DOI: 10.14529/jsfi220301

How to Assess the Quality of Supercomputer Resource Usage

Vadim V. Voevodin[1], *Denis I. Shaikhislamov*[1], *Dmitry A. Nikitenko*[1]

Supercomputer is an exceptionally valuable computational resource and it must be used as efficiently as possible. However, in practice, the efficiency of its usage leaves much to be desired. There are various reasons for this. One of the main ones is the low performance of user applications, but users themselves are often not aware of the presence of performance issues in their programs. Therefore, it is necessary for administrators of a supercomputer to be able to constantly monitor the performance and behavior of all running jobs. However, the problem is that the commonly used metrics for assessing the quality of resource consumption (such as CPU or GPU load, the amount of bytes transferred over the MPI network, etc.) are often far from being convenient and accurate. This paper describes the implementation and evaluation of the previously proposed assessment system, which, in our opinion, makes it possible to significantly ease the task of properly evaluating the quality of the supercomputer resource usage. We also touch upon another topic related to the assessment of the quality of using HPC resources — organization of HPC resource provisioning.

Keywords: supercomputing, high-performance computing, performance analysis, monitoring, workload analysis, resource utilization, resource provisioning.

Introduction

Many modern supercomputers are inefficient. If we study, in detail, the use of the computational resources of a supercomputer, it often turns out that a significant part of these resources is either idle, or poorly utilized, or wasted (for example, in the case of an incorrect program launch). Over time, the situation does not get better: the constant complication of supercomputer architecture leads to an increase in its overall performance, but efficiency often drops as it becomes more and more difficult to properly utilize all the available hardware capabilities.

This is exacerbated by the fact that many supercomputer users initially come from not HPC-related areas (chemistry, physics, medicine, etc.) and therefore do not have sufficient theoretical knowledge and practical skills in writing highly efficient parallel applications [7]. Moreover, HPC area is growing quite fast [2], so more and more new specialists for various scientific fields start using supercomputers. With that, if a user does not pay for the consumed supercomputer node-hours, then s/he is not always motivated to care much about the performance of the applications. For example, if a user runs a job at night, from his/her point of view it may not make much difference whether it will run for 4 or 6 hours.

In such a situation, it is important for supercomputer administrators to control the efficiency of the operation themselves. And for this, it is necessary to constantly monitor and analyze the quality of its functioning, including the entire flow of running applications. And, at this moment, in practice, an unpleasant situation often arises: administrators are usually able to collect and store a wide variety of data on the behavior and performance of supercomputing applications, but it is not clear what to do with this data and how to extract useful information from it? In particular, how to understand which jobs have performance issues? Common metrics usually used for that (like CPU load, load average, frequency of LLC cache misses, or amount of bytes sent over MPI network) in many cases are not so insightful and do not help much in promptly detecting performance issues.

[1]Lomonosov Moscow State University, Moscow, Russian Federation

For these purposes, we are developing an assessment system that is designed to quickly and accurately analyze the quality of the supercomputer resources usage. These assessments are needed for the initial analysis, which allows understanding, in general, which jobs have low efficiency and therefore need to be paid attention to. It is assumed that subsequent detailed analysis of the selected application, needed to determine the root causes of performance degradation and ways to eliminate them, should be performed using existing analysis tools, such as profilers, debuggers, etc.

We also draw attention to the problems of efficient HPC resource provisioning as a necessary part of the whole computing workflow. We have interviewed five large HPC centers in Russia and a few European ones to investigate the most concerning questions of HPC centers management and resource provisioning. In this paper, we give a short summary of these surveys regarding the observed problem.

The main contribution of this paper is the description of the methods for implementing previously proposed assessments in practice, as well as the demonstration of the applicability of these assessments using real-life collected statistics and interesting examples. Another contribution is a ranked list of the most important questions regarding HPC resource provisioning, built according to the survey of the largest HPC centers in Russia and a few European ones.

The rest of the paper is organized as follows. Section 1 describes our background, briefly presenting our previously proposed assessment system. Section 2 is devoted to the implementation of methods for collecting needed data and computing assessments. In Section 3, the analysis of some real-life statistics and specific examples collected on Lomonosov-2 supercomputer is performed. Section 4 describes machine learning techniques planned to be used for expanding the applicability of the proposed solution. Section 5 is aimed at questions of efficient resource provisioning and its assessment. Conclusions are described in the last section.

1. Previously Proposed Assessment System

In the previous paper [17], a description of the proposed assessments is given. It was decided that for each supercomputer job assessments should evaluate how "inefficiently" or "poorly" this job is using the given computational resource. In our case, we decided to assess how much working with the selected type of resource interferes with useful computations (which can be performed by CPU or GPU). If such interference is high, this means that a processor is far from being fully utilized, waiting for the execution of operations with the specified resource.

Assessments were developed for 6 types of resources — CPU, memory, MPI network, I/O, GPU and GPU memory, and specific formulas for their calculation were proposed (only general ideas were proposed for two GPU-related assessments). These assessments are briefly presented in Tab. 1. Hereinafter, assessment for a certain type of resources will be denoted as $\text{score}_{\text{type}}$ (for example, $\text{score}_{\text{cpu}}$).

The "Assessment based on" column specifies what idea underlies the proposed method for calculating each assessment. It can be seen that CPU and Memory assessments are based on Top-down approach developed by Intel [4, 20] (however, we have not seen the application of this approach not for one specific application but for the entire job flow, as done in our study). $\text{score}_{\text{mpi}}$ aggregates information about all MPI-related performance issues automatically detected by TASC [13]; the same is true for I/O assessment. Specific formulas for calculating these metrics were given in [17]. No specific formulas have been previously given for GPU-related assessments, but this has been done in this paper, see Section 2.

Table 1. A brief description of previously proposed assessments

Resource type	Assessment based on	Description
CPU	Top-down approach	Estimates the fraction of the CPU time when it was not fully utilized performing useful computations.
Memory	Top-down approach	Estimates the fraction of time that the processor was somehow idle, waiting for data from memory to be read or written.
MPI network	TASC	Evaluates the number and criticality of MPI-related issues causing a job to spend execution time on data exchange and not computations.
I/O	TASC	The same as above, but for I/O network.
GPU	—	GPU analogue of CPU assessment.
GPU memory	—	GPU analogue of Memory assessment.

