Primary Automatic Analysis of the Entire Flow of Supercomputer Applications*

Pavel Shvets$^{1\,[0000-0001-9431-9490]}$, Vadim Voevodin$^{1\,[0000-0003-1897-1828]}$, and Sergey Zhumatiy$^{1\,[0000-0001-5770-3071]}$

Research Computing Center of Lomonosov Moscow State University, Russia, Moscow
`shvets.pavel.srcc@gmail.com, vadim@parallel.ru, serg@parallel.ru`

Abstract. Supercomputers are widely used to solve various computationally expensive tasks that arise both in fundamental scientific and applied problems. However, the extreme complexity of supercomputers' architecture leads to low real efficiency of their usage. This is aggravated by the fast growth of massive involvement in the HPC area, which leads to a significant increase in the number of users who do not have experience in developing efficient parallel applications. There are many tools for analyzing and optimizing parallel programs, however, in our experience, they are used quite rarely. One of the main reasons for that is the complexity of their proper use — it is necessary to understand which tool should be used in current situation, how to use it correctly, how to interpret results obtained with this tool, etc. Our project as a whole is aimed at solving this problem. The main goal is to develop a software package that will help users understand what performance problems are present in their applications and how these problems can be detected and eliminated using existing performance analysis tools. The first task that arises in this project is primary mass analysis of the entire flow of supercomputer applications. Solving it will allow to promptly identify the facts of inefficient application behavior, as well as give first recommendations on how to further conduct more detailed performance analysis. In this paper, we describe the features of the software package subsystem aimed at solving this task.

Keywords: High-performance computing · Supercomputer · Mass analysis · Parallel application · Supercomputer job flow · Efficiency analysis

1 Introduction

Nowadays many scientists from different academic fields need to solve computationally expensive tasks in order to obtain new results in their research. This

---

* The results described in all sections except subsection 4.2 and section 5 (concerning development of classes) were obtained in the Lomonosov Moscow State University with the financial support of the Russian Science Foundation (agreement N 17-71-20114). The results in subsection 4.2 and section 5 describing classes of applications were obtained with support of Russian Presidential study grant (SP-1981.2016.5).

important step often requires high-performance computing technologies due to its extreme computational complexity, so the number of these tasks tends to constantly increase. Therefore, more and more specialists start to use supercomputer systems for conducting their experiments, leading to a constant growth of the HPC area [1].

But it is not enough just to use supercomputer systems for solving computationally expensive tasks. It is necessary to learn how to efficiently utilize provided supercomputing resources. Otherwise, the speed of conducting experiments can drastically decrease. However, the practice shows that the efficiency of using modern supercomputer systems is surprisingly low [2]. There are many reasons for that, and two of the most important ones are the tremendous complexity of modern supercomputers' architecture and the lack of users' experience in developing efficient parallel programs. Since the architecture is getting even more complex and the number of HPC users grows over time, the problem of low efficiency of supercomputer usage is becoming increasingly important.

The efficiency of parallel applications has been quite extensively studied for a long time, which led to the development of a big number of software tools aimed at analyzing and optimizing the performance of parallel programs. Profilers, tracing tools, emulators, debuggers — all these types of performance analysis tools can be used to detect and eliminate different kinds of bottlenecks in parallel programs leading to the efficiency increase.

Unfortunately, our experience of supporting large supercomputers like Lomonosov-2 and Lomonosov-1 systems (currently #1 and #3 in Russia [3]) shows that such tools are very rarely used. We can assume two main reasons for this. One of the reasons — high complexity of the proper use of such tools. The user needs to choose a performance analysis tool suitable for his particular situation, use it in a proper way and correctly interpret the obtained results. All these steps can be quite difficult, especially for an inexperienced HPC user. And the second reason is that users often do not even know that there are some performance issues with their applications.

So, a solution is needed that will solve both of these problems. Our project is aimed at developing a software package that will help users detect potential performance issues in their applications and guide them through the proper usage of existing performance analysis tools in order to detect and eliminate the root causes of found performance issues.

