Enhancement of the Data Analysis Subsystem in the Task-Efficiency Monitoring System HPC TaskMaster for the cHARISMa Supercomputer Complex at HSE University

Pavel Kostenetskiy(✉), Vyacheslav Kozyrev, Roman Chulkevich, and Alina Raimova

Higher School of Economics (HSE) University, 11 Pokrovsky boulevard, Moscow 109028, Russia
pkostenetskiy@hse.ru

Abstract. The detection of computational tasks that inefficiently utilize high-performance computing (HPC) resources is one of the major problems facing supercomputer centers. Such tasks can block valuable computational resources and slow down other supercomputer users' computations. *HPC TaskMaster*, a task-performance monitoring system developed at the Higher School of Economics, addresses this issue by analyzing task metrics, aggregating them, calculating indicator values, assigning tags, and automatically generating inferences about task performance. In this paper, we describe the enhancement of the HPC TaskMaster subsystem for analyzing the efficiency of tasks by introducing a new entity into it: *parameters*. This extension enables the detection of new types of problems, such as the incorrect selection of the type and number of computational resources. Additionally, it allows one to consider the variability of parameters in the inferences generated by the system.

Keywords: HPC cluster · Monitoring · Efficiency

1 Introduction

Improving the utilization efficiency of high-performance computational resources is an essential task for supercomputer centers. To control the efficiency of user tasks performed on the *cHARISMa* HPC cluster [8], the HSE Supercomputer Modeling Unit has developed HPC TaskMaster, a task-performance monitoring system [2,10].

The user interacts with the HSE high-performance computing cluster by running tasks through the SLURM scheduler. A task is a set of user processes for which the scheduler allocates computational resources (processor cores, graphics processors, computing nodes, and so forth) [17]. Each launch of a user's program

L. Sokolinsky et al. (Eds.): PCT 2024, CCIS 2241, pp. 49–64, 2024.
https://doi.org/10.1007/978-3-031-73372-7_4

for execution generates a new task. The task metrics are saved to a database and analyzed for efficiency. We define efficient tasks as those that create a load on the computational resources allocated to it above a given threshold [16].

The HPC TaskMaster system consists of five subsystems: a calculation user statistics subsystem, an automatic mailing subsystem, a reception and processing metrics subsystem, an inferences subsystem, and a data analysis subsystem. In this paper, we describe an enhancement of the data analysis subsystem [11]. The result of the analysis of a computational task consists of three parts: indicators, tags, and inferences. Indicators indicate improper use of supercomputer components. Tags describe the characteristics of the allocated computational resources and expand the information about the task properties. Based on tags and indicators, the HPC TaskMaster system makes inferences about the efficiency of the task.

The new version of the system introduces the concept of a *parameter*. Parameters are static information about the task, such as duration, number of allocated resources, task name, and so on. Now inferences are based on three entities: tags, indicators, and parameters. Using the parameter entity in the subsystem for inferences makes it possible to identify errors as:

- a suboptimal ratio of the number of CPUs and GPUs concerning a specific type of application software package;
- an incorrect time limit;
- use of an unsupported type of computing node to run tasks;
- and many others.

Besides, the paper introduces the concept of *ensembles of parameter ranges*, which makes it possible to create one inference in the system for different combinations of parameter values without duplicating the inferences for distinct parameter values.

The rest of the paper is organized as follows. General information about the HPC TaskMaster system is given in Sect. 2. In Sect. 3, we discuss related work. Section 4 describes the data used to analyze the task efficiency. Section 5 describes the entities developed in the HPC TaskMaster system and their meaning. A method for generating inferences based on previously described entities is considered in Sect. 6. Section 7 provides an example for a clear understanding of the analysis process of computational tasks for efficiency. Finally, Sect. 8 presents and describes a graph comparing the performance of HPC clusters with and without the HPC TaskMaster system.

2 Overview of the HPC TaskMaster System

The HPC TaskMaster system for monitoring the efficiency of tasks in a supercomputer center was developed in 2021 [8]. The system was constantly enhanced, and its functional capabilities increased [3,15]. Currently, the system provides the user with additional information about limits and quotas for using computational resources, the projects in which he participates, various statistics on the

performance of computations, and information on efficient and inefficient tasks. Project managers can monitor the launch of computational tasks for all project members. System administrators rely on a special interface to configure the system, view statistics on the efficiency of user tasks, and manage tags, indicators, and inferences in the system.