A review of related work is given in the previous paper. Here, we only briefly note that at the moment no studies have been found in which analogues for the proposed assessments were proposed. However, it should be noted that some of these assessments are based on existing research, in particular, the aforementioned Top-down approach.

2. Implementation of Methods for Calculating Assessments

The formulas for calculating assessments $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ given in the previous paper have been slightly improved. So, a slightly more precise formula is now used to calculate $\text{score}_{\text{mem}}$, and the formula for the $\text{score}_{\text{cpu}}$ has been changed to comply with the general rule: each score must take values from 0 to 100, where 0 is the best value and corresponds to no interference with useful computations, while 100 is the worst value, meaning that usage of specified resource degrade efficiency all the time. The new formulas are shown in (1) and (2). All specified names in formulas are the processor sensors (in Linux `perf` notation).

$$\mathbf{score_{cpu}} = 100 - 100 * uops\_retired.retire\_slots / (2 * cpu\_clk\_unhalted.thread\_any) \quad (1)$$

$$\mathbf{score_{mem}} = (min(cpu\_clk\_unhalted.thread,\ cycle\_activity.stalls\_ldm\_pending) + resource\_stalls.sb) \,/\, cpu\_clk\_unhalted.thread \quad (2)$$

We tested this implementations on the Lomonosov-2 supercomputer installed at Lomonosov Moscow State University. All the data needed to calculate these assessments on Lomonosov-2 has been collected using DIMMon monitoring system [14]. For these purposes, a new DiMMon module was developed which collects data from the required performance monitoring counters (PMC). It should be noted that at this stage we encountered unexpected technical challenges. To obtain all the needed data, it was necessary to start using the multiplexing mode, since Intel Xeon E5-2697 v3 and Intel Xeon Gold 6126 processors used at Lomonosov-2 (like many other modern processors) allow simultaneously collecting data from only 4 sensors, and we needed a

total of 8 sensors (5 for assessments, 3 for other purposes). And it turned out that the PAPI [15] library, which we previously used to obtain data from processor counters, gives quite a noticeable overhead working in the multiplexing mode, thereby slowing down user applications.

Starting to investigate this topic, it turned out that at the moment there was no detailed study of the overhead caused by multiplexing, so we did this study ourselves [19]. It showed that the least overhead is obtained when using the LIKWID [9] library, and this is the solution we started using in practice on the Lomonosov-2 supercomputer. However, it should be noted that even in this case, the overhead can still be noticeable: on average, application execution time slows down by 2.78%, rarely reaching a maximum of more than 10%. With this in mind, it was decided to collect an extended set of sensors (i.e. using multiplexing mode) for every third job, so that the average slowdown of all user applications would be less than 1%. In order to obtain $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ for other jobs as well, we plan to use machine learning methods, which is discussed in Section 4 of this paper.

The implementation of $\text{score}_{\text{mpi}}$ and score_{io} has not changed: each score aggregates information on all MPI-related (or I/O-related) performance issues automatically detected using primary analysis implemented in TASC [12]. The only change is a technical one: the assessments are now collected automatically for all jobs using TASC workflow, which greatly simplifies their calculation and analysis.

These scores are not so precise as $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$, since they build on only those issues that can be detected using TASC, and automatic detection in TASC is quite limited due to the general constraints of constant performance monitoring and analysis. But, unlike mentioned scores, they are collected for all the jobs running on the supercomputer. We are conducting a study of possible approaches to obtain more accurate assessment for MPI, for example, based on the metrics proposed in the PoP project [3]. We have implemented a software prototype using the PnMPI [10], a lightweight tool for automatically collecting information about MPI calls for all running HPC jobs. However, this prototype, although making it possible to obtain a noticeably more accurate assessment, shows too high overheads, therefore does not currently allow applying it in practice. We are working on further improvement of it, trying to reduce the overhead.

In case of GPU, we want to develop assessments similar to $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$, i.e. Top-down-related ones. There are studies suggesting Top-down metrics for GPU (for example, [21]), but we need to calculate assessments for all jobs and therefore can not collect a lot of data (as not to increase the overhead), so it was necessary to strike a balance between accuracy and the amount of data collected. The current proposed formulas for $\text{score}_{\text{gpu}}$ and $\text{score}_{\text{gpumem}}$ are shown in (3) and (4).

$$\mathbf{score_{gpu}} = 100 - 100 * Eligible\_Warps \,/\, Theoretical\_Occupancy \tag{3}$$

$$\mathbf{score_{gpumem}} = 100 * (Memory\_Dependency + Memory\_Throttle) \,/\, Active\_Warps \tag{4}$$

We intended to collect this data using a module of the DiMMon monitoring system, with the use of CUPTI [1] supported in PAPI. However, it turned out that apparently in our case it is technically impossible: most of our GPUs do not support external PMC data collection (capability >= 7.5 is required). There are other ways to collect the required data, such as using NVIDIA Nsight, but these solutions are likely to lead to significant overheads. At the moment, we are investigating ways for solving this technical problem.

3. Analyzing Real-life Statistics and Examples

In this section, we will show how suggested assessments can be used in practice to obtain useful and interesting insights. For this purpose, we analyzed all jobs that were executed on the Lomonosov-2 supercomputer during September 2022.

During this period, user run 14920 jobs in total. $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ were calculated for 9.5% of them. As mentioned before, data for these assessments is collected for every third job, but some other factors should be taken into account: there are many very short jobs — less than 1 minute — for which no monitoring data could been collected (usually, test or incorrect launches) due to data collection granularity; also, sometimes, monitoring system fails to collect all needed data. This explains why data was collected for only ∼10% of jobs. $\text{score}_{\text{mpi}}$ and score_{io} can be potentially collected for any job, but they also rely on the presence of monitoring data. Moreover, there are many cases when an application does not use communication network at all or use it efficient enough (thus no MPI-related issues are detected), so the value of $\text{score}_{\text{mpi}}$ is non-zero for 17.15% of jobs (where at least MPI-related issue was detected), while score_{io} is non-zero in 0.34% cases. Such a situation with score_{io} indicates that we probably should revise I/O issue detection in TASC, since it is unlikely that any kind of such issues occur so rarely.

