HPC TaskMaster – Task Efficiency Monitoring System for the Supercomputer Center

Pavel Kostenetskiy(✉), Artemiy Shamsutdinov, Roman Chulkevich, Vyacheslav Kozyrev, and Dmitriy Antonov

HSE University, 11, Pokrovsky boulevard, Moscow 109028, Russia
pkostenetskiy@hse.ru

Abstract. This paper is devoted to the monitoring system *HPC TaskMaster* developed at the HSE University for the *cHARISMa* cluster. This system automatically evaluates the efficiency of performing tasks of HPC cluster users and identifies inefficient tasks, thereby significantly saving the expensive machine time. In addition, users can view reports on completing their tasks, along with inferences about their work and interactive graphs. Particular attention in this paper is paid to determining the effectiveness of the task – the system allows the administrator to personally configure the criteria for evaluating the effectiveness of the task without the need for changes in the source code. The system is developed using open-source software and is publicly available for use on other clusters.

Keywords: HPC cluster · efficiency · monitoring

1 Introduction

A task efficiency monitoring system is essential for detecting incorrectly started calculations that entail the insufficiently efficient use of cluster resources. This paper describes a new task performance monitoring system, *HPC TaskMaster*, developed at the HSE University for the *cHARISMa (Computer of HSE for Artificial Intelligence and Supercomputer Modeling)* cluster.

The developed system allows users to view reports on the performance of their tasks together with interactive execution schedules and automatically identify tasks that worked inefficiently. Having access to the results of the analysis, users can run their tasks more efficiently in the future, which will significantly save the machine time of the cluster.

In addition, the system will allow the administrators of the cluster to collect statistics about user tasks, which was previously unavailable.

L. Sokolinsky and M. Zymbler (Eds.): PCT 2022, CCIS 1618, pp. 17–29, 2022.
https://doi.org/10.1007/978-3-031-11623-0_2

The most common examples of the inefficient usage of cluster resources are:

- allocation of insufficient or excessive resources for a task;
- running a non-parallel task on multiple CPU cores or GPUs;
- allocation of the compute node capacity without starting calculations.

The following requirements were defined for the design of the task performance monitoring system.

1. The system should collect the following data for each task:
   - utilization of specific CPU cores allocated for the task;
   - utilization of GPUs allocated for the task;
   - GPU memory utilization;
   - GPU power consumption;
   - utilization of RAM created by the task;
   - file system usage.
2. The system must analyze the collected data and use it to determine whether the task worked effectively.
3. The system must provide users with access to the list of completed tasks and reports on their completion using a web application.

The rest of this paper is organized as follows. A comparison of different monitoring systems is carried out in Sect. 2. In Sect. 3, the architecture of the system is described. The detection of inefficient user tasks is considered in Sect. 4. User statistics are provided in Sect. 5. Finally, Sect. 6 shows the conclusions of this work.

2 Related Work

The key feature of the HSE cluster is how it allocates resources for user tasks. Instead of allocating the entire compute node for one task, the user is given a certain number of processor cores and GPUs. As a result, several dozen tasks can be performed on the compute node at once, thus optimizing cluster resources. Due to this feature, ready-made solutions for monitoring system resources, such as Nagios and Zabbix, are not suitable for this cluster. *cHARISMa* already has a monitoring system of its own [4], however, it is designed to display only the global usage across the whole cluster and its nodes.

Since one of the HSE University goals is to provide cluster users with a secure system in the HSE University environment, a new monitoring system was built using open-source monitoring tools. Chan [3], Wegrzynek [11], Kychkin [6], Safonov [10] describe how using a combination of programs such as *Telegraf*, *InfluxDB* and *Grafana* allows one to quickly set up and run a cluster resource monitoring system. In [2,3], it is also described how the *Slurm* plugin acct_gather enables to collect metrics for *Slurm* tasks, which is precisely the data required for a task efficiency monitoring system. Since all programs, except *Telegraf*, are already installed on *cHARISMa*, this approach can be used to monitor tasks on the cluster.

The development of LIKWID Monitoring Stat [9], a task monitoring system using InfluxDB, Grafana and built-in LIKWID tools for monitoring tasks on the cluster, also draws attention. For each task, a dashboard is created from ready-made JSON templates, which allows creating personalized graphs for each task. The disadvantages of using the LIKWID Monitoring Stack on the HSE Cluster include the need to use LIKWID tools for the system to operate and the lack of a web interface for the system in addition to Grafana, which makes the system inconvenient for using on a cluster with a large number of users and tasks.