The current internal architecture of the HPC TaskMaster system consists of two main applications: the core and the cabinet [9]. The core application consists of the six modules enumerated below:

1) Mailing module: sends emails to users with notifications when inefficient tasks are detected or automatically canceled.
2) Cluster module: allows using data from another system on the HPC cluster for internal system functions. This module provides statistics on the HPC cluster utilization by users and projects.
3) Slurm module: gathers task data from the SLURM queue manager database.
4) Influx module: maintains a connection to an InfluxDB and collects task metrics.
5) Metrics module: performs different operations to aggregate metrics and prepare processed data for storage, including relational links to tasks.
6) Grafana module: generates dashboards for each task given its startup parameters and metrics availability and manages user access rights to these dashboards.

The cabinet application consists of the following two modules:

1) UserStats module: is responsible for generating various statistics on users and their tasks.
2) JobAnalysis module: evaluates aggregated metrics and other data for a task and assigns indicators, tags, and inferences to tasks according to this evaluation.

The main components of the HPC TaskMaster system are shown in Fig. 1. In this paper, we describe the changes and enhancement of the darker module.

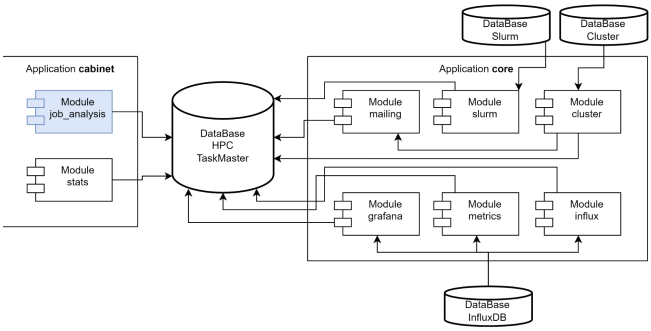

Fig. 1. Diagram of the HPC TaskMaster system main components

3 Related Work

Before the implementation of the HPC TaskMaster system on the cHARISMa HPC cluster, several existing monitoring systems were studied for matching with the HSE supercomputer.

The Automated performance analysis tools framework for HPC programs [6] is intended for applying specialized profilers and tracers to existing C++/Fortran code. The developers introduced 11 pre-configured configs for Intel Advisor, VTune, ScoreP, and others. The framework collected results are useful only for advanced users who develop their own parallel software and are interested in maximum optimization of the execution of algorithms. For ordinary users relying on proprietary software, the possibilities of using the framework are severely limited by the need to rebuild the software before each run.

The analytic platform for Savio supercluster [1] has goals very similar to ours. It defines job parameters, such as duration, job id, and job status, and depicts graphs of utilization with Grafana. This system has a disadvantage: it does not analyze job metrics, so it is not enough for the HPC TaskMaster aims.

For each job, the monitoring system for MPCDF [14] provides reports consisting of achieved performance in GFLOP/s, the memory bandwidth, the algorithmic intensity for each socket, and a dashboard with about 30 plots for CPU, GPU, network, filesystem, and software metrics. As the authors write, the system is still under development, specifically the data analytic module of the HPC monitoring system.

JobDigest is a system for analyzing the behavior characteristics of jobs run at Moscow State University's (MSU) HPC Center [12]. Dynamical and integral job characteristics are computed for each job upon completion. Each dynamic characteristic is represented by five values for every time interval: *min*, *max*, *min_avg*, *max_avg*, and *avg*. Some examples of dynamical job characteristics are CPU user load, load average, L1 cache misses, memory load, CPU nice load, CPU idle, etc. The integral job characteristics represent the average resource utilization and are built based on dynamic job characteristics. After completion, the job is assigned tags according to the integral characteristics and basic relevant information. The tags mark the scale of the job, partition, duration, details on resource utilization, and others. JobDigest has been tailored to suit the features of the OctoShell system and the specific architecture of the MSU HPC Center. Moreover, JobDigest works with the condition that only one job per computing node can be executed, while the cHARISMa HPC cluster allows for several tasks performed simultaneously. Therefore, the implementation of JobDigest as an analyzing system would be impossible for cHARISMa. In this regard, it was decided to develop a system with the required functionality: HPC TaskMaster. The following are some of the key differences between HPC TaskMaster and JobDiges:

1 The task analysis is carried out not only after the task completion but also during its execution. This makes it possible to detect inefficient tasks and cancel them, thereby providing free resources for more efficient tasks.

