On-the-Fly Calculation of Performance Metrics with Adaptive Time Resolution for HPC Compute Jobs

Konstantin Stefanov(✉)[0000−0002−0930−2713] and
Vadim Voevodin[0000−0003−1897−1828]

M.V. Lomonosov Moscow State University,
Moscow 119234, Russia
{cstef,vadim}@parallel.ru

Abstract. Performance monitoring is a method to debug performance issues in different types of applications. It uses various performance metrics obtained from the servers the application runs on, and also may use metrics which are produced by the application itself. The common approach to building performance monitoring systems is to store all the data to a database and then to retrieve the data which correspond to the specific job and perform an analysis using that portion of the data. This approach works well when the data stream is not very large. For large performance monitoring data stream this incurs much IO and imposes high requirements on storage systems which process the data.
In this paper we propose an adaptive on-the-fly approach to performance monitoring of High Performance Computing (HPC) compute jobs which significantly lowers data streams to be written to a storage. We used this approach to implement performance monitoring system for HPC cluster to monitor compute jobs. The output of our performance monitoring system is a time-series graph representing aggregated performance metrics for the job. The time resolution of the resulted graph is adaptive and depends on the duration of the analyzed job.

Keywords: Performance · Performance Monitoring · Adaptive Performance Monitoring · Supercomputer · HPC

1 Introduction

Performance monitoring in High Performance Computing (HPC) is a method to debug performance issues in different types of applications. It uses various performance metrics obtained from the compute nodes the application runs on, and also may use metrics which are produced by the application itself.

The common approach to building performance monitoring systems is to store all the data to a database. After the job is finished the data for the specific job is retrieved from the database. Then per job derived metrics like value average are calculated from the retrieved data. Those data are used to perform an analysis of the finished job. The overall picture is shown in Fig. 1. This approach works well when the data stream is not very large. For large performance monitoring data stream this incurs much IO and imposes high requirements on storage systems which process the data.

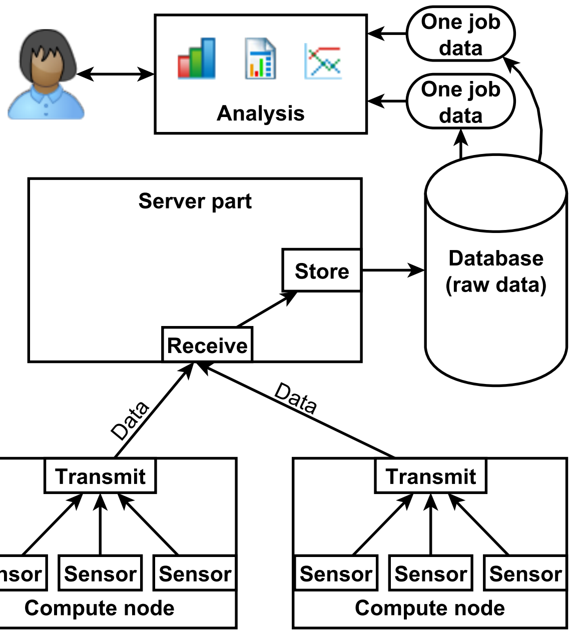

Fig. 1. Common approach to performance monitoring in HPC

Another approach may be to separate data for different applications and store data for different applications separately. This is easy to implement if the allocation of the resources to the applications is static and the rules to separate data flows from different applications can be set up before the monitoring system is started. But for HPC compute jobs the resources are assigned by the resource manager just before the job starts. Therefore we can't setup static separation of data for different jobs.

In this paper we propose an approach to performance monitoring in the case if the distribution of the resources to the applications is dynamic and may change on-the-fly. We used this approach to implement performance monitoring system for HPC cluster to monitor compute jobs. The output of our performance monitoring is a time-series graph representing aggregated performance metrics for the job which we call JobDigest [1]. The time resolution of the resulted graph is adaptive and depends on the duration of the analyzed job. The proposed approach is evaluated on Lomonosov-2, a largest Russian supercomputer.

The contribution of this paper consists of two parts. First, we propose a method for calculating performance monitoring metrics for HPC jobs on-the-fly by dynamically reconfiguring data processing paths when a job is started or finished. The performance data are analyzed without intermediate storing. This method allows reducing the amount of IO operations for processing performance monitoring data. Second, our approach automatically sets the time resolution of the result data from as small as 1 second to greater values. This enables to have the result of approximately equal size for every job without significantly reducing the available information about jobs.