In addition to monitoring cluster resources, the system must analyze the effectiveness of user tasks. A well-known system for creating reports on the effectiveness of tasks is JobDigest [7,8]. It analyzes the collected integral values and, based on them, applies a tag to the task describing the property of the task (for example, "low GPU utilization"). Although using tags is convenient for searching and filtering tasks, it is not always possible to provide an overall picture of the effectiveness of the task using tags alone.

Summarizing all the above, we can conclude that there is no ready-made task monitoring system fitting the individual characteristics of the cHARISMa cluster, which can be integrated into the HSE University environment. It is necessary to develop its own software system for evaluating the effectiveness of tasks, which can be flexibly configured for specific types of user tasks, delimit access for cluster users, and take into account the compliance of tasks with registered scientific and educational projects. As the basis of the system, it is worth using the open-source software *Telegraf*, *InfluxDB* and *Grafana*.

3 System Architecture

This section describes the monitoring infrastructure of the *HPC TaskMaster* system, shown in Fig. 1.

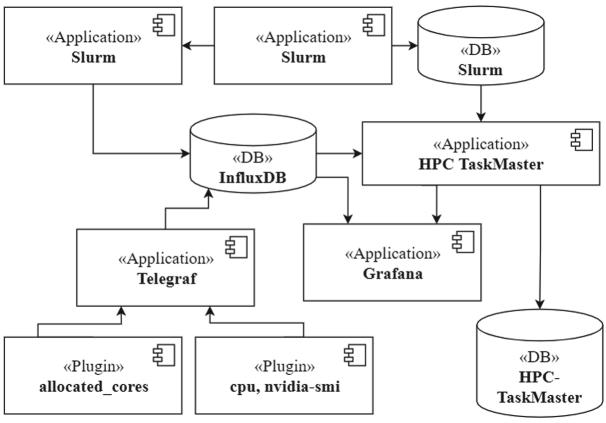

Fig. 1. Diagram of the system components

The *Slurm* task scheduler is used to run tasks on the cluster. The main data of *Slurm* tasks is stored in the *MySQL* relational database using the background process *slurm database (slurmdbd)*, and the task metrics are written to the *InfluxDB* time series database using the plugin *acct_gather*. This plugin collects memory and filesystem usage (read/write) for each task.

The required metrics of utilizing specific CPU cores and GPUs are collected with the *Telegraf* daemon, which has built-in plugins for these metrics. Thus, having the CPU and GPU IDs assigned to the task, the system can collect metrics for the components and, therefore, distinguish utilization for different tasks on one node. Additional metrics are collected using developed plugins in Python.

The collected metrics are stored in the *InfluxDB* database. InfluxDB was chosen as a time-series database because of Telegraf support and Slurm acct_gather plugin support, which allows one to store all the required metrics in one database.

Grafana is used as a tool for visualizing graphs on the *cHARISMa* cluster. *Grafana* provides great opportunities for configuring and formatting charts and also has support for creating them using the API. This API allows automating the creation of graphs for each task. New graphs for each task are created using JSON templates. Based on the available data about the task, when the user requests it, graphs are automatically built in Grafana. The created graphs are displayed on the system's website using iframe technology, where the user can interactively view the graphs for the period of task execution. In addition, the system creates graphs for both completed and running tasks. Thereby, the user can observe the work of his task in real time.

The advantage of using a combination of *Telegraf*, *InfluxDB* and *Grafana* is the ability to install and configure these tools on any cluster. Moreover, these tools make the monitoring system quite flexible – additional data for the system can be collected using the built-in plugins of *Telegraf* or developed ones.

It is important to pay attention to the fact that the *HPC TaskMaster* system has a negligible impact on the performance of compute nodes; the installed Telegraf daemon uses only 0.03% of the overall CPU performance. In addition to Telegraf, another source of the computing cluster load is InfluxDB. Installed on the head node, InluxDB uses an average of 5 GB of storage per month. To free up storage, a retention policy that compresses metrics older than 6 months is used.

The *HPC TaskMaster* system is developed on Django, a Python web framework that has a large number of available packages and a wide range of tools for developing web applications, which allows one to develop a monitoring system using Telegraf, InfluxDB and Grafana. In addition, Django has a built-in administration panel through which the administrator can configure the monitoring system himself without making changes to the source code of the program.