2 Inferences are generated for each task in the HPC TaskMaster system, not only based on indicators and tags (in JobDigest, categories and tags are similar entities) but also on ensembles of parameter ranges. This allows for a more accurate determination of the types of errors for various tasks. Another advantage associated with the availability of parameter ranges is that the HPC TaskMaster system can recommend the user to change the number and type of computational resources allocated to his task. By following the recommendations, users can improve the efficiency of their tasks.

4 Collected Data About Tasks

When launching and executing tasks on the high-performance computing cluster installed at the Higher School of Economics, the HPC TaskMaster system collects two types of data:

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

4.1 Task Parameters

Parameters are static data about the task. The list of parameters and their types is presented in Table 1.

Table 1. List of parameters

№	Parameter	Type
1	ID	Integer
2	Task name	String
3	Status	
4	Launch command	
5	Type of computing nodes	
6	Number of computing 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	

The set of parameters defined in the computational task is denoted by $P = \{p_i\}_{i=1}^{m}$, where m is the number of parameters.

4.2 Ranges of Parameter Values

The parameter values may vary from task to task. To operate with parameters, the *ranges of parameter values* (hereinafter referred to as *ranges*) d_i are defined in the system. The set of all ranges specified in the subsystem for inferences is denoted by $D = \{d_i\}_{i=1}^{w}$, where w is the number of ranges. A range is determined by the parameter of the computational task, its possible values, and the data type. Examples of ranges are shown in Table 2.

Table 2. Examples of ranges

№	Range	Limits	Type
1	Task duration	Min, Max	DateTime
2	Number of CPU cores		Integer
3	Number of GPUs		
4	Number of nodes		
5	Node type	Enumeration	String

By carrying out experimental launches of various types of tasks with different allocated resources [7], we chose the best launch parameters for the cHARISMa HPC cluster. For example, on the nodes of types a, b, and c, most Gromacs tasks run better on 16–20 CPU cores when using 1 GPUs, whereas it is better to use 40 CPUs when using 2 GPUs. On the node of type e, it is recommended to use 4 GPUs and at least 44 CPU cores. In addition, the recommended ratio of CPU and GPU resources is selected in such a way that the user cannot block the entire computing node. This situation can reveal whether the user uses 1 of 4 GPUs and all CPU cores on the node.

Thanks to the ranges of parameter values, the HPC TaskMaster system can recommend novice users a better choice of parameters for the efficiency of tasks using standard scientific packages.

4.3 Metrics

Table 3 shows the metrics collected by the HPC TaskMaster system during the execution of each task. The metrics form a time series θ_i. The set containing all time series is denoted by $\theta = \{\theta_i\}$.

The frequency of collecting metrics from the node is configured to obtain comprehensive information about the task without overloading the system with data collection and storage.

5 Data Processing

Aggregated metrics are calculated based on the collected data (see Sect. 5.1). Each task is assigned tags (see Sect. 5.2). Indicators are calculated according

Table 3. Collected metrics and the corresponding collecting frequency

№	Metrics	Frequency, seconds	Units of measurement
1	Usage of CPU cores by the user	10	percentage
2	Usage of CPU cores by the system		
3	GPU usage		
4	RAM usage		Megabyte
5	GPU memory usage		
6	InfiniBand usage by the node		
7	Amount of data read from the storage	30	
8	Amount of data recorded on the storage		
9	Amount of data read from the local disk		
10	Amount of data recorded on the local disk		
11	GPU power consumption	10	watt-hour

to the aggregated metrics. Tags can be generated according to the specified parameters. Indicators and tags are employed to assess the efficiency of tasks.

5.1 Aggregated Metrics

To simplify the analysis, aggregated metrics $\Lambda^k = (\lambda_1^k, \ldots, \lambda_c^k)$ are calculated for each time series [19]. These metrics include the minimum, maximum, average, median, and standard deviations. In addition to these aggregated metrics, the tuple Λ contains the average task load of each node individually and the average load of all used computing nodes [18].