The paper is organized as follows. Section 2 presents common information about performance monitoring in HPC and describes related work in field of performance monitoring. Section 3 gives the details about our proposed approach to calculating performance metrics on-the-fly. Section 4 gives an overview of making time resolution of data for compute job adaptive and depending of job duration. Section 5 describes the test performance monitoring system we implemented to evaluate our approach, and section 6 contains the conclusion.

2 Background and Related Work

In HPC clusters when the compute job is ready to start, it is allocated a set of compute nodes. In most cases a node is assigned exclusively to one job. When the job is finished, the nodes assigned to it become free and will be allocated to other jobs. The main object of the performance monitoring in HPC is a compute job.

Many approaches to performance monitoring have been proposed. Some of them are intended for inspecting only selected jobs. Most often whether the job is to be evaluated is decided before the jobs is started. Such performance monitoring systems [2–7] start agents on nodes which are allocated to the inspected job when the job is starting. In this approach only the data which are needed to evaluate the performance of the selected jobs are collected. But with this approach it is not possible to determine the reasons why a job's performance was worse than it was expected if the job was not set up for performance monitoring before the job was started. Another drawback of this approach is that the data which require superuser privileges to collect are hard to get, as the job and the monitoring agent are run by an ordinary user. And still the data in the database is stored without a label which point to job ID which produced the data. When a job is to be analyzed, the data are retrieved using time period and node list as a key and performance metrics are calculated afterwards.

There also exist approaches [8, 9] which run agents on every node of the compute system and store all the data into a database. When a job performance metrics are to be inspected, the job's data are retrieved from the database using time period and node list when and where the job ran. This approach incurs great amount of unnecessary data stored to the database. Indeed, all the data for the jobs which will not be examined later would never be retrieved and are just a waste of all type of resources: CPU to process, net to transmit, IO operations to save, and disk space to store. Moreover, as a database that stores all the data for the compute system may become quite large, the time to retrieve data for the specific job may become quite long, so saving all data creates unnecessary delays when analyzing data.

These performance monitoring systems usually perform data averaging on periods from 1 to 15 minutes. This period is chosen based on the amount of data which the system can process.

3 Calculating Aggregate Performance Data for Compute Jobs On-the-Fly

Our approach is to separate data flows for different jobs by marking them with a label with job ID. When these separated flows are processed, the data for different jobs are collected separately.

Every compute node runs a monitoring agent which retrieves performance metrics for the OS and the hardware of the compute node. The agent sends the data to a central (server) part of the monitoring system which calculates the aggregated metrics. In our approach the monitoring agent is able to mark data with a label with job ID, and this label can be set when the job is started on the specific node and cleared when the job is finished.

Resource manager is the program component responsible for allocating compute nodes to the compute jobs. Resource manager prologs and epilogs are the programs (scripts) which are executed when a job is about to start or finish. We use them to signal the monitoring agents on the nodes allocated to the job that they should start or cease to label the data.

When the data labeled with a job ID is received in the server part of the monitoring system, they are demultiplexed by the job ID and sent to a collector responsible for collecting information for that specific job. When the job is finished, that object stores the collected data to the disk. Our approach is presented in Fig. 2.

The monitoring agents on compute nodes retrieve data every 1 second and send them to the server part of the monitoring system immediately after all the metrics are collected. In the server part the data are aggregated over a short interval (we use 1 second as an initial value). For every such interval and for every performance metric we collect five aggregated metrics:

– Minimum value of the metrics over the interval
– Maximum value
– Average value
– Minimum of all nodes among average value for one compute node
– Maximum among averages for one compute node

The last two metrics are calculated as follows: first, we calculate an average of values for the given performance metric for the given compute node. Then we