The task performance monitoring system works according to the following principles:

– metrics are collected on each compute node using Telegraf and stored in the InfluxDB database on the head node. Metrics from the *acct_gather* plugin are also stored in InfluxDB;
– the system updates its local MySQL database by comparing its tasks with those from the Slurm database;
– while the task is running, aggregated metrics are collected for it from the InfluxDB database with a certain period;
– if the task is completed, its aggregated metrics are collected for the last time;
– the collected aggregated metrics are analyzed by the system, and an inference about the efficiency of the task is generated.

4 Detecting Inefficient Tasks

The user interacts with the HSE high-performance computing cluster [4] by launching tasks through the SLURM workload manager. A task is a set of user processes for which the workload manager allocates computing resources (compute nodes, CPUs, GPUs, etc.) Each launch of the user's program for execution generates a new task, which is collected in the database and analyzed.

Here we define task efficiency as the usage of allocated resources above a certain threshold.

4.1 Collected Data

HPC TaskMaster collects two types of data about running tasks on the HPC cluster:

1) parameters characterizing the running task;
2) metrics that characterize the execution of the task.

Parameters. Table 1 shows the task parameters and their type.

Metrics
Table 2 shows the metrics collected during the execution of the task. The metrics form a time series θ_i. $\Theta = \{\theta_i\}$ denotes the set of all-time series of the task.

The frequency of collecting metrics can be adjusted and selected in such a way as to obtain sufficiently detailed information about the task without overloading the system with data collection and storage.

4.2 Data Processing

Aggregated Metrics
To simplify the analysis, aggregated metrics $\Lambda^k = (\lambda_1^k, \cdots, \lambda_m^k)$ are calculated for each time series [5]. They include the minimum, maximum, average, median and standard deviations. In addition to them, the tuple Λ includes the average load of each node and the combined average load of the nodes.

Table 1. Parameters of the task

№	Parameter	Type
1	ID	Integer
2	Task name	String
3	Status	
4	Launch command	
5	Type of compute nodes	
6	Number of compute nodes	Integer
7	Number of CPU cores	
8	Number of GPUs	
9	Exit code	
10	User ID	
11	Project ID	
12	Start date and time	Date
13	End date and time	

Table 2. Collected metrics and collection frequency

№	Metrics	Frequency, seconds	Units of measurement
1	CPU cores usage by the user	10	percentages
2	CPU cores usage by the system		
3	GPU usage		
4	RAM usage		kilobyte
5	GPU memory usage		
7	GPU power consumption		watt
8	File System access	60	megabyte

Tags

Since the task parameters are a heterogeneous set of data (integers, strings, dates), to simplify their analysis, a system of tags, i.e., "labels" indicating the type of task, execution time, and other properties of the task, is introduced. Table 3 contains a list of tags currently available in the system. Additional tags can be developed and implemented into the system.

The tuple $T^k = (\tau_1^k, \ldots, \tau_n^k)$ is assigned to the task with the ID k, where n is the number of tags in the system. The τ_i element corresponds to the indicator of the i tag and takes the value 1 if all conditions are met and the tag is assigned to the task, and 0 otherwise.

Indicators

To determine if the task is working inefficiently, it is necessary to evaluate the disposal of the components involved in the task. To do this, the concept of *indicator of problems* is introduced.

Table 3. List of tags

№	Tags	Type
1	Jupyter-notebook task	String
2	LAMMPS task	
3	VASP task	
4	Allocation of resources for calculations	
5	The task lasted less than a minute	
6	The task was completed with an error	

Indicators, dimensionless values inversely proportional to the value of the metrics, are used to evaluate the disposal of the components involved in the task.

Indicators take a value from 0 (with the full use of allocated resources) to 1 (otherwise). For example, the value of the indicator l_j is calculated from the aggregated metric $\lambda_j^k \in \Lambda^k$ using formula (1).

$$l_j^k = 1 - \frac{\lambda_j^k - a_j}{b_j - a_j}, \quad l_j \in [0, 1], \tag{1}$$

where a_j, b_j are the admin defined parameters referring to the minimum and maximum possible values of the j-th element of the aggregated metrics.

Indicators are placed in the tuple of indicators $L^k = (l_1^k, \ldots, l_m^k)$.

The list of currently available indicators is presented in Table 4. Additional indicators can be developed and implemented into the system. The number of indicators for a specific task depends on the number of cores, compute nodes and GPUs used.

Table 4. List of indicators

№	Indicators
1	Low average CPU usage
2	Low average CPU core usage
3	Low average GPU usage
4	Low GPU memory usage
5	The task was completed with an error

4.3 Inferences

To help users to interpret the results, the system has a set of inferences $\Phi = (\phi_i)$. Inferences are the result of the analysis of the task.