5.2 Tags

Since task parameters are a heterogeneous set of data (integers, strings, dates), a system of tags is introduced to simplify the analysis. Tags indicate the task type, run time, and other task properties. Table 4 contains the list of all tags currently available in the system. Additional tags can be developed and implemented in the system.

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

5.3 Indicators

Indicators are used to simplify the work with aggregated metrics and bring them to the same type. Indicators are dimensionless values inversely proportional to the value of metrics. They take a value from 0 (if the allocated resources are

Table 4. List of tags

№	Tag
1	The task is running on 1 core
2	The number of CPUs is not a power of 2 (recommendation)
3	A small task is running on multiple nodes
4	An odd number of CPU cores
5	An odd number of GPUs
6	Imbalance in allocated resources
7	2d Blume task
8	Amber task
9	Abinit task
10	CP2K task
11	Jupyter Notebook task
12	HPCG task
13	GROMACS task
14	LAMMPS task
15	LINPACK task
16	MATLAB task
17	Quantum Espresso task
18	Singularity task
19	srun/salloc task
20	VASP task

fully used) to 1 (if otherwise). The value of the indicator l_j is calculated from the aggregated metric $\lambda_j^k \in \Lambda^k$ via the formula

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

where a_j and b_j are parameters determined by the settings and corresponding to the minimum and maximum possible value of the j-th element of the aggregated metrics.

The computed values of the indicators are placed in the tuple of indicators $L^k = (l_1^k, \ldots, l_f^k)$, where f is the number of indicators related to each task.

Table 5 presents the list of currently available indicators. The number of indicators for a specific task depends on the number of cores, computing nodes, and GPUs used.

6 Inferences

To simplify the interpretation of the results, a set of inferences $\Phi = \{\phi_i\}_{i=1}^n$, where n is the number of inferences, was introduced to the system. The inference

is the result of analyzing the data of the task completion. Ensembles of parameter ranges (see Sect. 6.1), tag conditions, and indicator values are defined for each inference. When all conditions are met, the inference is assigned to the task. Several inferences can be applied to a task at once.

6.1 Ensembles of Parameter Ranges

For the convenience of forming inferences and to avoid creating duplicate inferences for the same type of problems for tasks with different parameter values, we introduced *parameter ranges* into the system (see Sect. 4.2). The parameter ranges are combined into *ensembles of parameter ranges* (hereinafter called *ensembles*) $\gamma^k = (d_1^k, d_2^k, \ldots, d_w^k)$. The set of all ensembles of parameter ranges in the inferences is denoted by $\Gamma = \{\gamma_i\}_{i=1}^{s}$, where s is the number of ensembles.

Let us introduce a function $f(\gamma, P^*)$ that returns 1 if the task parameters from the set P^* fall within the corresponding ranges of the ensemble γ, and is 0 otherwise. We define the function by a product of Kronecker deltas, namely,

$$f(\gamma, P^*) = \prod_{i=1}^{w} \delta_i = \begin{cases} 1 & p_i \in d_i^\gamma, \\ 0 & p_i \notin d_i^\gamma, \end{cases} \quad p_i \in P^*,\ d_i^\gamma \in D, \tag{2}$$

Table 5. List of indicators

№	Indicator
1	Lustre: high recording load (warning)
2	Lustre: high read load (recommendation)
3	SSD: too high write load (warning)
4	SSD: high read load (recommendation)
5	Omissions in metrics
6	GPU: low correlation of load graphs
7	CPU: low correlation of core load graphs
8	GPU: high idle rate (all time)
9	CPU: high idle rate (all time)
10	GPU: low average load in the last hour (warning)
11	GPU: low average load in the last 3 h (task cancelation)
12	CPU: low average load in the last hour (warning)
13	CPU: low average usage over the last 3 h (task cancelation)
14	RAM: low RAM usage (recommendation)
15	GPU: low graphics memory usage (recommendation)
16	GPU: low average usage
18	CPU: low average load
17	CPU: low load of individual cores
19	CPU: good average load

where $P^* \subset P$ contains the task parameters whose ranges are defined in the set D, and w is the number of ranges for which the task parameters are checked.

In the HPC TaskMaster system, the creation of ensembles in the subsystem for inferences is implemented via a web interface. An example is shown in Fig. 3.