Figure 1 shows distribution of $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ for all jobs during September 2022. Each dot corresponds to a specific job, axis X is $\text{score}_{\text{cpu}}$ for a job, axis Y is $\text{score}_{\text{mem}}$. The general trend is noticeable — $\text{score}_{\text{cpu}}$ grows together with $\text{score}_{\text{mem}}$, which is generally logical, since the more memory interferes, the less the processor is able to perform useful computations. Also, it can be seen that "good" $\text{score}_{\text{cpu}}$ values, i.e. values that are close to zero, are not common. This is also not surprising, since the actual performance when using modern processors is very rarely close to peak performance. On the contrary, values of $\text{score}_{\text{mem}}$ stay in the low range of 10-40%. This means that memory usage on average does not interfere much with useful calculations. However, we can see jobs with $\text{score}_{\text{mem}}$ of almost 100%, meaning that usage of memory degrades application performance constantly during job execution. We can also notice, for example, that there is a job showing $\text{score}_{\text{cpu}}$ of ∼95%, but $\text{score}_{\text{mem}}$ is very low in this case with less than 10% (right bottom dot). This means that this program definitely has serious performance issues, but these issues are not related to memory usage.

Such a representation allows getting a general idea about the quality of usage of the selected resources in general and, in some cases, to identify anomalies.

Let us look at the available statistics from a different angle. Figure 2 shows top 25 jobs during the selected time period (September 2022) with the highest assessments overall (ranked by the sum of all assessments). With this we want to look at jobs with the most noticeable performance issues. Jobs are shown along the X axis, indicating the name of the user who ran them. Axis Y shows the value of three assessments — $\text{score}_{\text{cpu}}$, $\text{score}_{\text{mem}}$ and $\text{score}_{\text{mpi}} + \text{score}_{\text{io}}$ (last two are combined for convenience, all the more so as the second assessment is almost always zero, and both assessments are calculated in a similar way and therefore can be summed).

We can see that values of all these estimates range from 40% to 80%, and there seems to be a certain trend: the farther — the lower the $\text{score}_{\text{mem}}$ is, while it is not observed for two other assessments.

The most interesting fact in this figure is that 19 out of 25 jobs belong to one user (user1), as it can be seen in the X-axis label. This clearly indicates that performance issues have occurred repeatedly in the jobs of this user, and judging by the number of such jobs, this does not look

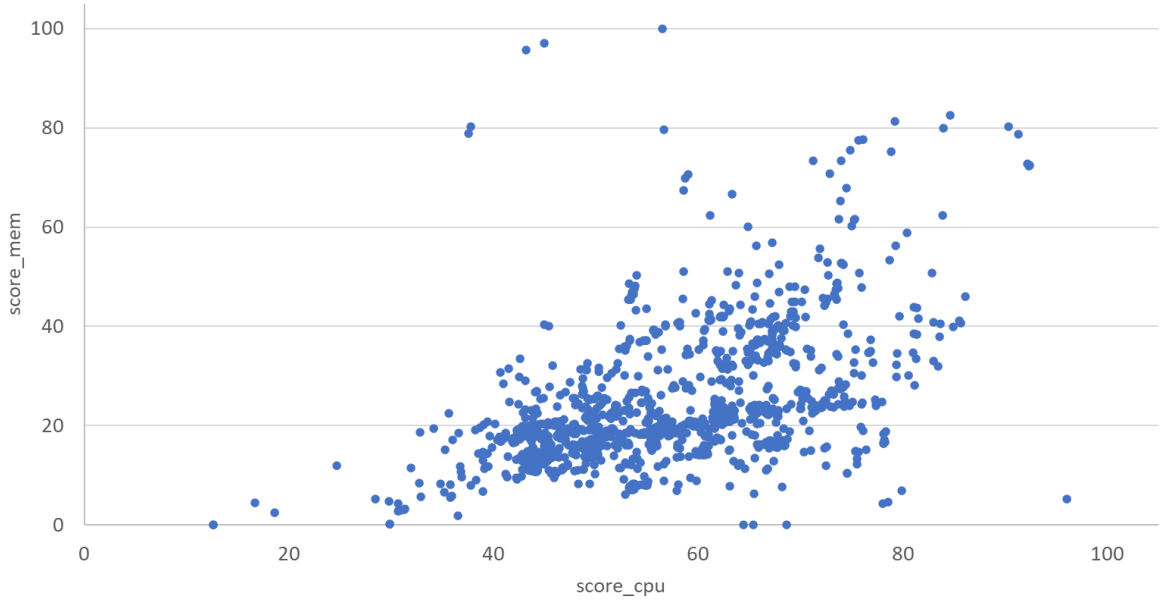

Figure 1. Distribution of $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ for all jobs during September 2022

like single random cases, but the presence of constant issues. Further analysis showed that these jobs in fact did show very low performance, and this was due to the fact that the user ran a series of test runs with intensive use of collective MPI-operations, which led to such bad assessments. In this case, the nature of the tests being run does not allow improving the use of computational resources, however, given that these tests are short and consume few node-hours, this is not a noticeable problem at the level of the entire supercomputer.