Different requirements for tags and indicator values are set for each inference. An inference is assigned to the task when all the conditions are met. Several inferences can correspond to one task at once.

Denote the union of tuples of indicators L and tags T as

$$N^k = (l_1^k, \ldots, l_n^k, \tau_1^k, \ldots, \tau_m^k). \tag{2}$$

Let Ω_i be a set of conditions for the output of ϕ_i to the elements of the tuple N^k.

Then we can match the set C^k to each problem:

$$C^k = \{\phi_i \in \Phi : \Pi_{\omega \in \Omega_i} \mathbb{1}_\omega(N^k) = 1\}, \tag{3}$$

where $\mathbb{1}_\omega$ is the indicator function equal to 1 if the condition $\omega \in \Omega_i$ is met. In other words, the tuple C^k contains the inferences assigned to the task.

4.4 Example

Let us consider a computational task performed on the *cHARISMa* supercomputer using 176 cores and 16 NVIDIA Tesla V100 GPU accelerators on 4 compute nodes. Table 5 shows the parameters of the task.

Table 5. Parameters of the task

№	Parameter	Value
1	ID	405408
2	Task name	SimpleRun
3	Status	Successful
4	Exit code	0
5	Launch command	sbatch run_task.sh
6	User ID	2000
7	Project ID	32
8	Start date and time	November 11, 2021 10:13:28
9	End date and time	November 12, 2021 13:19:09
10	Type of compute nodes	type_a
11	Number of compute nodes	4
12	Number of CPU cores	176
13	Number of GPUs	16

The aggregated metrics across all compute nodes for the example task are shown in Table 6.

Table 6. Aggregated metrics by node

№	Metrics	Value
1	Avg. load of cores on comp. node cn-001	99.36
2	Avg. load of cores on comp. node cn-002	99.11
3	Avg. load of cores on comp. node cn-003	99.15
4	Avg. load of cores on comp. node cn-004	99.51
5	Avg. load of comp. nodes	99.28
7	Avg. utilization of GPUs on comp. node cn-001	71.62
8	Avg. utilization of GPUs on comp. node cn-002	71.6
9	Avg. utilization of GPUs on comp. node cn-003	71.15
10	Avg. utilization of GPUs on comp. node cn-004	71.8
11	Avg. utilization of GPUs	71.54

Table 7 shows the aggregated metrics of the time series for compute node cn-001. Data for compute nodes cn-002, cn-003, cn-004 are not shown to save space.

Table 7. Aggregated metrics of compute node cn-001

Node cn-001		**Min**	**Avg**	**Max**
CPU usage by the **system**				
1	Core 1	0	0.12	11.4
⋮	⋮	⋮	⋮	⋮
44	Core 44	0	0.13	7
CPU usage by the **user**				
45	Core 1	0	98.9	100
⋮	⋮	⋮	⋮	⋮
88	Core 44	0	99.8	100
89	Average usage of cores on the node	99.36		
GPU usage №:0				
90	Utilization	0	71.62	99
91	Memory usage, MB	0	7095.3	8780
92	Power consumption, Watt	66	128.9	156.1
⋮	⋮	⋮	⋮	⋮
GPU usage №:3				
99	Utilization	0	71.61	99
100	Memory usage, MB	0	7095.3	8780
101	Power consumption, Watt	66	129	155.9
102	RAM usage, MB	0.35	128.29	715.44
File system access, GB				
103	Read	0	141389.39	288706.41
104	Write	0	1302.76	2753.31

Tags of the Task
Based on the parameters of the task from Table 5 and the tags from Table 3, no tag will be assigned to task *405408*, since it is completed without an error and is not the launch of one of the packages. Therefore, the tuple of task tags will have the form $T^{405408} = (0, 0, 0, 0, 0, 0)$.

Indicators of the Task
Based on the data from Tables 6, 7, the system calculates the values of the indicators shown in Table 8.

