Automatic Detection of HPC Job Inefficiencies at TU Dresden's HPC Center with PIKA

Frank Winkler(✉) and Andreas Knüpfer

Center for Information Services and High Performance Computing (ZIH),
Technische Universität Dresden, 01062 Dresden, Germany
{frank.winkler,andreas.knuepfer}@tu-dresden.de

Abstract. The efficient use of High Performance Computing (HPC) resources is essential and requires continuous performance monitoring. The NHR HPC Center at TU Dresden has been using PIKA [5], a continuous job-level performance monitoring system, for more than five years. It is active by default and allows retrospective analysis and comparison with previous jobs. Its results are available to users for their jobs as well as to admins and HPC support for all jobs. It has proven to be very useful for reactive user support on various aspects of efficient use of HPC resources in general as well as on specific performance issues of individual users.

At the same time, the continuously collected data can be scanned proactively, as users may not yet be aware of performance issues. In this article, we report on our methods for scanning for job inefficiencies, and to inspire discussion about appropriate methods. It covers the most useful heuristic checks for a variety of aspects. We focus on meaningful performance criteria and commonly observed performance problems. All parameters and thresholds are derived from experience and tuned to detect the most severe cases in the average job mix. The heuristics range from simple cases that compare against appropriate thresholds to more sophisticated tests with some pre-processing.

Keywords: Job Monitoring · Data Analysis · Performance · Efficiency

1 Introduction

Even when an HPC system is well configured and maintained, HPC jobs can be inefficient due to a variety of factors. Common reasons are poorly optimized codes, incorrect resource allocations, or suboptimal data access. To address these issues, HPC operators need a tool that continuously monitors and analyzes the performance of jobs and classifies them into efficient and inefficient jobs. Criteria for efficient usage are shortest possible runtimes (compared to similar jobs), high utilization of the hardware and an even distribution of computational workloads across processing units.

For early detection and notification of user job inefficiencies, automated heuristics are needed to search jobs for inefficient patterns based on empirical data from the HPC center.

A. Bienz et al. (Eds.): ISC High Performance 2023 Workshops, LNCS 13999, pp. 295–306, 2023.
https://doi.org/10.1007/978-3-031-40843-4_22

PIKA is a performance monitoring stack for HPC clusters to identify potentially inefficient jobs. It consists of several open source components. These include a metric data collector to record runtime performance data on each compute node and a job metadata collector that captures all required job metadata for both exclusive and node-sharing jobs. In order to analyze a job-specific runtime metric, the job metadata must be mapped to it, in particular the node list, the list of physical cores used, and the start and end time of the job. This enables job-specific analysis for performance metrics such as CPU load, instructions per cycle, floating points operations per second or I/O bandwidths. The PIKA web-frontend contains detailed timeline visualizations of each runtime metric and provides correlation analysis between different metrics.

In order to provide HPC operators and users with additional performance feedback, PIKA performs a regular post-processing to prepare the data for further analysis. This includes, for example, the aggregation of all runtime data into so-called performance footprints. Based on the footprint and a corresponding performance model PIKA can for example determine whether a job is memory-bound or compute-bound [5]. However, other performance issues like the overall idle CPU time, load imbalances, periodic blocking I/O phases, or memory leak suspicions require more detailed analysis. For this purpose, a new analysis engine was implemented, which scans jobs for performance issues on a weekly basis. In addition, the PIKA web-frontend has been enhanced with a new feature that allows operators to sort and analyze HPC users by importance whose jobs included performance issues. Those users will be analyzed by the HPC support team and contacted accordingly in order to solve the problem together.

Section 2 describes heuristics for detecting performance issues based on job-specific timeline metrics. A case study of jobs with performance issues on the HPC system at TU Dresden is described in Sect. 3. A review of related work follows in Sect. 4.

2 Job Performance Issues

Job performance issues can be caused by the HPC user due to improper use of the job scheduling system as well as by HPC system problems such as file system overload or poor network configuration. Mostly, however, users are responsible for the efficiency of their jobs. Without knowing the application running in the job, we show that it is possible to capture inefficient jobs by their performance data. Each per-job performance metric captured by PIKA consists of one or more vectors of performance values, depending on the allocated resources and its granularity, e.g. per node, per socket, or per physical core. For example, the CPU/GPU load vectors make it very easy to detect pathological problems, e.g. when more CPUs/GPUs are requested than are used. These types of tests are easy to implement, and it is sufficient to define certain thresholds, such as when a CPU load can be considered idle. For other aspects, e.g. periodic synchronous offloading or the detection of memory leaks, one has to design exactly tailored tests and adjust parameters exactly, sometimes also ensure sufficient data quality.