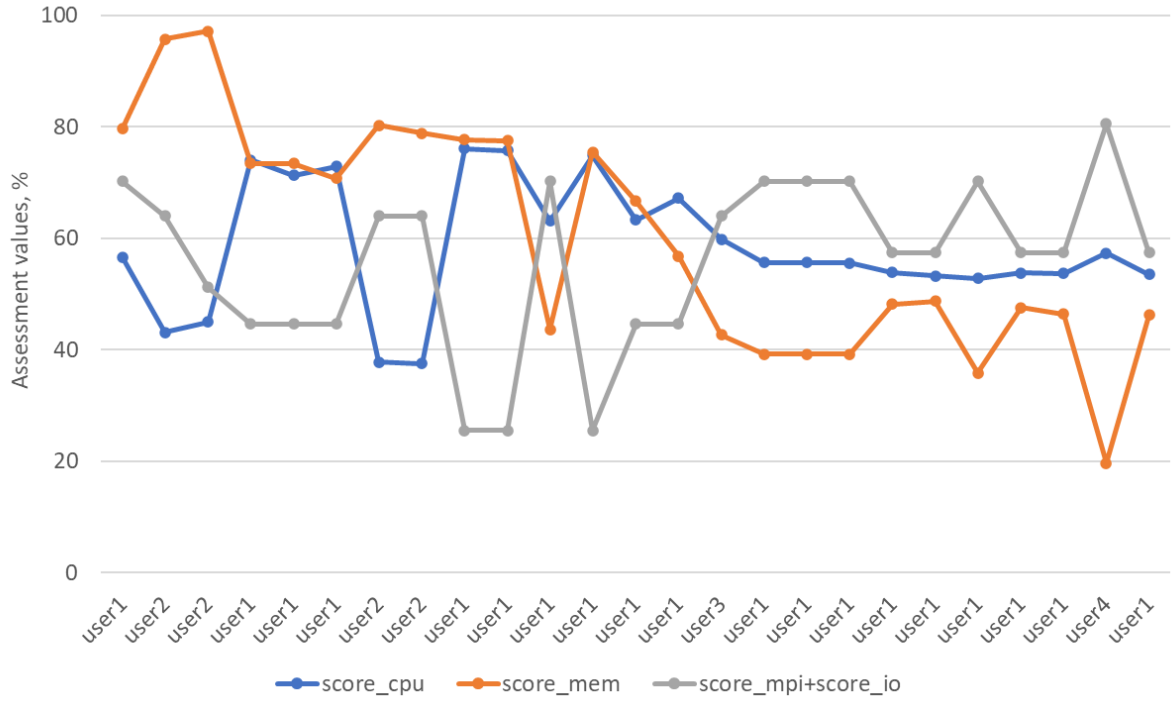

Figure 2. Top25 jobs with the highest sum of assessments

Considering the above, system administrators should pay attention to job #15 launched by user3: this job is much more computationally expensive than all the others presented in the Fig. 2. At the moment, this user has had only few launches, but they all consume noticeable amount of node-hours and all show bad assessment values, so it is worth paying attention to them: if their number increases, it is worth contacting the user and trying to find out what is the reason for this behavior. Moreover, all such jobs have a high $\text{score}_{\text{mpi}} + \text{score}_{\text{io}}$ score and they occupy quite a lot of compute nodes, which means that the task of optimizing MPI usage should be of the priority.

Now let us move on from considering individual jobs to studying aggregated information on individual users. In Fig. 3, top 50 users based on their weighted $\text{score}_{\text{cpu}}$ average during September 2022 are shown. Blue line shows weighted $\text{score}_{\text{cpu}}$ average for specific users. Here, weights are node-hours, i.e., value for each user is calculated as `sum(score_i*nodehour_i)/sum(nodehour_i)`, where `score_i` is $\text{score}_{\text{cpu}}$ for i-th job of selected user, and `nodehour_i` is the amount of node-hours i-th job has consumed. Orange dots correspond to total node-hours consumed by each user during the selected time period (note that these values are shown on the secondary Y axis).

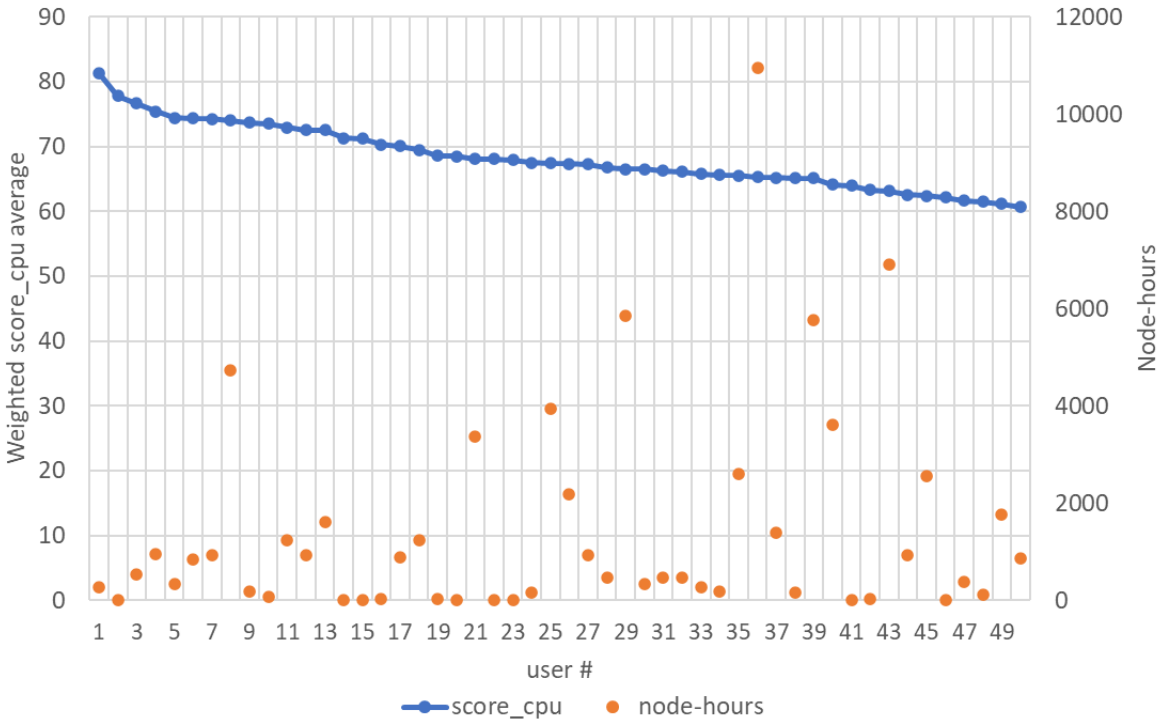

Figure 3. Top user ranking based on weighted $\text{score}_{\text{cpu}}$ average

Such information can be useful in different ways. For example, we noticed that users #1, #4, #5 and #11 are from the same project. Moreover, they all actively use LAMMPS software package [16], which suggests that these users are highly inefficient at using this package (because many other LAMMPS users have noticeably better $\text{score}_{\text{cpu}}$ values). Further analysis has shown that many of job launches under discussion seems to be incorrect: they have ended with FAILED finish state, e.g. one third of user #1 jobs has this state. This most likely means that users of this project have problems with the correct and efficient usage of the LAMMPS package, and system administrators should pay attention to this. It is interesting to note that the common

metrics for evaluating the processor load, CPU user load or load average, for all the considered jobs show almost optimal values, that is, it would be impossible to detect this situation using the standard approach.