Table 8. List of indicators

N	Indicator	Value
Compute node **cn-001**		
1	Core 1	0.011
⋮	⋮	⋮
44	Core 44	0.002
207	GPU №:0 utilization	0.284
⋮	⋮	⋮
210	GPU №:3 utilization	0.284
223	GPU №:0 memory usage	0.778
⋮	⋮	⋮
226	GPU №:3 memory usage	0.778
⋮	⋮	⋮
Compute node **cn-004**		
205	Core 1	0.011
⋮	⋮	⋮
206	Core 40	0.002
207	GPU №:0 utilization	0.279
⋮	⋮	⋮
208	GPU №:4 utilization	0.28
209	GPU №:0 memory usage	0.779
⋮	⋮	⋮
210	GPU №:3 memory usage	0.778
Summary		
239	Avg. load of cores on node **cn-001**	0.006
240	Avg. load of cores on node **cn-002**	0.009
241	Avg. load of cores on node **cn-003**	0.008
242	Avg. load of cores on node **cn-004**	0.005
243	Avg. load of nodes	0.007
244	Avg. utilization of GPUs on node **cn-001**	0.284
245	Avg. utilization of GPUs on node **cn-002**	0.284
246	Avg. utilization of GPUs on node **cn-003**	0.289
247	Avg. utilization of GPUs on node **cn-004**	0.282
248	Avg. utilization of GPUs	0.285

Inferences of the Task

After the previous steps, we get a tuple

$N^{405408} = (l_1, \ldots, l_{202}, \tau_1, \ldots, \tau_6)$

As an example, let us consider the three outputs presented in Table 9.

Table 9. Inferences

ϕ_i	Inference	Conditions	Cond. is met
1	Successful task	$l_i \leq 0.5$, $i = 1, \cdots, 248$	Yes
		$\tau_i = 0$, $i = 5, 6$	Yes
2	Task completed with an error	$\tau_5 = 1$	No
3	Inefficient CPU usage	$l_i > .5\, i = 1, \cdots, 206, 239, \cdots, 243$	No
4	GPU is not used	$l_i \leq 0.5$, $i = 1, \cdots, 206, 211, \cdots, 215$	No
		$l_i > 0.8$, $i = 207, \cdots, 238, 244, \cdots, 248$	No

Based on the tuple N^{405408}, the system will associate the set $C^{405408} = \{\phi_1\}$ with task *405408*, since the task is executed without errors and all resources are used.

An example of the task report with an inference of inefficient salloc usage is shown in Fig. 2.

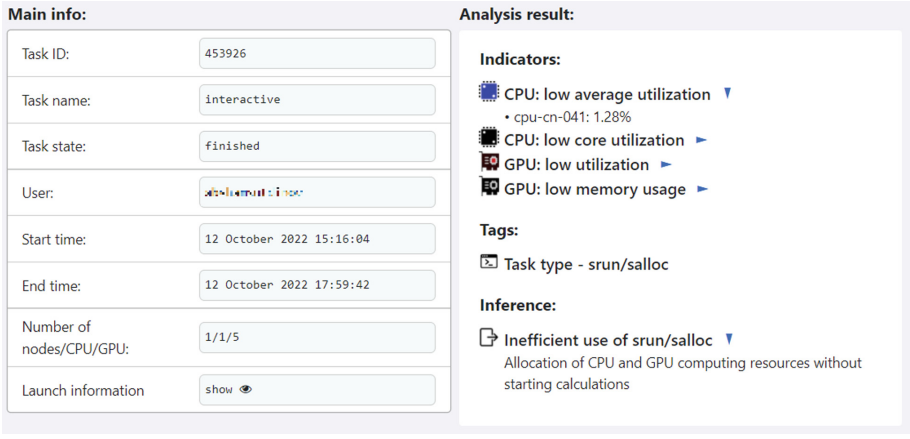

Fig. 2. Task report

5 User Statistics

System administrators have access to inference statistics for each cluster user for a selected period of time. An example of statistics is shown in Fig. 3. Using

this pie chart, administrators can understand which types of tasks are causing difficulties for the user. After determining the problem that the user has encountered, he can get a personal consultation to solve this problem.