6.2 Inference Forming

In the first stage, the indicators and tags from both the task and each inference are compared. We denote the union of tuples of indicators and tags related to the computational task k by

$$N^k = (l_1^k, \ldots, l_f^k, \tau_1^k, \ldots, \tau_h^k).$$

Let Ω_i be a set of conditions for the inference ϕ_i to the indicators and tags similar to the tuple N^k.

Then the task k can be mapped to a tuple of its inferences R^k based only on indicators and tags:

$$R^k = \left(\prod_{j=1}^{f+h} \mathbf{1}_{N^k}(w_j) \right)_{i=1}^{n}, \quad w_j \in \Omega_i, \tag{3}$$

where w_j is the value of a specific condition from the set of conditions Ω_i, n is the number of inferences in the system, and $\mathbf{1}_{N^k}(w_j)$ is an indicator function [4] equal to 1 if the value of the inference condition is included in the task condition.

Next, we use function (2) to check the values of task k parameters if they are in the corresponding ranges (see Sect. 6.1) for each ensemble in the inference. Another condition is that the value of the corresponding inference in the tuple R^k is equal to 1. Then the computational task k is put in correspondence to the set of its inferences C^k,

$$C^k = \left\{ \phi_i \in \Phi : R_i^k = 1 \wedge \exists j \in [1, s] : f(\gamma_i, P^k) = 1 \right\}, \quad i \in [1, n], \tag{4}$$

where s is the number of ensembles in the inference, γ_j is the specific ensemble, and P^k is a set that contains those task parameters whose ranges are defined in the set of ranges D.

The set C^k consists of inferences corresponding to the task k.

Figure 2 shows a diagram of the inference formation and its components.

Inefficient user tasks can be canceled by the HPC TaskMaster system based on inferences. If a computational task does not load the allocated resources above the threshold, it is assigned an inference that leads to the cancelation of the task. Three hours before the cancelation the user receives a notification of low task efficiency. If the user takes action and the efficiency of the HPC cluster increases, the task continues to be executed. Otherwise, the task is canceled.

When inefficient tasks are removed, the released resources are allocated to new tasks from the queue. Thus, the HPC TaskMaster system may ultimately lead to a situation when only efficient tasks are executed on the HPC cluster. This approach ensures the best real performance on the supercomputer.

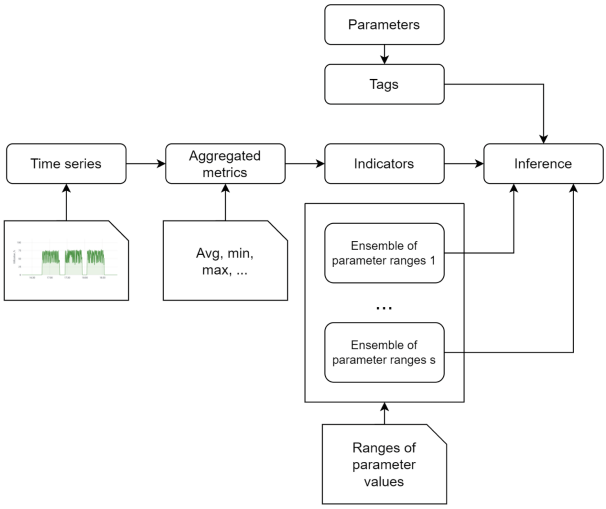

Fig. 2. A diagram for generating inference according to indicators, tags, and ensembles

6.3 Hierarchy of Inferences

Each inference, when created, is assigned a priority, whose value determines its relevance compared to other inferences. In the HPC TaskMaster system, the user is shown the inference with the highest priority out of all suitable inferences.

7 Example

Let us consider two computational tasks, № 1 and № 2, executed on the cHARISMa HPC cluster. Table 6 represents the parameters of the given tasks.

Table 6. Parameters of the tasks

№	Parameter	Task № 1	Task № 2
1	ID	1	2
2	Task name	eq363TMOS	eq364TMOS
3	Status	completed	completed
4	Launch command	sbatch 02_eq.sh	sbatch 03_eq.sh
5	Type of computing nodes	type_a	type_b
6	Number of computing nodes	1	1
7	Number of CPU cores	16	3
8	Number of GPUs	2	1
9	Exit code	0	0
10	User ID	***	***
11	Project ID	123	124
12	Start date and time	November 20, 2023 11:43:07	November 20, 2023 12:24:13
13	End date and time	November 20, 2023 11:52:19	November 20, 2023 12:32:46

Based on the parameters from Table 6 and the tags from Table 4, task № 1 is assigned the tag "GROMACS task". The tuple of the task tags would be written as $T^1 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$. Similarly, two tags are assigned to task № 2: "GROMACS task" and "The number of CPUs is not a power of 2 (recommendation)". The tuple of the task tags would be written as $T^2 = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$.