These are just a few examples of how the proposed assessments can be applied in practice. The proposed solution is currently fully operational and available on the Lomonosov-2 supercomputer; in the future, we plan to integrate their use within the standard operating procedure for system administrators.

4. Using Machine Learning to Predict Assessment Values

It has been mentioned earlier that $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ are collected not for all jobs. But it would definitely be great to evaluate these assessments for as much jobs as possible. In order to achieve that, it is planned to apply machine learning methods. Previously, two methods for detecting similar supercomputer applications have been developed at the Research Computing Center of Lomonosov Moscow State University [11]. The first method is based on the use of the Doc2Vec neural network [6] to analyze static information on function and variable names obtained from binary and object files. The second method for detecting similar applications is based on applying the DTW algorithm [5] to multivariate time series, which are formed from the performance characteristic values obtained from the monitoring system during application execution. Such characteristics include, for example, CPU user load, L1 cache miss rate, the amount of data transferred over the MPI network per second, etc.

These methods have already shown high accuracy in practice, and therefore it has been decided to adapt these methods to solve a related problem: determining the values of proposed assessments based on historical data for jobs with no assessments obtained using the standard way described above. The solution should be mainly based on the dynamic method, as it is more suitable for fine-grain similarity detection. But, it is possible to combine both methods: first, we perform a primary filtering of jobs which can potentially be similar using a static method, and then apply a dynamic method to the remaining jobs. As the first results showed, this not only makes it possible to speed up the analysis process (since the dynamic method is much slower than the static one), but also to increase the accuracy. Thus, we propose the following general working algorithm:

- We collect a knowledge base, which accumulates historical information about assessment values for real-life Lomonosov-2 jobs (for which the standard method described above has worked).
- When a new job is found, and standard method failed to assess it, we do the following:
  - We find all jobs in the knowledge base that are similar to the new one using static analysis.
  - If we find statically similar jobs, we search for similar jobs among them using dynamic method with coarse threshold (step A).
  - If we do not find statically similar jobs, we select similar jobs in the whole knowledge base by dynamic method using a lot finer threshold (step B).

The idea is that if we find statically similar jobs the probability that dynamic analysis will falsely find similar job is low, so we can use coarse threshold to get more jobs of comparison.

After the analysis, we have a set of jobs that are similar to the target. For each similar job assessment values are known (they are stored in the knowledge base), so the question arise: how

should we predict the target's assessment value based on them? We considered three methods: median, average and softmax weighted average. Softmax weighted average is computed as:

$$softmax\ weighted\ average = \frac{\sum_i e^{-distance_i} * score_i}{\sum_i e^{-distance_i}}.$$

There is also a question of how to evaluate the accuracy of our prediction. Because the task of predicting assessment values is the regression task, we can use the regression evaluation metrics. The most common are Mean Absolute Error (MAE) and Mean Squared Error (MSE). In terms of the values, both MSE and MAE are positive numbers, and lesser the value — the better. MAE shows average error of the regression model, and higher the value — higher the error on average. This metric is very important in showing whether the predictions are close or not, but it has a big drawback — if there is a small percentage of predictions with very high error then the MAE will not be affected that much, because huge amount of accurate predictions will average the score out. That is why analyzing MSE is also important: it squares the error, and only then average is computed. This ensures that predictions with high error have greater effect on the score, thus, if the MSE is higher than the percentage of predictions with higher error will be higher too.

To test whether our proposed algorithm will work, we collected data on ∼1500 jobs with execution time over 30 minutes. In this case, we can consider only quite long jobs because if execution time is small there is very little useful information about job behavior. Each job has assessments of the $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ that we can use for further comparison with the predicted value.

During testing, we changed 2 parameters of the specified algorithm: aggregation function (median, average or softmax weighted average) and selected approach. We decided to test three different approaches:

- Combined approach #1 — the default approach proposed earlier, using both steps A and B in the working algorithm.
- Combined approach #2 — the default approach, but with step A only, no step B, since step B allows analyzing more jobs but tend to decrease the accuracy.
- Dynamic analysis only — using dynamic analysis only, with no static analysis involved.

Tables 2 and 3 show the evaluation results for $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ correspondingly.

Table 2. Accuracy of predicting $\text{score}_{\text{cpu}}$

Selected configuration	**% of jobs predicted**	**MAE**	**MSE**	**% of jobs with AE >10**
Combined approach #1, softmax	56.62	1.39	7.42	0.65
Combined approach #2, softmax	33.06	1.66	8.12	0.47
Dynamic analysis only, softmax	80.53	3.71	37.97	8.01
Combined approach #1, median	56.62	1.35	7.88	0.95
Combined approach #2, median	33.06	1.68	9.09	0.71
Dynamic analysis only, median	80.53	3.89	50.96	8.90
Combined approach #1, average	56.62	1.48	8.06	0.65
Combined approach #2, average	33.06	1.77	8.79	0.47
Dynamic analysis only, average	80.53	4.12	43.81	9.61

Table 3. Accuracy of predicting $score_{mem}$

Selected configuration	**% of jobs predicted**	**MAE**	**MSE**	**% of jobs with AE >10**
Combined approach #1, softmax	56.60	1.39	7.42	0.65
Combined approach #2, softmax	33.19	1.64	13.02	0.54
Dynamic analysis only, softmax	80.81	3.11	34.97	5.54
Combined approach #1, median	56.85	1.32	9.98	0.77
Combined approach #2, median	33.19	1.67	14.14	0.72
Dynamic analysis only, median	80.81	3.23	40.33	6.67
Combined approach #1, average	56.85	1.41	9.91	0.60
Combined approach #2, average	33.19	1.79	13.77	0.54
Dynamic analysis only, average	80.81	3.42	37.89	6.56