The rest of the paper is organized as follows. Section 2 is devoted to the brief description of the overall system being developed in this project. Section 3 describes the background as well as the related works developed by other scientists. Section 4 contains the description of the proposed method for primary mass analysis: subsection 4.1 describes the input performance data used in this method, and subsection 4.2 explains what primary analysis results are provided by the developed solution. The examples of using this method on real-life applications of Lomonosov-2 supercomputer are shown in section 5. Section 6 includes conclusion and our plans for future work.

2 Overview of the Overall System

The main goal of the overall system that is being developed in this project is to help supercomputer users to detect and eliminate performance issues in their applications. In our opinion, in order to achieve this goal, the system must be built according to the following main principles and requirements:

1. **Mass user orientation**. This system is aimed at users of any proficiency level. For basic system usage, additional knowledge and training should not be required; however, a user with a higher level of training or experience should be able to obtain more detailed information with the level of detail he needs.
2. **Usage of existing performance analysis tools**. A lot of successful work has already been done by other researchers in studying particular aspects of job performance, and we plan to apply this knowledge in our project, using existing performance analysis tools whenever possible. Some examples of such tools are shown in Section 3.
3. **Active interaction with end user**. Potential performance issues detected by our software package may not always be absolutely correct or complete, so we plan to develop a feedback mechanism which will enable users to explicitly confirm (or adjust) the job analysis results. This will help us to increase the accuracy of our solution.
4. **Accumulation of gained knowledge**. It is supposed to accumulate a rather extensive knowledge base about real-life efficiency problems in different types of applications, as well as their root causes and ways to eliminate these causes. The presence of such a database will allow in the future, when analyzing a new program, to find similar, already studied situations and thus use the experience gained earlier.

Our proposed solution that is being developed consists of several important parts. The first part is devoted to the primary mass analysis of the overall supercomputer job flow, which enables us to detect basic performance issues in the parallel applications running on this supercomputer. This part is done automatically, without any user involved, since at this initial step users are not aware of any issues in their programs. In this part we need to analyze every running program and notify its owner (and/or system administrators, if the situation is critical) in case we have found any bottlenecks that can potentially decrease the program efficiency.

The next part is responsible for the detailed analysis of a particular program. If a user wants to find out more information about performance of his program, especially if any potential performance issues were found during the first part, he will be able to start a particular type of detailed analysis (e.g. tracing of MPI programs, analysis of memory efficiency or low-level analysis) for studying a particular aspect of program behavior. In this case our software package should guide the user through this process, helping in an automated way to choose an appropriate existing analysis tool, use it properly and interpret the results achieved with this tool. This part should be done in close cooperation with the user, because it allows to conduct an efficiency analysis that combines user expertise and knowledge about his program as well as our experience in using performance analysis tools. If obtained information is not enough, it should be possible to start another type of detailed analysis.

The last part of our solution is aimed at analyzing the efficiency of the supercomputer as a whole. It is devoted to help system administrators detect and solve problems similar to the ones studied in first two parts, but scaled up to the level of the whole system. The main goal of this part is to analyze the efficiency of the overall job flow, concerning the current configuration of the supercomputer.

In this paper, we focus mainly on the first part of the project — primary mass analysis of the supercomputer job flow. The other parts are not yet implemented, although they are under active development.

3 Background and Related Work

The fundamental problem to be solved by this project is the low efficiency of supercomputing system usage as a whole and applications executed on these systems in particular. As it was stated earlier, there is a lot of existing performance analysis tools, but almost all of them are focused on studying only one particular aspect of application behavior like the optimality of cache usage, MPI efficiency, I/O analysis, etc.

At the same time, there are only a small number of studies designed to help a user to understand this diversity of software tools and to conduct an independent comprehensive analysis of possible problems in his applications. One of such studies was the POP (Performance Optimization and Productivity) CoE (Center of Excellence) Project [4, 5]. Its objectives are the closest to our project. A competence center was created to help users from European organizations to improve the efficiency of their applications. This center performs an audit of user applications and offers methods for refactoring code to improve performance. The major difference between our project and the described one is that in the POP CoE Project all the work on study and optimization of applications was performed manually by the Center's experts. In our proposed project, it is planned to develop a software tool suite that will help make this process more automated (although less insightful) and available to the masses.