The values of task № 1 indicators calculated based on the aggregated metrics are given in Table 7.

Table 7. List of indicators for task № 1

N	Indicator	Value
Computing node **cn-011**		
1	Core 1 utilization	0.706
⋮	⋮	⋮
16	Core 16 utilization	0.695
17	GPU (ID=3) utilization	0.853
18	GPU (ID=4) utilization	0.846
19	GPU (ID=3) memory usage	0.779
20	GPU (ID=4) memory usage	0.778
Summary		
21	Average load of cores on the node **cn-011**	0.699
22	Average load of GPUs on the node **cn-011**	0.849

Correspondingly, the values of task № 2 indicators calculated based on the aggregated metrics are given in Table 8.

Table 8. List of indicators for task № 2

№	Indicator	Value
Computing node cn-019		
1	Core 1 utilization	0.011
2	Core 2 utilization	0.006
3	Core 3 utilization	0.004
4	GPU (ID=0) utilization	0.206
5	GPU (ID=3) memory usage	0.127
Summary		
6	Average load of cores on the node **cn-019**	0.007
7	Average load of GPUs on the node **cn-019**	0.016

We obtain the tuple $N^1 = (l_1, \ldots, l_{22}, \tau_1, \ldots, \tau_{20})$ for task № 1 and, respectively, the tuple of conditions $N^2 = (l_1, \ldots, l_7, \tau_1, \ldots, \tau_{20})$ for task № 2. We consider an example of an assignment of inferences to tasks with five inferences. In Fig. 3, we provide an example of ensembles of parameter ranges for the inference number 2. The analysis results for tasks № 1 and № 2 are shown in Table 9.

Based on the tuple N^1, the set of inferences $C^1 = \{\phi_1, \phi_2, \phi_4\}$ is assigned to task № 1. Similarly, based on the tuple N^2, the set of inferences $C^2 = \{\phi_2, \phi_4\}$ is assigned to task № 2.

The inference with the highest priority from the set C^1 is shown. In this case, it is ϕ_2. Likewise, the inference ϕ_2 is shown for task № 2.

Table 9. Checking of inferences for tasks № 1 and № 2

ϕ_i	Inference	Priority	Conditions	Check for task № 1	Check for task № 2
1	The allocated resources are distributed unevenly among computing nodes	4	$\tau_7 = 1$	No	No
2	When using GPUs, Gromacs uses fewer cores than recommended in the manual.	5	$l_i \leq 1, i \in [1, f]$	Yes	Yes
			$\tau_{14} = 1$	Yes	Yes
			$\gamma_1 =$ (Nan, [1, 19], [1, 1], Nan, $[a, b, c]$)	No	Yes
			$\gamma_2 =$ (Nan, [1, 39], [2, 2], Nan, $[a, b, c]$)	Yes	No
3	Inefficient usage of Jupiter Notebook	7	$\tau_{12} = 1$	No	No
			$\gamma_1 =$ ([00 : 30 : 00, ∞], Nan, Nan, Nan, Nan)	No	No
			$l_i \geq 0.5, i \in [1, f]$	Yes	No
4	The task does not use the GPUs efficiently enough	10	$l_i \geq 0.5, i \in [1, f]$	Yes	Yes
			$\gamma_1 =$ ([00 : 05 : 00, ∞], Nan, Nan, Nan, Nan)	Yes	Yes
5	The task does not use the CPUs efficiently enough	11	$l_i \geq 0.5, i \in [1, f]$	Yes	No
			$\gamma_1 =$ ([00 : 00 : 00, 01 : 30 : 00], Nan, Nan, Nan, Nan)	Yes	Yes

8 The Impact of the HPC TaskMaster on the HPC Cluster Performance

The implementation of the HPC TaskMaster system and the enhancement of the data analysis subsystem increased the efficiency of the cHARISMa HPC cluster. The reason for this improvement is the cancelation of tasks according to their inference. For example, the waiting time for tasks in the queue has been reduced by a factor greater than 5.

