, B., Fenn, M., Goasguen, S.: Dynamic provisioning of virtual organization clusters. In: CCGRID 2009 Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 364–371. https://doi.org/10.1109/CCGRID.2009.37. ISBN 978-0-7695-3622-4
11. Iakushkin, O., Shichkina, Y., Sedova, O.: Petri Nets for modelling of message passing middleware in cloud computing environments. In: Gervasi, O., et al. (eds.) ICCSA 2016. LNCS, vol. 9787, pp. 390–402. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42108-7_30
12. Iakushkin, O., Malevanniy, D., Bogdanov, A., Sedova, O.: Adaptation and deployment of PanDA task management system for a private cloud infrastructure. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10408, pp. 438–447. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62404-4_32
13. Church, P., Wong, A., Brock, M., Goscinski, A.: Toward exposing and accessing HPC applications in a SaaS cloud. In: IEEE 19th International Conference on Web Services, Honolulu, HI, pp. 692–699 (2012). https://doi.org/10.1109/ICWS.2012.119
14. Church, P., Goscinski, A., Tari, Z.: SaaS clouds supporting non computing specialists. In: 11th International Conference on Computer Systems and Applications (AICCSA), Doha, pp. 1–8 (2014). https://doi.org/10.1109/AICCSA.2014.7073171
15. Balashov, N.A., Bashashin, M.V., Kuchumov, R.I., Kutovskiy, N.A., Sokolov, I.A.: JINR cloud service for scientific and engineering computations. Mod. Inf. Technol. IT Educ. **14**(1) (2018). https://doi.org/10.25559/SITITO.14.201801.061-072
16. Baranov, A.V., Balashov, N.A., Kutovskiy, N.A., Semenov, R.N.: JINR cloud infrastructure evolution. Phys. Part. Nuclei Lett. **13**, 672–675 (2016). https://doi.org/10.1134/S1547477116050071
17. Menon, A., Santos, J.R., Turner, Y., Janakiraman, G.J., Zwaenepoel, W.: Diagnosing performance overheads in the XEN virtual machine environment. In: VEE 2005, Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments, pp. 13–23, New York, NY, USA (2005)
18. Bashashin, M.V., et al.: Numerical approach and parallel implementation for computer simulation of stacked long Josephson junctions. Comput. Res. Model. **8**(4), 593–604 (2016)