Furthermore, several software tools that allow to analyze different aspects of application performance can be pointed out. For example, Valgrind package [6] helps to perform a comprehensive study of different memory usage issues. It has a core that performs instrumentation of binary files and records all of the memory operations executed. This core functionality is then used by a number of tools, each aimed at a particular type of memory usage analysis (heap profiler, detection of memory-based errors, cache performance, etc).

The next package — Score-P [7] — is aimed at providing a convenient unified access to several popular analysis tools like Scalasca [8] or Vampir [9]. This approach makes it easier for the user to analyze MPI or OpenMP programs. Similar functionality is provided by the TAU Performance System toolkit [10].

Among commercial solutions, the Intel Parallel Studio XE package [11] can be mentioned. The tools from this package cover many aspects of program behavior, however, this package stands out more on an abstract level and almost does not assist a user in selecting tools suitable in his particular situation or in analyzing the obtained results for the application being studied.

Since this paper is mainly focused on the primary mass analysis of the supercomputer job flow, related works in this area should also be mentioned. None of the existing solutions described earlier can help a user to detect applications with performance issues; they can be applied only if it is already known that this particular application needs to be analyzed. To solve this problem, primary analysis of each job in the supercomputer job flow should be performed. In practice, monitoring systems like Nagios [12] or Zabbix [13] are usually used for this purpose in order to detect inefficient behavior of running applications. But such systems do not perform any complex analysis of the collected performance data; the main task of detecting suspicious programs lies on system administrators.

There are a number of studies aimed at performing related tasks more automatically. For example, several studies are aimed at detecting abnormally inefficient behavior in computing systems using machine learning methods [14, 15]. A good review of various systems for detecting bottlenecks and performance issues in computing systems is provided in [16]. Another approach is based on detecting abnormal behavior of a compute node, instead of analyzing application performance data. Such approach is being studied in [17, 18]. Other works like [19, 20] are aimed at identifying patterns (or types) of applications, which helps to better understand the structure and behavior of the job flow in general.

Our group in RCC MSU also conducts research of the efficiency of the supercomputer job flow. In [21] we propose a machine learning-based method for automatically detecting abnormal jobs using system monitoring data. Another approach called JobDigest [22] provides reports on dynamic analysis of performance data for each application running on the supercomputer.

Many of these studies can be used for detecting particular types of possible performance issues. But the goal of our project is to develop a global approach that will help to integrate all this knowledge and use it for detecting root causes of performance issues as well as providing recommendations on further detailed analysis.

4 Developing Method for Primary Mass Analysis

This section is devoted to the description of the first part of the overall project — primary mass analysis. Within this part, we want to detect potential performance issues for every job in the supercomputer job flow, find out potential root causes of these issues if possible, and make a set of recommendations for the user on further analysis and elimination of these root causes. Such type of analysis is primary and therefore should not be detailed, especially knowing the fact that at this stage we do not have any information about the internal structure of the analyzed program. Therefore, in almost every case it will be impossible to detect and eliminate all performance issues and root causes. So the major aim of primary analysis is to set the direction of further analysis that should be performed within the next part of the overall project devoted to the detailed analysis of each particular application.

In order to implement this primary part, we need to collect the input data on supercomputer job flow as well as develop methods for analyzing this data. Further in this section, a description of these two major steps will be given. It should be mentioned that these steps are platform-dependent, therefore currently they can be directly applied only to the Lomonosov-2 supercomputer. But the overall idea and the proposed method can be used in other supercomputer centers as well.

4.1 Input Data Used in the Proposed Method

At the first step, we need to collect performance data on every job running on the supercomputer. This data is usually collected by monitoring systems, since they provide the most accurate and complete information about job behavior. In our case, we use a proprietary system on the Lomonosov-2 system, but we are planning to move to a more flexible and dynamically reconfigurable DiMMon system [23] being developed in RCC MSU.

We collect all kind of job performance characteristics (provided by monitoring systems) like CPU load, load average, memory usage intensity (number of cache misses per second, number of load and store operations per second), network usage intensity (number of bytes or packets sent/received per second). Since at this stage the main focus is on the mass analysis of the overall job flow, and not on the fine study of a particular application, we do not need detailed information on this application behavior during its execution. Therefore, we currently collect only integral values (minimum, maximum, average) of performance characteristics.