The graph in Fig. 4 shows the dynamics of the average waiting time for supercomputer resources running computational tasks. The yellow line on the

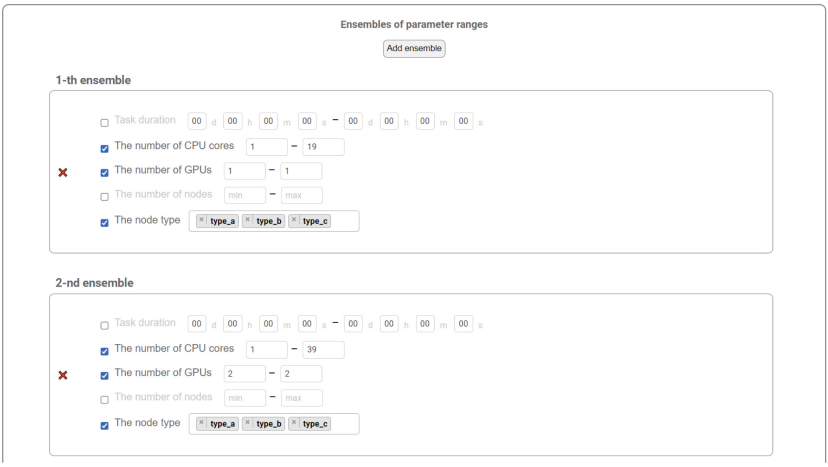

Fig. 3. Web interface for the ensembles in the HPC TaskMaster system

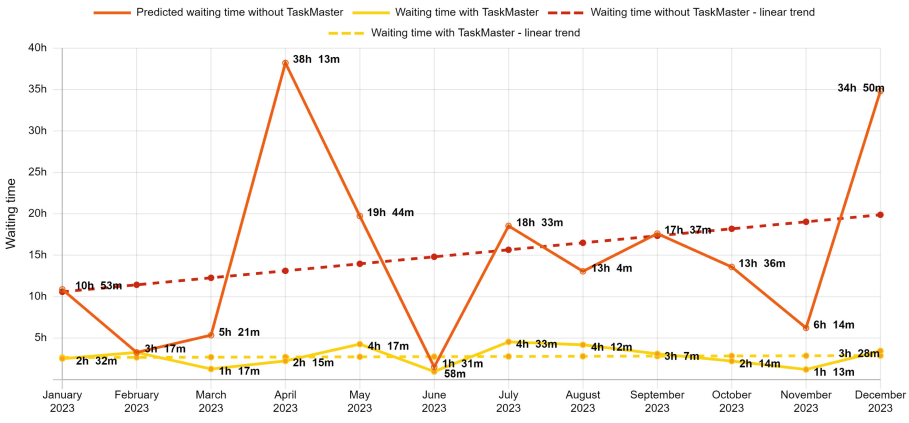

Fig. 4. The average waiting time for tasks in the queue

graph represents the real average waiting time for a task on the cHARISMa HPC cluster. The red line corresponds to the theoretical waiting time for resources on a cluster without the HPC TaskMaster task-performance monitoring system. When calculating the theoretical waiting time, all canceled inefficient tasks are considered.

9 Conclusions

The HPC TaskMaster system has been put into operation and is now successfully handling task-flow optimization. The system automatically cancels inefficient tasks, informs users about inefficient use of GPU and CPU resources, and gives recommendations on the correct allocation of computational resources to execute their tasks.

The enhancement of the task data analysis subsystem has been successfully implemented in the HPC TaskMaster system, and the inference subsystem effectively assigns inferences to tasks whose problems depend on parameters.

The HPC TaskMaster system is an open-source project [5] and is available for installation on any HPC cluster. In 2022, the HPC TaskMaster system received a certificate of state registration for computer programs. In 2023, it was included in the Russian Software Registry [13].

Acknowledgments. The research was supported in part by computational resources of HPC facilities at HSE University [8].