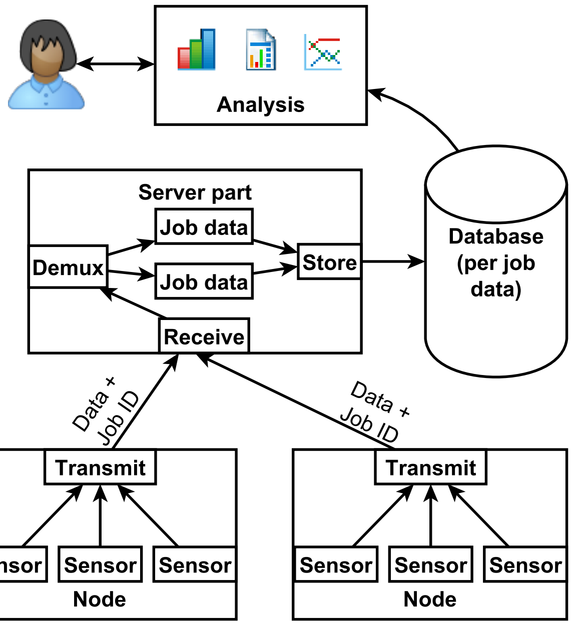

Fig. 2. The proposed approach to calculating per job performance metrics

calculate minimum and maximum among all compute nodes running the specific job. These metrics represent a measure of performance imbalance between different nodes running one job.

With this approach we do not have to store large stream of raw data, and then read it from disk thus creating large IO. Instead we collect all the data and calculate derived aggregated metrics in memory. The final result is written to disk only after the job is finished, and the size of the data written is much smaller than the size of the raw data for the job.

4 Making Time Resolution of the Result Adaptive to the Duration of the Compute Job

Another feature of our approach is a variable time period on which we do averaging of raw data. When a job is started, this time period is set to 1 second. Then we calculate predefined number (we use 1000) of samples for derived metrics. When the last sample is calculated, we merge every two adjacent samples into one and double the averaging period. So for jobs shorter than 1000 seconds we have aggregated metrics for averaging period of 1 second. For longer jobs we have between 500 and 1000 samples for aggregated metrics and each sample is done over 2^n-second period (n is such an integer that job length L satisfies $500 \cdot 2^n < L \leq 1000 \cdot 2^n$).

This allows us to have detailed graphs for relatively short jobs. As the data for every job is visualized in a browser, it is important not to have very large number of data points for every chart, otherwise it will take too long to draw the chart. Our approach gives a reasonable tradeoff between the need in detailed time resolution of the data for every job and the performance of the visualization part. And more, such an approach allows us to have upper bound on memory consumption of the server part of the monitoring system for one running job. As the data are collected in memory, this upper bound on memory consumption is important.

Such an approach can be useful for analyzing performance of short jobs. For example, a tool for anomaly detection is being developed in RCC MSU, which uses monitoring data for detecting jobs with abnormally inefficient behavior [10]. This analysis is based on machine learning techniques that require a significant amount of input data. Currently it is not possible to accurately detect abnormal behavior in short jobs (less than approx. 1 hour) due to the lack of input data from existing monitoring systems. Using the performance monitoring system with the adaptive approach will help to eliminate such problem, still allowing to analyze long jobs as well.

5 Evaluation of the Proposed Approach

To evaluate our approach we implemented performance monitoring system for producing JobDigests for all compute jobs executing on a HPC cluster. Our implementation is based on DiMMon monitoring system framework [11].

In DiMMon all processing of monitoring data (including data acquisition from OS or other data sources) is done in monitoring nodes which operate in a monitoring agent. A monitoring agent can run on a compute node or on a dedicated server.

In our implementation every compute node runs a monitoring agent. Server monitoring agent runs on a dedicated server. Compute node monitoring agent contains nodes which acquire data from OS (sensor nodes), a node that prepends a label with job ID to data, a node which transmit data to server monitoring agent for processing, and a resource manager interaction control node which receives notifications about job start and finish. Compute node agent structure is presented in Fig. 3. Server monitoring agent structure is shown in Fig. 2.

We implemented prolog and epilog scripts for SLURM [12] resource manager. The prolog script receives a list of nodes where the job is to be started. The prolog notifies agents on the compute nodes that they should start to send the performance data to server monitoring agent and that data should be marked with job ID (”resource manager interaction” node receives these notifications). It also notifies server monitoring agent that the job with given ID is about to be started. The server agent programs its demultiplexer to create separate data path for that job data. It also creates collector entity which will collect the data for that job.