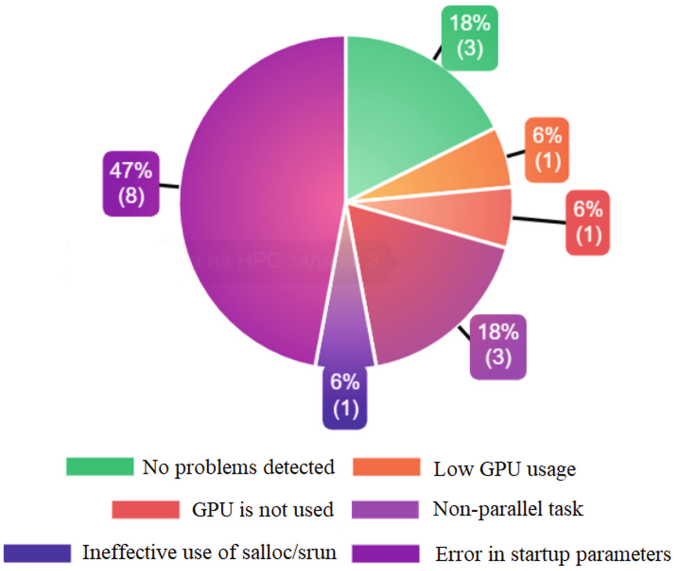

Fig. 3. Graphs of the utilization of computing resources by the task

Statistics of the most active users of the cluster with the lowest percentage of effective tasks are compiled monthly; personal consultations are held on the basis of the statistics. By tracking trends in user efficiency by month, we can conclude how the *HPC TaskMaster* system can increase the efficiency of using cluster resources.

6 Conclusions

The developed task performance monitoring system, *HPC TaskMaster*, is a powerful tool that provides all the necessary information (main information, aggregated metrics, graphs, and inferences) about tasks in one place. This system will help users to identify the problem for existing scientific applications and applications of their development, thereby simplifying work with the cluster for users, allowing them to perform scientific calculations faster and more efficiently in the future.

HPC TaskMaster is constantly evolving and improving. Among the future directions for development are:

– monitoring the effectiveness of individual categories of applications using machine learning tools;

– adding new types of indicators and tags to generate new inferences;
– smart recognition of the type of running application;
– development of a module for notifying users about the launch of inefficient tasks by them.

HPC TaskMaster is available to all cluster users of *cHARISMa* via the personal account of the supercomputer complex. *HPC TaskMaster* is also available for public use [1], and any suggestions for improving the project are greatly appreciated.

The research was performed using the *cHARISMa* HPC cluster of the HSE University [4].

References

1. Open Source/HPC TaskMaster GitLab. https://git.hpc.hse.ru/open-source/hpc-taskmaster
2. Slurm Workload Manager - acct_gather.conf. https://slurm.schedmd.com/acct_gather.conf.html
3. Chan, N.: A resource utilization analytics platform using grafana and telegraf for the Savio supercluster. In: ACM International Conference Proceeding Series. Association for Computing Machinery (2019). https://doi.org/10.1145/3332186.3333053
4. Kostenetskiy, P.S., Chulkevich, R.A., Kozyrev, V.I.: HPC resources of the higher school of economics. J. Phys. Conf. Ser. **1740**, 012050 (2021). https://doi.org/10.1088/1742-6596/1740/1/012050
5. Kraeva, Y., Zymbler, M.: Scalable algorithm for subsequence similarity search in very large time series data on cluster of phi KNL. In: Manolopoulos, Y., Stupnikov, S. (eds.) DAMDID/RCDL 2018. CCIS, vol. 1003, pp. 149–164. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23584-0_9
6. Kychkin, A., Deryabin, A., Vikentyeva, O., Shestakova, L.: Architecture of compressor equipment monitoring and control cyber-physical system based on influxdata platform. In: 2019 International Conference on Industrial Engineering, Applications and Manufacturing, ICIEAM 2019 (2019). https://doi.org/10.1109/ICIEAM.2019.8742963
7. Nikitenko, D., et al.: JobDigest - detailed system monitoring-based supercomputer application behavior analysis. In: Voevodin, V., Sobolev, S. (eds.) Supercomputing. Communications in Computer and Information Science, vol. 793, pp. 516–529. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71255-0_42
8. Nikitenko, D.A., Voevodin, V.V., Zhumatiy, S.A.: Deep analysis of job state statistics on Lomonosov-2 supercomputer. Supercomput. Front. Innov. **5**(2), 4–10 (2018). https://doi.org/10.14529/jsfi180201
9. Rohl, T., Eitzinger, J., Hager, G., Wellein, G.: Likwid monitoring stack: a flexible framework enabling job specific performance monitoring for the masses (2017). https://doi.org/10.1109/CLUSTER.2017.115
10. Safonov, A., Kostenetskiy, P., Borodulin, K., Melekhin, F.: A monitoring system for supercomputers of SUSU. In: Proceedings of Russian Supercomputing Days International Conference, vol. 1482, pp. 662–666. CEUR-WS (2015)
11. Wegrzynek, A., Vino, G.: The evolution of the ALICE O 2 monitoring system. In: EPJ Web of Conferences, vol. 245 (2020). https://doi.org/10.1051/epjconf/202024501042