Table 1. Possible performance issues with the inefficient HPC jobs of a user

Performance Issue	Description
Idle CPU/GPU Time (ICT/IGT)	Summed time intervals of all CPUs/GPUs across all jobs in which the load was close to zero
Idle CPU/GPU Ratio (ICR/IGR)	Quotient of "Idle CPU/GPU Time" and "Total CPU/GPU Time" across all jobs
Maximum Unused CPU/GPU Ratio (Max UCR/UGR)	Maximum ratio of "unused" to "used" CPUs/GPUs across all jobs
Maximum CPU/GPU Load Imbalance (Max CLI/GLI)	Maximum of the average standard deviation of CPU/GPU load across all jobs
Maximum I/O Congestion (Max IOC)	Maximum rate of metadata operations at a measuring point across all jobs. The attribution per job starts with 40 operations
Maximum I/O Blocking Phases (Max IOB)	Maximum periodic number of phases with an inverse correlation between CPU load and I/O metrics across all jobs. The attribution per job starts with 10 periodic phases
Maximum Synchronous Offloading (Max SO)	Maximum periodic number of phases with an inverse correlation between CPU and GPU load across all jobs. The attribution per job starts with 10 periodic phases
Maximum Memory Leak (Max ML)	Maximum of the linear increase of memory usage over time across all jobs

Table 1 lists all HPC job performance issue metrics on a per-user basis that we are able to generate from the job-specific vectors of performance metrics using appropriate heuristics. These have proven very useful to us in identifying users with inefficient jobs. In the following, we present the individual heuristics for each performance issue and go into detail about the threshold values and special features to be considered. It should be mentioned that we currently have chosen the thresholds in a way to provide a manageable set of "worst jobs" in the current job mix. It might be necessary to slightly adjust those thresholds in the future or for other centers.

2.1 Setup and Data Preparation

PIKA stores all metric data in the time series database InfluxDB [1]. This data is available on a long-term basis and is used for visualization and analysis. The metric data used in the following analysis is sampled every 30 s. We analyze the metrics of CPU/GPU load, memory usage, and all recorded metrics of I/O bandwidths and I/O metadata operations. The first step of the analysis is to find eligible jobs. Each of these jobs must have run for at least one hour, have more than one physical core allocated, and terminated with a "completed", "out of memory", or "timeout" Slurm status. Based on the runtime metrics of a

job, the performance issues listed in Table 1 are to be checked against the job and attributed to the job if necessary. Performance issues related to the core performance are identified for both exclusive and node-sharing jobs. All other metrics that are recorded on a per-node basis are identified for exclusive jobs.

Regarding node exclusivity and simultaneous multithreading, there are some important remarks. A job is marked as exclusive when either Slurm's exclusive flag has been set or all compute nodes associated with the job have been fully allocated. In terms of simultaneous multithreading (SMT), PIKA only stores the average over the hardware threads per physical core. Therefore, the maximum utilization of a CPU core has the value 1. In the following, we will use the term CPU to refer to a physical core.

2.2 Straightforward Heuristics

In the following, heuristics and their thresholds are presented for CPUs/GPUs that either have long idle phases, are completely unused or have large load imbalances. Idle time of CPUs/GPUs results from unused resources as well as phases in the job where the load is close to zero. These idle phases can also be caused by load imbalances, where some CPUs/GPUs may be overloaded while others remain underutilized respectively idle. For the issue metrics described below, we have defined the following heuristics and thresholds.

1. A measuring point of a CPU is idle, if the usage is below 0.01.
2. A measuring point of a GPU is idle, if the usage has the value 0.
3. A CPU/GPU is unused, if the idle count per measurement point is greater than $(n-2)$ measurement points.
4. A load imbalance is attributed to a job, if the average standard deviation of CPUs/GPUs is greater than 0.2.

Idle CPU/GPU Time is the summed idle time over all CPUs/GPUs across all jobs. Internally, we multiply the idle counts of each CPU/GPU with 30 s and sum them up. Note that the idle time is only an estimation and upper limit as we only measure every 30 s. Nevertheless, this estimation is well suited to capture long idle time phases.