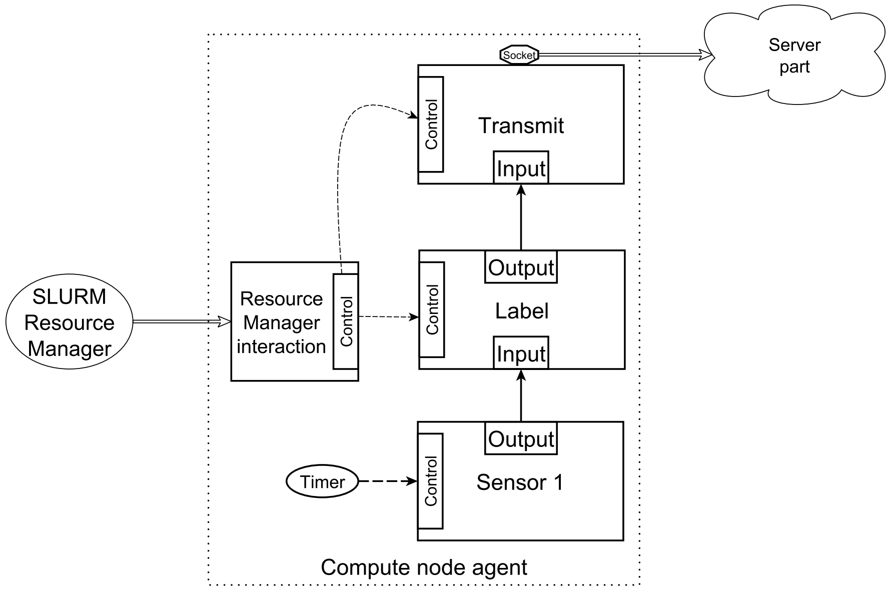

Fig. 3. Compute node monitoring agent structure

The epilog script notifies agent running on compute nodes of the finishing job to cease sending data to the server monitoring agent. It also notifies server monitoring agent that the job is finished and the aggregated data for it should be written to disk.

For evaluation we collect CPU usage level data and memory (RAM) consumption data, 21 sensors in total. CPU usage level data provide information about what fraction of CPU time is used for executing user mode code, operating system, processing interrupts etc. There are 9 such sensors provided by modern Linux kernels. Memory consumption data give the distribution of different type of memory usage: free memory volume, the amount of memory used for file cache etc. The data from the kernel is retrieved once every 1 second.

We run our performance monitoring system on Lomonosov supercomputer and Lomonosov-2 supercomputers installed in M.V.Lomonosov Moscow State University. Lomonosov is a cluster consisting of more than 6000 compute nodes. Lomonosov-2 is a cluster consisting of more than 1600 compute nodes with peak performance of 2.9 Pflops. These two machines execute about 100–200 parallel jobs simultaneously each.

We run monitoring agents on every compute node and one server monitoring agent. Compute node monitoring agents consume about 1 MB of RAM and less then 0.1% of CPU time each. Server monitoring agent run on a server with 2 4-core Intel Xeon E5450 processors with 4 300 GB SATA disks connected as RAID 10. Our monitoring agent is single-threaded, so it occupies only 1 core. It consumes about 60 MB of RAM and less than 20% of CPU (CPU consumption depends on number of jobs running on the cluster at the moment). All data for jobs executed in 1 week was about 1.6 GB in size. So our new system stores about 11 MB of data per sensor per day.

The results are saved for every job as 5 collected aggregate values for every sensor collected. It is presented to the user as a graph with 5 lines for a sensor.

For illustrating the advantages of our adaptive time resolution approach let us present the comparison of the time series graphs for the same job produced by the monitoring system described in this paper and the previous JobDigest system. The previous JobDigest system has a fixed averaging period of 120 seconds for every sensor of every job.

In Fig. 4 a part of the JobDigest for a compute job is given. The part consists of the data for two sensors: CPU user load level and Free memory sensor (it gives the amount of Free RAM as reported by Linux kernel). The averaging period (time resolution of the graph) is 1 second. The job ran for 15 minutes.

The same graphs produced by the previous JobDigest system are given in Fig. 5 for comparison. Differences in Y-axis scale is caused by the fact that the previous JobDigest system presents CPU usage level in % (the range is 0–100) and our currents system gives the same value in the range 0–1. The memory volume for the previous system is given in kilobytes, and in the current system it is given in bytes, so B (Billion) suffix instead of M (Million) for the previous system is used.

We can clearly see the difference in time resolution. On graphs produced by our systems one can identify stages and some other properties of the jobs being evaluated, while the graphs produced by the previous system contain several dots each and cannot reveal any details of the job run time behaviour.

To asses the volume of the data used by the performance monitoring system based on our proposed approach let's compare to other existing system, which is now used for creating JobDigests [1]. That system creates approx. 2.1 GB of data per day and stores min, max and average for every sensor (3 values) per node every 120 seconds. It stores data of 27 sensors, so it stores approx. 80MB per sensor per day. Our new system compared to the existing one stores more than 7 times less data while maintaining time resolution up to 120 times better (depends on job length).