In future, we plan to add more complex data for primary analysis. For example, we want to use more "intellectual" data on job performance collected by other analysis tools like anomaly detection system being developed in RCC MSU [21]. Adding such data will help to provide more accurate primary analysis and detect more types of possible performance issues. Another source of input data that can be helpful at this part is the information about the state of supercomputer components during the job execution. For example, a network switch failure can seriously slow data transfer for a number of running jobs, and high overall load of the file system can decrease the intensity of job I/O. Therefore, the information about such issues with software or hardware infrastructure should also be analyzed in order to detect possible root causes of job inefficiencies.

It can be noted that the same input performance data will be used in other parts of the overall project (e.g., for fine analysis of a particular program), but in that case the data must be much more detailed, since integral values are definitely not enough for deep analysis of root causes of performance issues.

4.2 Results of Primary Analysis

After collecting performance data on every job, it is necessary to perform the analysis step in order to obtain results that can be helpful for detecting and dealing with potential performance issues. Primary analysis part provides two types of results. The main results are obtained by rule triggering, and additional information is provided by classes.

The first type of performance information that can be specified for each job is based on classes. We have developed a number of classes that describe different types of job behavior. For each job, it is decided whether it belongs to a particular class or not. There are classes for describing 1-node and serial jobs; highly communicative jobs that actively use Infiniband network; jobs with behavior suitable for the supercomputer; jobs with low or high memory locality; etc. Currently we have implemented 20 classes. Each job can simultaneously belong to any number of classes.

Such kind of information allows to assess the behavior of the program in general. This data can be useful to the user since it contains quite specific information about the job properties. For example, if a job belongs to two classes: “memory intensive” and “low memory locality”, a user can understand that he most likely needs to optimize memory usage in his program.

Information about classes can help to better understand the general behavior of the job. But usually they alone are not enough to detect performance issues and even more so to determine their root causes. For this purpose, we have developed a set of 25 rules that are being checked for every job in the supercomputer job flow. If triggered, each rule describes a particular potential performance issue detected in the job behavior, as well as determines a set of its possible root causes and recommendations on what further detailed analysis should be performed.

Rules can be devoted to the study of various aspects of the program efficiency. For example, there is a rule aimed at detecting issues when the execution of MPI operations becomes the bottleneck. In this case it is necessary to optimize the MPI part of the program, for which profiling tools like mpiP, HPCToolkit, Scalasca or other tools can be used during the detailed analysis part. Another rule describes a situation when the job underutilizes the computational resources of a compute node due to a small number of active processes. In this situation, the job performance can be possibly increased by using more processes per node.

All rules are divided into several groups based on the semantics of performance issues being detected by these rules. Currently there are 9 groups of rules that are aimed at detecting the following types of issues:

1. **A hanged job**. This group of rules is intended to detect the following situation: the overall activity of a job is so low (concerning CPU load, GPU load, memory and network usage intensity) that we can be sure that this job is hanged or not working correctly.
2. **Significant disbalance of workload**. The group is devoted to any kind of computational disbalance in a job, either in a compute node or between different nodes.

3. **(for GPU-based jobs) Computational resources are underutilized**. Rules in this group are currently used to detect jobs that are heavily using GPU, while CPU host is being almost idle.
4. **Incorrect behavior / operation mode**. This group helps to detect jobs with strange behavior that normally should not appear (from a technical point of view) — low job priority, high CPU iowait, etc. Such jobs appear mostly due to the issues with supercomputer hardware or system software, but in some cases they can be caused unintentionally by the end user.
5. **Inefficient MPI usage**. The group is quite extensive and includes different kinds of rules. For example, one rule from this group evaluates network locality (average distance between compute nodes used in a job); another rule detects jobs with high intensity of MPI usage while showing low utilization of compute nodes; other rule estimates the amount of system resources spent on MPI part.
6. **Inefficient size of MPI packets**. Rules in this group help to detect a particular problem associated with inappropriate size of MPI packages that could lead to high data transfer overhead.
7. **Serial-like jobs that are not well suitable for the supercomputer**. This group detects jobs that can be quite efficient overall, but they are not well suitable for a supercomputer. It concerns, for example, serial jobs for which it is probably more efficient to use other types of computer systems like cloud systems. Rules in this group can also be useful for system administrators and supercomputer management, because such information can help them better understand and control the structure of the supercomputer job flow.
8. **Suspiciously low intensity of using computational resources**. Particular type of underutilization is also studied by rules from this group, which try to detect jobs that show suspiciously low CPU/GPU usage as well as being not communicative (almost do not use communication network).
9. **Wrong partition was chosen**. The group is used to catch jobs that were launched in inappropriate supercomputer partition. This concerns, for example, jobs that are running in a GPU partition but do not utilize GPU devices at all.