Idle CPU/GPU Ratio is an issue metric that characterises the ratio of idle to total time over all CPUs/GPUs across all jobs. Internally, we compute the quotient of idle time and the sum of CPU/GPU hours per job. A value close to zero means that all CPUs/GPUs were used, while a value close to 1 means, that the user has caused almost no CPU/GPU usage for all jobs.

Maximum Unused CPU/GPU Ratio is an issue metric that characterises the maximum ratio of unused to used CPUs/GPUs across all jobs. Internally, we compute the quotient of unused and used CPUs/GPUs per job and finally provide the maximum quotient of all jobs. A value close to zero means that almost all CPUs/GPUs were used by all jobs, while a value close to 1 means, that almost no CPU/GPU was used by at least one job.

Maximum CPU/GPU Load Imbalance quantifies how the load was distributed over all CPUs/GPUs across all jobs. Internally, we calculate the standard deviation vectors across all CPUs/GPUs for each job and take the average. If the standard deviation is within the range of our defined threshold, the load imbalance issue is attributed to the job. Finally, we provide the maximum average standard deviation of all jobs. A value close to our defined threshold means that the CPUs/GPUs were reasonably evenly distributed in all jobs, while a value close to 1 means that at least one CPU/GPU was almost fully utilized and at least one CPU/GPU was almost idle in at least one job. A value equal to 0 means that no load imbalances could be detected for any of the jobs.

Another heuristic is the detection of inappropriate I/O behavior due to inefficient handling of metadata operations. Using a large number of metadata operations in a very short period of time can overload the file system, which in turn can affect the performance of the job and other jobs. The following performance issue detects jobs with high open/close rates.

Maximum I/O Congestion quantifies the peak rate of I/O metadata operations across all jobs. Internally, for each job, the open/close rate vectors summed over all nodes are queried and added up. Then, the maximum of the summed metadata vector is determined for each job and attributed to the job if the value has at least the threshold we set. Since our cluster has multiple mount points of the Lustre file system, we perform the heuristics described above for each mount point. The mount point that returns the largest value is automatically the mount point used by the job. Finally, we provide the maximum rate of metadata operations of all jobs.

2.3 Periodic Performance Issues

In scientific HPC applications, it is common for the same algorithm to be executed repeatedly for many iterations. One reason for this is that scientific computations often involve solving complex mathematical problems, such as numerical simulations of physical systems. These problems require the repeated execution of the same algorithm on different sets of input data or with different parameter values to obtain accurate results. A common bottleneck in an iteration is the use of synchronous rather than asynchronous large I/O operations at checkpoints for intermediate results, which causes the program to block and wait for the I/O operation to complete. Offloading computation to GPUs can significantly speed up the computation process, if used efficiently. To maximize the performance of those applications, CPU load should overlap GPU load whenever possible.

In the following, we present a heuristic for detecting periodic phases with an inverse correlation between two performance metric vectors. We use this heuristic to detect repetitive I/O blocking phases or synchronous offloading. For the implementation we use the NumPy [2] and SciPy [3] Python packages. Before specifically addressing the corresponding performance issues, we first describe this heuristic in general.

1. Acquire two mean metric vectors (signals) to be analyzed and check whether they are suitable for further analysis.
2. Compute the FFT of both signals using a fast Fourier transform algorithm.
3. Compute the frequency spectrum of both signals from the FFT output.
4. Normalize the amplitudes of each frequency spectrum to 1 and calculate the element-wise sum of both frequency spectra.
5. Find the maximum amplitude of the summed frequency spectrum and check if this is a dominant frequency.
6. If the conditions of a dominant frequency are met, determine the Pearson correlation coefficient between both signals.
7. If the correlation coefficient meets a defined threshold for inverse correlation, attribute the number of periodic phases (dominant frequency multiplied by job duration) to the corresponding job.

In both periodic performance issues described above, we analyze how the CPU load vector correlates with the I/O metrics or the GPU load vector. To avoid having to run the complex heuristic for every eligible job, we first validate whether the metric vectors are useful for our tests. The extensive analysis is **not** executed if the CPU/GPU vector has one of the following properties:

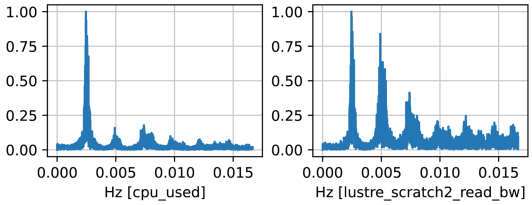
- The mean value of the CPU/GPU load vector is less than 0.1.
- The difference of the maximum and minimum value of the mean CPU/GPU load vector is less than 0.7.