6 Conclusion and Future Work

In this paper we present an approach for calculating aggregate metrics for HPC jobs on-the-fly. We label the data for the job by a label based on job ID and thus we may collect the data for separate jobs and store them in a database on a per-job basis. This greatly reduces to requirements for the database to store the collected data and allows us to save data with up to 1-second resolution on an ordinary server.

Future work is to add more sensors to our performance monitoring system including hardware processor performance counters. Another important area of

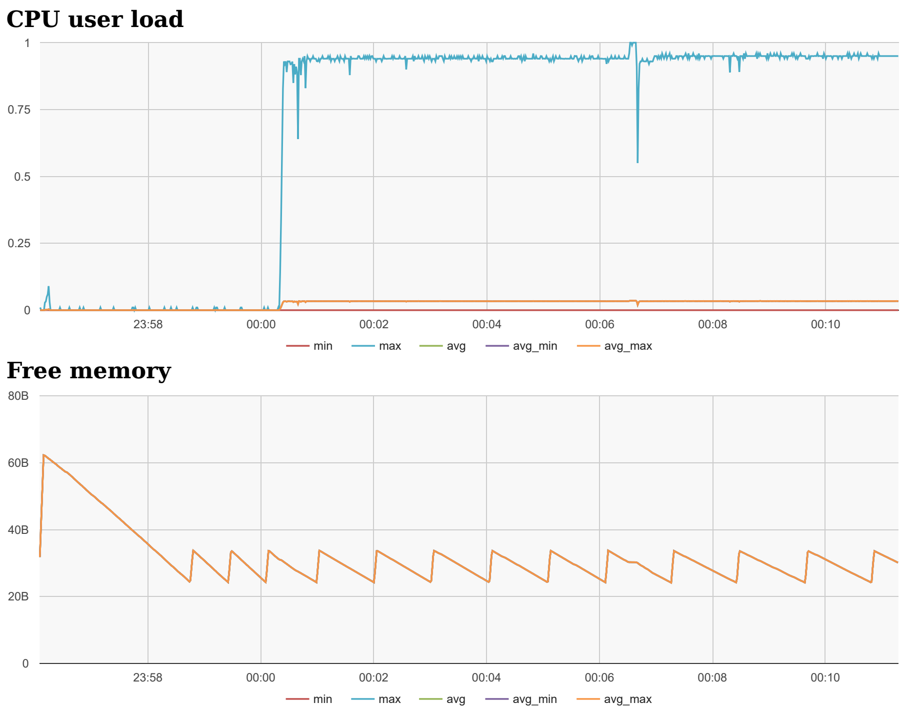

Fig. 4. An example of JobDigest produced by our system

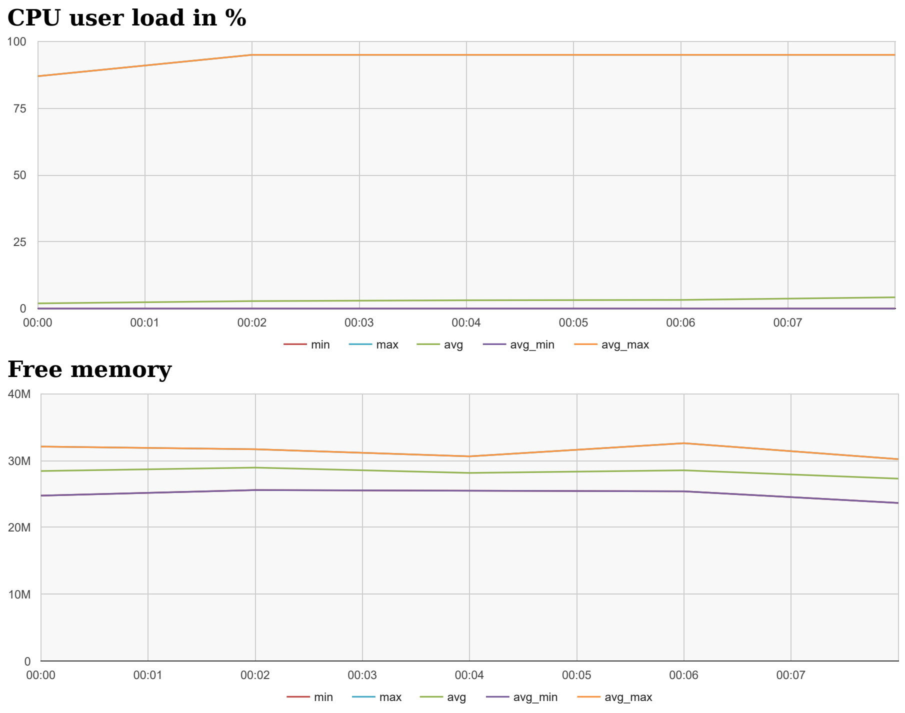

Fig. 5. A JobDigest produced by previous system

our work is to use the data produced by our new system as the input data for various analysis tools like abnormal job detecton system described in [10].

Acknowledgements

The work is supported by the Russian Found for Basic Research, grant 16-07-01121
The research is carried out using the equipment of the shared research facilities of HPC computing resources at M.V.Lomonosov Moscow State University
This material is based upon work supported by the Russian Presidential study grant (SP-1981.2016.5)

References

1. 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: Communications in Computer and Information Science. vol. 793, pp. 516–529 (2017). https://doi.org/10.1007/978-3-319-71255-0_42
2. Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., Tallent, N.R.: Hpctoolkit: tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience **22**(6), 685–701 (2010). https://doi.org/10.1002/cpe.1553
3. Eisenhauer, G., Kraemer, E., Schwan, K., Stasko, J., Vetter, J., Mallavarupu, N.: Falcon: on-line monitoring and steering of large-scale parallel programs. In: Proceedings Frontiers '95. The Fifth Symposium on the Frontiers of Massively Parallel Computation. pp. 422–429. IEEE Comput. Soc. Press, McLean, VA (1995). https://doi.org/10.1109/FMPC.1995.380483
4. Gunter, D., Tierney, B., Jackson, K., Lee, J., Stoufer, M.: Dynamic monitoring of high-performance distributed applications. In: Proceedings 11th IEEE International Symposium on High Performance Distributed Computing. pp. 163–170. IEEE Comput. Soc (2002). https://doi.org/10.1109/HPDC.2002.1029915