According to the provided results, dynamic only analysis is a clear winner in terms of the percentage of jobs predicted, as it covers >80% of jobs. But the quality of such prediction is quite low: more than 8% of jobs have the absolute error (AE) greater than 10, which is very high for a score ranging from 0 to 100. Combined approach #2 has the least percent of detected jobs with MAE and MSE slightly higher than if we use combined approach #1, but the percent of jobs with AE greater than 10 is the least among all configurations. Therefore, the combined approach #1 can be considered as the best option. It is in the middle ground between the other two variants in terms of percentage of detected jobs, has the best MSE and MAE, and also has a lot less percentage of jobs with high AE than dynamic analysis only. The benefits can also been seen in Fig. 4. Graphs show the distribution histogram, where each bar corresponds to a specific difference interval between true and predicted scores, and its height represents how many jobs show the specified difference interval. Two combined approaches are very similar in "tails" (where error is quite high), but approach #1 has a higher amount of accurate predictions (bars around zero value), almost by a factor of 2.

In terms of aggregation methods, there is no clear winner. Median shows that it has the least MAE, but highest MSE and percentage of jobs with high AE. This means that most of the time predictions are very accurate, but the cases of wrong predictions are way off the true values. Average has the least percentage of jobs with high AE, but has almost the same MSE as median. This means that there are many cases when the prediction is not so close to the ground truth, but the AE is still less than 10. Softmax weighted average is in the middle of the previous two: it has the average MAE and percentage of jobs with high AE, but a lot better MSE. This means that the predictions are very close to the median yet excluding some cases with high AE. But, overall, all aggregation variants are similar. This can clearly been seen in distribution histogram in Fig. 5, where all the graphs are practically the same. It tells us that in this case the aggregation function almost does not matter, and the defining factor whether the regression model will be accurate or not is determined by the selected approach. It is worth investigating on how to increase the percentage of found jobs in combined approach #1 while still retaining the quality of the predictions.

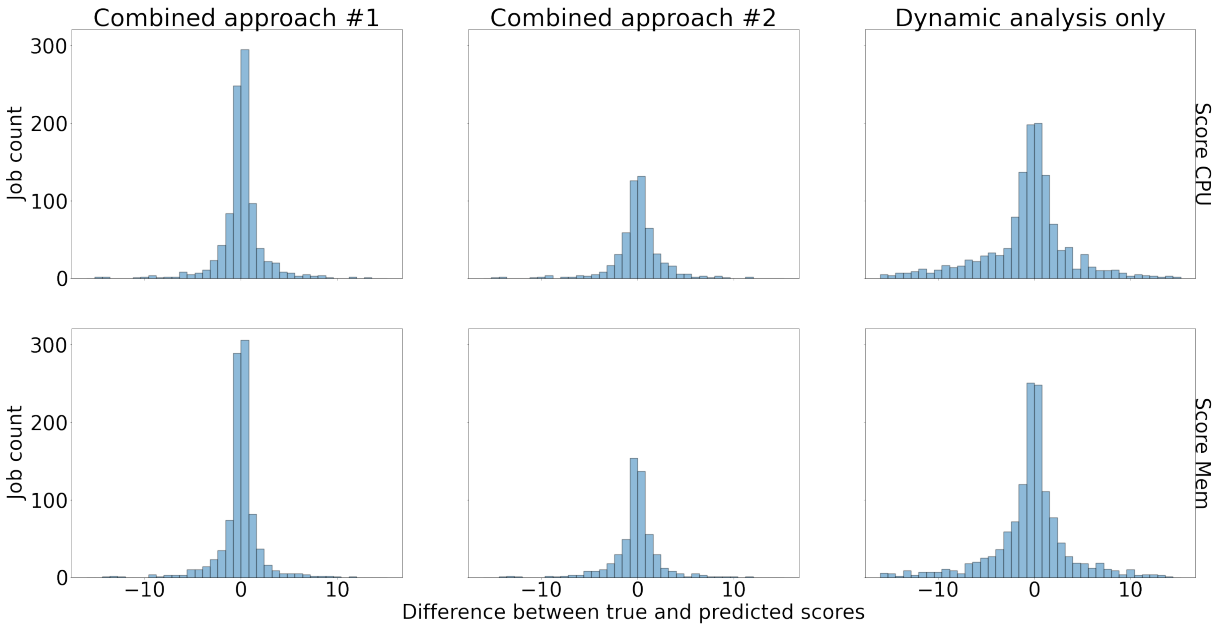

Figure 4. The distribution histogram for different approaches, softmax weighted average