All obtained primary results (information provided by classes and rules) is made available to the supercomputer users. In our MSU supercomputer center, we use the Octoshell system [24] to communicate with users. Octoshell is a work management system that helps to manage all tasks concerning accounting, helpdesk, research project management, etc. Using the Octoshell website, any user of MSU supercomputer center can study analysis results obtained for his jobs.

5 Evaluation of the Proposed Method

The pilot version of the proposed method for primary analysis was implemented and is being evaluated on the Lomonosov-2 supercomputer. In this section we will describe some technical details of the working algorithm for performing primary analysis, as well as show examples of using the implemented method on real-life jobs.

The subsystem for performing primary analysis runs on a separate server connected to the Lomonosov-2 supercomputer. This subsystem can be logically divided into two parts: data collection module and analysis module. The data collection module constantly awaits for a signal from a resource manager about the end of a job execution via the HTTP POST protocol (technically there is no big difficulty to analyze running jobs as well, this will be implemented in the near future). After getting a signal, this module collects performance data from the external source provided by the monitoring system, processes this data and stores it in an internal PostgreSQL database.

This data is then used by the analysis module, which checks what classes or rules are triggered for each job. The obtained primary results are then sent using HTTP POST protocol to the Octoshell front-end. Flask web-server [25] is used for supporting interactions between this subsystem and external modules (external data sources, Octoshell front-end, etc).

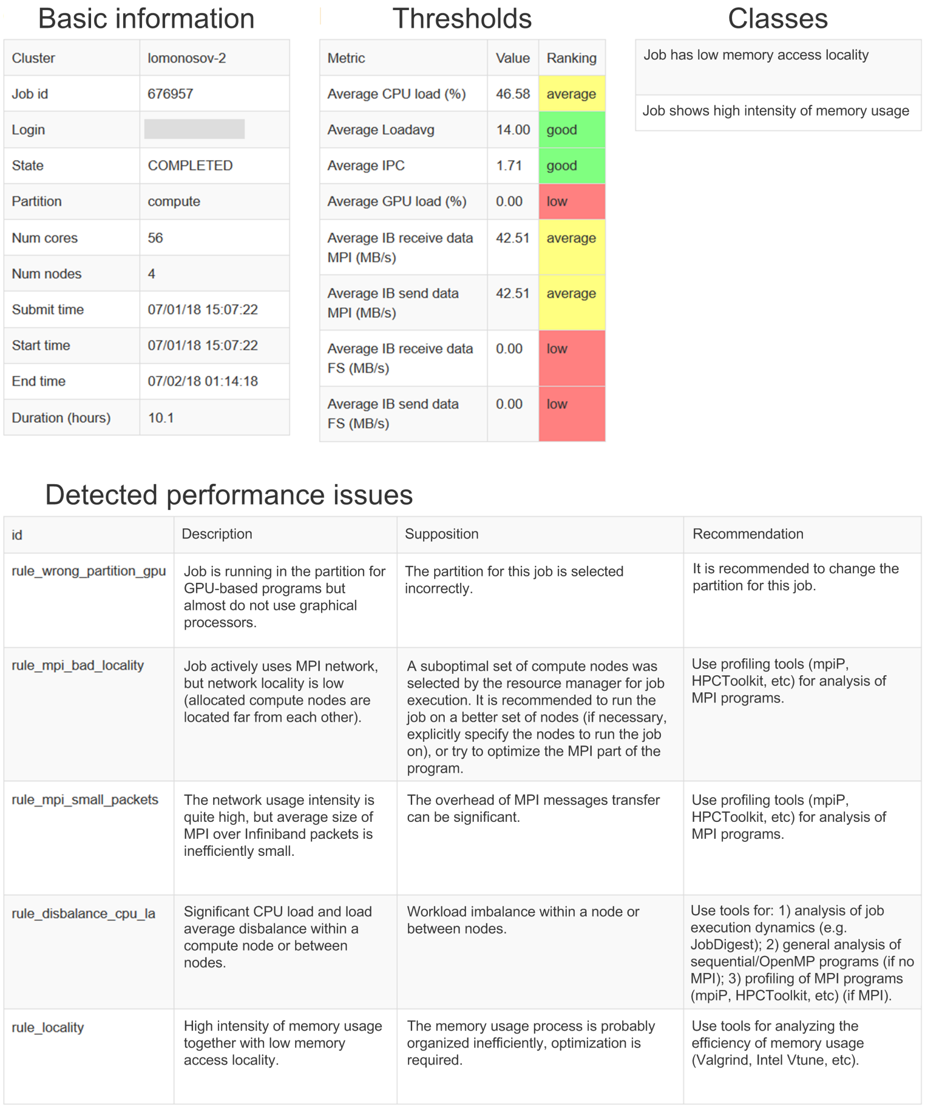
It is worth mentioning the overheads brought by this subsystem. Since the described implementation is running on a separate server, there is no application performance degradation introduced by the primary analysis part. The only source of possible overheads is the monitoring system collecting performance data on the running jobs. Our experiments has shown that this data collection process requires less than 1% of CPU load on compute nodes and less than 1% of communication network bandwidth under normal conditions, so overall overheads are very low.