5. Jagode, H., Dongarra, J., Alam, S.R., Vetter, J.S., Spear, W., Malony, A.D.: A holistic approach for performance measurement and analysis for petascale applications. In: Allen, G., Nabrzyski, J., Seidel, E., Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) Computational Science ICCS 2009. Lecture Notes in Computer Science, vol. 5545, pp. 686 – 695. Springer Berlin Heidelberg, Berlin, Heidelberg (may 2009). https://doi.org/10.1007/978-3-642-01973-9
6. Mellor-Crummey, J., Fowler, R.J., Marin, G., Tallent, N.: Hpcview: A tool for top-down analysis of node performance. The Journal of Supercomputing **23**(1), 81–104 (aug 2002). https://doi.org/10.1023/A:1015789220266
7. Ries, B., Anderson, R., Auld, W., Breazeal, D., Callaghan, K., Richards, E., Smith, W.: The paragon performance monitoring environment. In: Supercomputing '93. Proceedings. pp. 850–859. IEEE (1993). https://doi.org/10.1109/SUPERC.1993.1263542
8. Kluge, M., Hackenberg, D., Nagel, W.E.: Collecting distributed performance data with dataheap: Generating and exploiting a holistic system view. Procedia Computer Science **9**, 1969–1978 (jan 2012). https://doi.org/10.1016/j.procs.2012.04.215

9. Mooney, R., Schmidt, K., Studham, R.: Nwperf: a system wide performance monitoring tool for large linux clusters. In: 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No.04EX935). pp. 379–389. IEEE (2004). https://doi.org/10.1109/CLUSTR.2004.1392637
10. Shaykhislamov, D., Voevodin, V.: An approach for detecting abnormal parallel applications based on time series analysis methods. In: Parallel Processing and Applied Mathematics - PPAM-2018. 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
11. Stefanov, K., Voevodin, V., Zhumatiy, S., Voevodin, V.: Dynamically reconfigurable distributed modular monitoring system for supercomputers (dimmon). In: Sloot, P., Boukhanovsky, A., Athanassoulis, G., Klimentov, A. (eds.) 4th International Young Scientist Conference on Computational Science. Procedia Computer Science, vol. 66, pp. 625–634. Elsevier B.V. (2015). https://doi.org/10.1016/j.procs.2015.11.071
12. Slurm workload manager, https://slurm.schedmd.com/