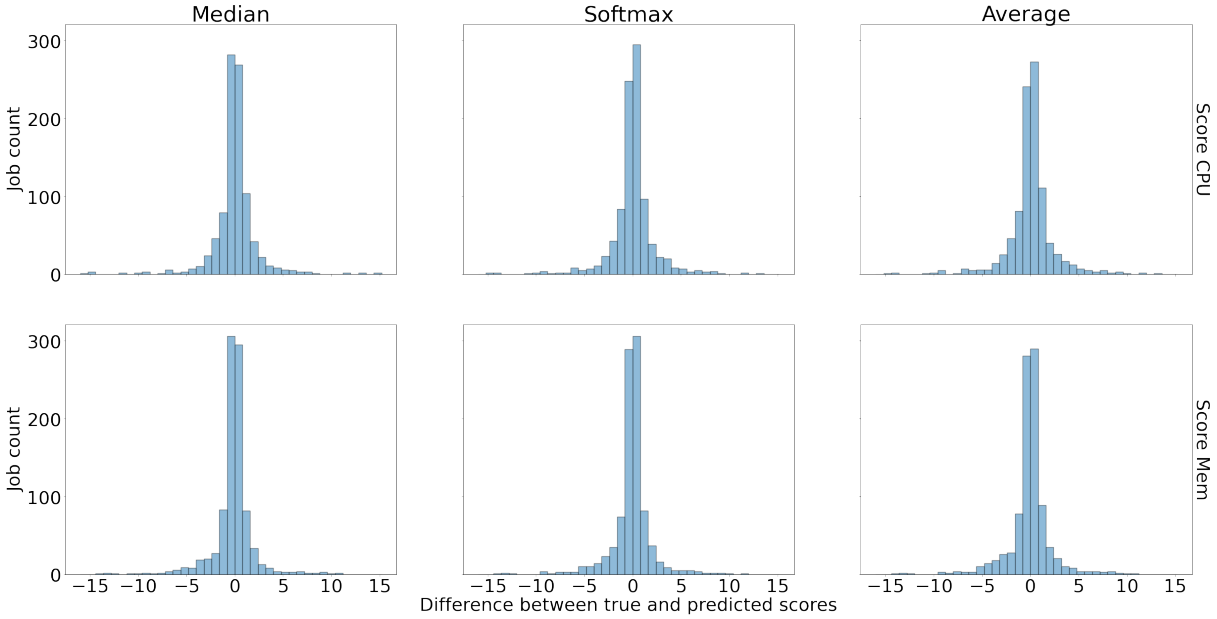

Figure 5. The distribution histogram for different aggregation methods, combined approach #1

5. Assessing the Quality of HPC Resource Provisioning

The assessment of the quality of using HPC resources would be incomplete if we do not take into account the issues of organizing the provision of these resources and the level of user interaction with the supercomputing center. In other words, it is expedient to jointly consider and assess the quality of computing resources provisioning. There are two main stages to be distinguished here:

- getting access to HPC resources;
- computing and user support.

Getting access for the first time can be a relatively complex process, mostly when getting access to the centers of collective use on a gratuitous basis, within the framework of collaborations, etc. Here, requests for confirmation of membership in a particular project, guarantees for the

use of resources for certain purposes and only for them, confirmation of the amount of resources provided, etc. are quite likely. All these questions are individual for each HPC facility, but, at the same time, they have much in common, which means that they can be formalized and automated in many ways. In particular, the Octoshell supercomputer center management system [8] provides such intuitive entities as "application", "surety", "access", which can be omitted in the system, or brought to the required form with relative ease.

At the same time, even when obtaining access on a reimbursable basis, one still needs to declare the work aims and prove allocated resources needed, even at a high level of abstraction. Indeed, for large centers, the correct use of computing power is not only a matter of profit, but also of reputation. So, in many domestic and foreign HPC centers, both for calculations and for other services, for example, application optimization, reviewing has been introduced, and it will not be possible to receive even paid services without passing it. However, given that obtaining access is a one-time task for a research project, the significance of the degree of development of solving this issue is strictly proportional to the number of projects and their activity. At the same time, an inefficient solution to this issue adds difficulties and issues to the system holder to a much greater extent than to users.

The situation is quite different with the support phase of the calculations. It is obvious that at this stage there is a number of chains of interaction between the user and the system holder. Let us consider the main ones in descending order of importance based on a survey of representatives of the largest HPC centers in Russia and some of the leading European centers.

The first group with the highest priority includes the following two areas.

1. Availability of up-to-date user documentation

It includes frequently asked questions sections, documentation on the software and hardware stack, a detailed description of typical work scenarios and use cases (registration, expertise, correct project workflow, links to resources, related publications, etc.).

The phase requires little but constant attention from the system holder, and up-to-date information is extremely important for users. Unfortunately, the general practice is that, despite being in demand, users prefer to report bugs and inability to conduct computing or research before carefully studying the documentation. However, this is precisely the task of the system owner: to keep the documentation section in such a clear and up-to-date form that users intuitively access the information they are looking for on their own, thereby reducing the unnecessary burden on administrators.

2. User support, solving user problems

Helpdesk is usually implemented as a ticket system. The fundamental importance of full-fledged user support is to minimize the time for solving emerging problems of a different nature. This is not possible without a substantial staff of administrators, effective rubrics/templates for user requests, a detailed history for each issue, the ability to reassign and engage participants to solve the problem at hand. Absolutely all surveyed HPC centers put this functionality in first place immediately after ensuring the availability of all necessary up-to-date documentation.

The next area with a moderate priority is almost unanimously recognized as improving the efficiency of user tasks.

3. Improving the efficiency of user applications

Indeed, the return of the entire HPC center consists of all the results obtained within all ongoing projects, that is, they depend on the effectiveness of each launch. There is a clear division into the commercial use of resources and gratuitous use (including paid by a third-party source).

In commercial exploitation, the user's interest in this issue is higher, despite the fact that all users are interested in minimizing the time to obtain the final result.

It would seem that it is not so important for the owner of a computing system how quickly the applied tasks will be solved if the resources spent are paid in proportion to the time and scale of the tasks. However, almost all centers are interested in demonstrating their efficiency and productivity, both for maintaining the image/reputation and to substantiate directions for further own development, as well as to attract new users and just to help users solve their tasks faster.

In a number of foreign HPC centers, special departments have been allocated to analyze user applications, mainly on a commercial basis. In addition, support staff from equipment vendors is sometimes involved in resolving such issues, while users receive assistance, and the system manufacturer receives feedback, which is extremely important when designing new systems.

There are not very many developments in this direction among local supercomputing facilities: the shortage of staff of administrators and support groups is affecting, nevertheless, we can clearly see the demand by a number of users [18]. Works at the HPC center of Moscow State University stand out, namely the TASC system, aimed at assisting in the analysis of the efficiency of user jobs. It is important that the system provides users with data and recommendations on each run and with the possibility of feedback.

The areas with the lowest priority were identified by the system holders interviewed as follows.

4. Organization of the user's workspace

This includes user's personal account, organization of notifications about significant events in workflows, a "one-stop" service. Often, such solutions are bundled with the system provided by the supplier, so only large centers with complex workflows face the need for improvements.

5. Promoting user results

This direction is image-building, largely based on bullet 3. The presentation of success stories, vivid illustrations of the tasks being solved, announcements, publications: all this, of course, is necessary and contributes to the formation of the correct image of the supercomputer center, but it is relevant only when its productive functioning is already ensured.

Given that this distribution is built on the basis of the answers of large HPC systems holders, the considered gradation can be used as a basis for assessing the quality of user service provision, for example, as a basis for a scoring system with an appropriate distribution of weight coefficients according to the priorities mentioned above.