The example of the web page with primary analysis results for a particular real-life job is shown in Fig. 1. Such page is available for every job executed on the Lomonosov-2 supercomputer, so any user can analyze any his job he is interested in.

The structure of this page is as follows. The upper left "Basic information" block just shows the description of this job — job id, size and duration of the job, status of its completion and name of used partition.

The most basic information on the job performance is provided in the upper central "Thresholds" block. For this purpose, the average values of main performance characteristics are studied. For each characteristic, the average value is compared to manually selected thresholds and therefore marked as "low", "average", "high". Based on the thresholds in Fig. 1, we can understand that generally this job seems to be quite efficient (normal CPU load, high load average and instruction per second (IPC) values), it does not use GPU at all as well as does not put any load on a file system, but it does use Infiniband network for MPI communication. Although such information is very general and does not help to detect performance issues, it is still useful for understanding the overall job behavior.

Further, in the upper right block the triggered classes are shown. We can see that this job belongs to two classes which indicate that this job works ac-



Basic information

Cluster	lomonosov-2
Job id	676957
Login	[redacted]
State	COMPLETED
Partition	compute
Num cores	56
Num nodes	4
Submit time	07/01/18 15:07:22
Start time	07/01/18 15:07:22
End time	07/02/18 01:14:18
Duration (hours)	10.1

Thresholds

Metric	Value	Ranking
Average CPU load (%)	46.58	average
Average Loadavg	14.00	good
Average IPC	1.71	good
Average GPU load (%)	0.00	low
Average IB receive data MPI (MB/s)	42.51	average
Average IB send data MPI (MB/s)	42.51	average
Average IB receive data FS (MB/s)	0.00	low
Average IB send data FS (MB/s)	0.00	low