References

1. Chan, N.: A resource utilization analytics platform using Grafana and Telegraf for the Savio supercluster. In: PEARC '19: Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), pp. 1–6 (2019). https://doi.org/10.1145/3332186.3333053
2. Chulkevich, R.A., Kozyrev, V.I., Kostenetskiy, P.S., Raimova, A.A.: Implementation of NVIDIA GPUDirect technology on the HSE university HPC cluster. In: Russian Supercomputing Days: Proceedings of the International Conference, pp. 186–194 (2023)
3. Chulkevich, R.A., Kozyrev, V.I., Shamsutdinov, A.B., Kostenetskiy, P.S.: Comparison of the performance of a parallel storage supercomputer with different versions of the LUSTRE file system. In: Russian Supercomputing Days: Proceedings of the International Conference, pp. 159–161 (2022)
4. Demina, M.V., Kudryashov, N.A.: Point vortices and classical orthogonal polynomials. Regular Chaotic Dyn. **5**, 371–384 (2012). https://doi.org/10.1134/S1560354712050012
5. HPC TaskMaster open source repository. GitLab. https://git.hpc.hse.ru/open-source/hpc-taskmaster
6. Keiffa, M., Voigta, F., Fuchsa, A., Kuhnb, M., Squara, J., Ludwigc, T.: Automated performance analysis tools framework for HPC programs. Procedia Comput. Sci., 1067–1076 (2022). https://doi.org/10.1016/j.procs.2022.09.162

7. Kondratyuk, N., Nikolskiy, V., Pavlov, D., Stegailov, V.: GPU-accelerated molecular dynamics: state-of-art software performance and porting from Nvidia CUDA to AMD HIP. Int. J. High Performance Comput. Appl. **35**, 312–324 (2021). https://doi.org/10.1177/10943420211008288
8. Kostenetskiy, P.S., Chulkevich, R.A., Kozyrev, V.I.: HPC resources of the higher school of economics. J. Phys. Conf. Ser. **1740**, 012,050 (2021). https://doi.org/10.1088/1742-6596/1740/1/012050
9. Kostenetskiy, P.S., Chulkevich, R.A., Kozyrev, V.I., Shamsutdinov, A.B., Antonov, D.A.: HPC taskmaster - task efficiency monitoring system for the supercomputer center. Commun. Comput. Inf. Sci. **1618** (2022). https://doi.org/10.1007/978-3-031-11623-0_2
10. Kostenetskiy, P.S., Shamsutdinov, A.B., Chulkevich, R.A., Kozyrev, V.I.: HPC taskmaster - a system for monitoring the effectiveness of HPC cluster tasks. In: Russian Supercomputing Days: Proceedings of the International Conference, pp. 18–25 (2021)
11. Kostenetskiy, P.S., Shamsutdinov, A.B., Chulkevich, R.A., Kozyrev, V.I.: Development of a subsystem for analyzing the efficiency of computing resources for the HPC taskmaster system. In: Parallel Computational Technologies (2023)
12. Nikitenko, D., et al.: JobDigest - detailed system monitoring-based supercomputer application behavior analysis. In: Russian Supercomputing Days: Proceedings of the International Conference, pp. 185–198 (2017)
13. Russian Software Registry entry No18920 from 05.09.2023 for HPC TaskMaster - Task Efficiency Monitoring System for the Supercomputer Center. https://reestr.digital.gov.ru/reestr/1765708/
14. Stanisic, L., Reuter, K.: MPCDF HPC performance monitoring system: enabling insight via job-specific analysis. Lect. Notes Comput. Sci. **11997**, 613–625 (2020). https://doi.org/10.1007/978-3-030-48340-1_47
15. Voevodin, V.V., et al.: Administration, monitoring and analysis of supercomputers in Russia: a survey of 10 HPC centers. Supercomputer Front. Innov. **3** (2021). https://doi.org/10.14529/jsfi210305
16. Voevodin, V.V., Nikitenko, D.A.: Recurrent monitoring of supercomputer noise. Supercomputing Front. Innov. **10**, 27–35 (2023). https://doi.org/10.14529/jsfi230304
17. Voevodin, V.V., Shaikhislamov, D.I., Nikitenko, D.A.: How to assess the quality of supercomputer resource usage. Supercomputing Front. Innov. **9**, 4–18 (2022). https://doi.org/10.14529/jsfi220301
18. Zymbler, M.L., Goglachev, A.A.I.: Fast summarization of long time series with graphics processor. Mathematics **10** (2022). https://doi.org/10.3390/math10101781
19. Zymbler, M.L., Kraeva, Y.A.: High-performance time series anomaly discovery on graphics processors. Mathematics **11** (2023). https://doi.org/10.3390/math11143193