Conclusions and Future Work

This paper describes the implementation and evaluation of the previously proposed assessment system for analyzing the quality of HPC resource usage. For each supercomputer job, assessments evaluate how "inefficiently" or "poorly" this job is using the given computational resource.

The exact formulas for calculating all assessments are proposed, and their implementation (except for GPU-related assessments) was carried out on the Lomonosov-2 supercomputer. The collected real-life statistics and examples given in this paper show that proposed solution can be useful in many ways for system administrators, providing different insights on the quality of resource usage by particular jobs, users, projects, software packages, etc.

The authors also outline that granting high quality of resource provisioning and interaction with users is a necessary step when improving the efficiency of HPC resource usage. Five most popular types of such interaction are given in the paper according to the survey of large HPC centers in Russia and Europe.

Acknowledgements

The results described in this paper, except for Section 5, were achieved at Lomonosov Moscow State University with the financial support of the Russian Science Foundation (agreement No. 21-71-30003). The results described in Section 5 were achieved at Lomonosov Moscow State University with the financial support of RFBR, project number 20-07-00864. The research is carried out using the equipment of shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. CUPTI :: CUDA Toolkit Documentation, https://docs.nvidia.com/cuda/cupti/index.html

2. High Performance Computing Market Size to Surpass USD 64.65, https://www.globenewswire.com/news-release/2022/04/04/2415844/0/en/High-Performance-Computing-Market-Size-to-Surpass-USD-64-65-Bn-by-2030.html

3. POP Standard Metrics for Parallel Performance Analysis | Performance Optimisation and Productivity, https://pop-coe.eu/node/69

4. Top-down Microarchitecture Analysis Method using VTune, https://software.intel.com/en-us/vtune-cookbook-top-down-microarchitecture-analysis-method

5. Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, USA, July 1994. Technical Report WS-94-03. pp. 359–370. AAAI Press (1994). https://doi.org/10.5555/3000850.3000887

6. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, June 21-26, 2014. JMLR Workshop and Conference Proceedings, vol. 32, pp. 1188–1196. JMLR.org (2014), http://proceedings.mlr.press/v32/le14.html

7. Nikitenko, D.A., Shvets, P.A., Voevodin, V.V.: Why do users need to take care of their HPC applications efficiency? Lobachevskii Journal of Mathematics 41(8), 1521–1532 (2020). https://doi.org/10.1134/s1995080220080132

8. Nikitenko, D., Voevodin, Vad.V., Zhumatiy, S.: Driving a petascale HPC center with Octoshell management system. Lobachevskii Journal of Mathematics 40(11), 1817–1830 (2019). https://doi.org/10.1134/S1995080219110192

9. Röhl, T., Eitzinger, J., Hager, G., Wellein, G.: LIKWID Monitoring Stack: A flexible framework enabling job specific performance monitoring for the masses. In: 2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017. pp. 781–784. IEEE (2017). https://doi.org/10.1109/CLUSTER.2017.115

10. Schulz, M., de Supinski, B.R.: P^nmpi tools: a whole lot greater than the sum of their parts. In: Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing, SC 2007, Reno, Nevada, USA, November 10-16, 2007. ACM Press (2007). https://doi.org/10.1145/1362622.1362663

11. Shaikhislamov, D., Voevodin, Vad.: Solving the problem of detecting similar supercomputer applications using machine learning methods. In: Parallel Computational Technologies. CCIS, vol. 1263, pp. 46–57. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55326-5_4

12. Shvets, P., Voevodin, V., Zhumatiy, S.: Primary automatic analysis of the entire flow of supercomputer applications. In: CEUR Workshop Proceedings. pp. 20–32 (2018)

13. Shvets, P., Voevodin, Vad., Nikitenko, D.: Approach to workload analysis of large HPC centers. In: Parallel Computational Technologies. CCIS, vol. 1263, pp. 16–30. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55326-5_2

14. Stefanov, K., Voevodin, Vl., Zhumatiy, S., Voevodin, Vad.: Dynamically Reconfigurable Distributed Modular Monitoring System for Supercomputers (DiMMon). Procedia Computer Science 66, 625–634 (2015). https://doi.org/10.1016/j.procs.2015.11.071

15. Terpstra, D., Jagode, H., You, H., Dongarra, J.J.: Collecting performance data with PAPI-C. In: Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing, September 2009, ZIH, Dresden. pp. 157–173. Springer (2009). https://doi.org/10.1007/978-3-642-11261-4_11

16. Thompson, A.P., Aktulga, H.M., Berger, R., *et al.*: LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Comp. Phys. Comm. 271, 108171 (2022). https://doi.org/10.1016/j.cpc.2021.108171

17. Voevodin, Vad., Zhumatiy, S.: Universal assessment system for analyzing the quality of supercomputer resources usage. In: Supercomputing. RuSCDays 2021. CCIS, vol. 1510, pp. 427–442. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92864-3_33

18. Voevodin, Vad.V., Chulkevich, R.A., Kostenetskiy, P.S., *et al.*: Administration, Monitoring and Analysis of Supercomputers in Russia: a Survey of 10 HPC Centers. Supercomputing Frontiers and Innovations 8(3), 82–103 (Oct 2021). https://doi.org/10.14529/jsfi210305

19. Voevodin, Vad.V., Stefanov, K.S., Zhumatiy, S.A.: Overhead analysis for performance monitoring counters multiplexing. In: Russian Supercomputing Days, RuSCDays 2022. LNCS, Springer, Cham (2022, in print)

20. Yasin, A.: A Top-Down method for performance analysis and counters architecture. In: ISPASS 2014 - IEEE International Symposium on Performance Analysis of Systems and Software, Monterey, CA, USA, March 23-25, 2014. pp. 35–44. IEEE (2014). https://doi.org/10.1109/ISPASS.2014.6844459

21. Zhou, K., Krentel, M.W., Mellor-Crummey, J.: Tools for top-down performance analysis of GPU-accelerated applications. In: Proc. of the 34th ACM Int. Conf. on Supercomputing, Barcelona, Spain, June, 2020. pp. 1–12. ACM (2020). https://doi.org/10.1145/3392717.3392752