Classes

Job has low memory access locality
Job shows high intensity of memory usage

Detected performance issues

id	Description	Supposition	Recommendation
rule_wrong_partition_gpu	Job is running in the partition for GPU-based programs but almost do not use graphical processors.	The partition for this job is selected incorrectly.	It is recommended to change the partition for this job.
rule_mpi_bad_locality	Job actively uses MPI network, but network locality is low (allocated compute nodes are located far from each other).	A suboptimal set of compute nodes was selected by the resource manager for job execution. It is recommended to run the job on a better set of nodes (if necessary, explicitly specify the nodes to run the job on), or try to optimize the MPI part of the program.	Use profiling tools (mpiP, HPCToolkit, etc) for analysis of MPI programs.
rule_mpi_small_packets	The network usage intensity is quite high, but average size of MPI over Infiniband packets is inefficiently small.	The overhead of MPI messages transfer can be significant.	Use profiling tools (mpiP, HPCToolkit, etc) for analysis of MPI programs.
rule_disbalance_cpu_la	Significant CPU load and load average disbalance within a compute node or between nodes.	Workload imbalance within a node or between nodes.	Use tools for: 1) analysis of job execution dynamics (e.g. JobDigest); 2) general analysis of sequential/OpenMP programs (if no MPI); 3) profiling of MPI programs (mpiP, HPCToolkit, etc) (if MPI).
rule_locality	High intensity of memory usage together with low memory access locality.	The memory usage process is probably organized inefficiently, optimization is required.	Use tools for analyzing the efficiency of memory usage (Valgrind, Intel Vtune, etc).

Fig. 1. Web page with primary analysis results for a selected job

tively with the memory subsystem, but the memory access locality is quite low, meaning that this job does not work efficiently with the memory. This kind of information can already help the user to understand possible bottlenecks of his job.

The main part of this page is the “Detected performance issues” block in the bottom, where the information about triggered rules is shown. We can see that 5 rules apply to this job. The first rule shows that the wrong supercomputer partition was chosen for executing this job, since this partition is for GPU-based jobs but this job does not use GPU at all.

The second and third rules are aimed at detecting MPI-based performance issues. Rule #2 says that this job uses MPI network quite intensively, but the network locality (average logical distance between allocated compute nodes) is low, meaning that the nodes are located far from each other, leading to possibly significant MPI-based overheads. Rule #3 detects another possible MPI-based performance issue of inefficient size of MPI packets. In this case the average size of MPI packets is very small, which often leads to MPI-based overheads as well, due to the network latency and big ratio of service data in MPI packets.

Next rule studies the utilization of compute nodes based on CPU load and load average values. According to this rule, this job shows significant workload disbalance, either within a compute node or between different nodes. This usually leads to suboptimal job performance (processes with less work need to be idle waiting for processes with more work) and can usually be optimized by balancing the workload.

The last rule summarizes the information about memory usage provided by two classes described earlier and gives more detailed description of what is the supposition in this case and how this can be further analyzed.

Such kind of information can be very helpful to the user since it allows to understand the major issues with the job and sets the directions for further performance analysis.

6 Conclusions and Future Work

In this paper we describe our approach for performing primary mass performance analysis of every job in the supercomputer job flow. This approach is intended to help to automatically detect jobs with potential performance issues and notify users about them, which is the first step necessary for finding and eliminating the root causes of job performance degradation. The proposed approach is based on analyzing the performance data provided by the monitoring system running on a supercomputer. Using this performance data, a set of 25 rules is developed. Each rule, being triggered for a particular job, describes a detected potential performance issue as well as suppositions on its root causes and recommendations for further analysis. Any user can study primary analysis results obtained for any of his jobs. The pilot version of the approach for performing primary analysis is implemented and being evaluated on the Lomonosov-2 supercomputer, showing the correctness and applicability of the proposed idea.

In the near future, we plan to add more input data for the primary analysis like "intellectual" information on job performance or data on the state of supercomputer components during the job execution; this will enable more precise and holistic primary analysis.

Further, we are going to start developing methods for performing detailed analysis of a particular job (the second part of the overall software solution) using existing performance analysis tools. This detailed analysis will be partly based on the results obtained during primary mass analysis described in this paper.

References

1. Joseph, E., Conway, S.: Major Trends in the Worldwide HPC Market. Tech. rep. (2017)
2. Voevodin, V., Voevodin, V.: Efficiency of Exascale Supercomputer Centers and Supercomputing Education. In: High Performance Computer Applications: Proceedings of the 6th International Supercomputing Conference in Mexico (ISUM 2015). pp. 14–23. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32243-8_2
3. Current rating of the 50 most powerful supercomputers in CIS, http://top50.supercomputers.ru/?page=rating
4. Bridgwater, S.: Performance optimisation and productivity centre of excellence. In: 2016 International Conference on High Performance Computing & Simulation (HPCS). pp. 1033–1034. IEEE (jul 2016). https://doi.org/10.1109/HPCSim.2016.7568454, http://ieeexplore.ieee.org/document/7568454/
5. Performance Optimisation and Productivity — A Centre of Excellence in Computing Applications, https://pop-coe.eu/
6. Nethercote, N., Seward, J.: Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. SIGPLAN Not. **42**(6), 89–100 (2007). https://doi.org/10.1145/1273442.1250746, http://doi.acm.org/10.1145/1273442.1250746
7. Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Others: Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Tools for High Performance Computing 2011, pp. 79–91. Springer (2012)
8. Geimer, M., Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Mohr, B.: The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience **22**(6), 702–719 (2010)
9. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The vampir performance analysis tool-set. In: Tools for High Performance Computing, pp. 139–155. Springer (2008)
10. Shende, S.S., Malony, A.D.: The TAU parallel performance system. The International Journal of High Performance Computing Applications **20**(2), 287–311 (2006)
11. Intel Parallel Studio XE, https://software.intel.com/en-us/parallel-studio-xe
12. Infrastructure monitoring system Nagios, https://www.nagios.org/
13. Documentation on Zabbix software, http://www.zabbix.com/ru/documentation/

14. Guan, Q., Fu, S.: Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In: Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on. pp. 205–214. IEEE (2013)
15. Fu, S.: Performance metric selection for autonomic anomaly detection on cloud computing systems. In: Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE. pp. 1–5. IEEE (2011)
16. Ibidunmoye, O., Hernández-Rodriguez, F., Elmroth, E.: Performance anomaly detection and bottleneck identification. ACM Computing Surveys (CSUR) **48**(1), 4 (2015)
17. Tuncer, O., Ates, E., Zhang, Y., Turk, A., Brandt, J., Leung, V.J., Egele, M., Coskun, A.K.: Diagnosing Performance Variations in HPC Applications Using Machine Learning, pp. 355–373. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-58667-0_19
18. Lan, Z., Zheng, Z., Li, Y.: Toward automated anomaly identification in large-scale systems. IEEE Transactions on Parallel and Distributed Systems **21**(2), 174–187 (2010)
19. Gallo, S.M., White, J.P., DeLeon, R.L., Furlani, T.R., Ngo, H., Patra, A.K., Jones, M.D., Palmer, J.T., Simakov, N., Sperhac, J.M., Innus, M., Yearke, T., Rathsam, R.: Analysis of XDMoD/SUPReMM Data Using Machine Learning Techniques. In: 2015 IEEE International Conference on Cluster Computing. pp. 642–649. IEEE (sep 2015). https://doi.org/10.1109/CLUSTER.2015.114
20. Zander, S., Nguyen, T., Armitage, G.: Automated traffic classification and application identification using machine learning. In: Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on. pp. 250–257. IEEE (2005)
21. Shaykhislamov, D., Voevodin, V.: An Approach for Detecting Abnormal Parallel Applications Based on Time Series Analysis Methods. In: Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, vol. 10777, pp. 359–369. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-78024-5_32
22. Nikitenko, D., Antonov, A., Shvets, P., Sobolev, S., Stefanov, K., Voevodin, V., Voevodin, V., Zhumatiy, S.: JobDigest – Detailed System Monitoring-Based Supercomputer Application Behavior Analysis. In: Supercomputing. Third Russian Supercomputing Days, RuSCDays 2017, Moscow, Russia, September 25–26, 2017, Revised Selected Papers. pp. 516–529. Springer, Cham (sep 2017). https://doi.org/10.1007/978-3-319-71255-0_42
23. Stefanov, K., Voevodin, V., Zhumatiy, S., Voevodin, V.: 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
24. Nikitenko, D., Voevodin, V., Zhumatiy, S.: Resolving frontier problems of mastering large-scale supercomputer complexes. In: Proceedings of the ACM International Conference on Computing Frontiers - CF '16. pp. 349–352. ACM Press, New York, New York, USA (2016). https://doi.org/10.1145/2903150.2903481
25. Flask (A Python Microframework), http://flask.pocoo.org/