With the second restriction, we want to make sure that there is enough variation of the minimum and maximum CPU/GPU load over time. To include the I/O metrics in the analysis, we have set the following conditions:

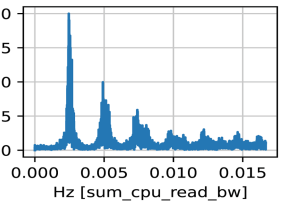
- The mean value of an I/O bandwidth vector is at least 1 MB/s.
- The mean value of an I/O metadata vector is at least 1 OPS.

To achieve more precise results for subsequent analysis, we make further adjustments to the appropriate vectors. We round all CPU load values to the first decimal place and set all I/O metric values that are less than the average to zero to avoid capturing very small FFT frequencies and to obtain a better correlation coefficient as we focus only on I/O peak values. To compute the FFT, we must ensure that all vectors are aligned, which means that there must be a value for each timestamp. If a value is missing for a certain time step, it is set with the previous value, if available, or with zero.

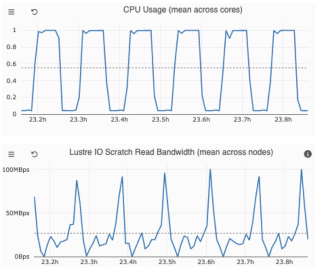
In the following, we discuss the term dominant frequency (DF), the threshold for the correlation coefficient (CC) of an inverse correlation as well as other considerations. The DF is located at the maximum amplitude of the normalized summed frequency spectrum (Fig. 1b). It is valid if the maximum amplitude is in the range between 1.8 and 2.0 and the median over all amplitudes does not exceed the value 0.1. If the value is 2.0, both metrics have exactly the same periodic behavior. If it is slightly below 2.0, a metric has one or more stronger subfrequencies unrelated to the metric being compared. If the value is below 1.8, the difference between the maximum amplitudes of both metrics is too large. The



(a) Normalized FFT plots of CPU load and I/O read bandwidth signals. For the original measured signals, the mean value was previously subtracted from each measured signal.



(b) Summed FFT plot of CPU load and read bandwidth. The dominant frequency is 0,0024 Hz with an amplitude of 2.0. The median over all amplitudes is 0.06.



(c) An excerpt from PIKA's timeline visualization showing an inverse correlation between CPU load and I/O read bandwidth. Taking into account that the values of the read bandwidth below the average line were set to zero, the calculated correlation coefficient is −0.46. The number of periods results from the dominant frequency from (b) multiplied by the job duration. Due to a zoomed section, the number of periods is higher than shown on this chart.

Fig. 1. Heuristic for detecting periodic I/O blocking phases

median of 0.1 ensures that there are only a few frequencies with high amplitudes in the entire frequency spectrum. A CC between two metrics ranges from −1 to 1, where a value of −1 indicates a perfect inverse correlation, 0 indicates no correlation, and 1 indicates a perfect correlation. After numerous tests on selected jobs with inverse correlations between CPU load vector and an I/O bandwidth vector, we settled on a CC threshold of −0.4 to identify all these jobs. To obtain an even more accurate CC value, we consider only the phase of the job between 25% and 75% of the runtime, to avoid degrading the value in the initialization and completion of the job.

Maximum I/O Blocking Phases quantifies the periodic number of phases with an inverse correlation between CPU load and I/O metrics across all jobs. Internally we perform the heuristic tests described above for each job. We check the correlation of the CPU load vector with each suitable I/O metric vector. Based on our preconditions, we automatically filter out all I/O metrics that do not belong to the used file system mount point. If we find more valid correlations between CPU load and I/O metric, we select the correlation with the better correlation coefficient. There may be a correlation of CPU Load with I/O bandwidth and I/O metadata. In general, however, read bandwidths, for example, have the same periodic behavior as read requests operations. If a valid frequency and correlation are found, the number of periodic phases is attributed to the

job, provided that the value has reached the threshold we have set. Finally, we provide the maximum number of periodic phases of all jobs.

Maximum Synchronous Offloading characterizes the periodic number of phases with an inverse correlation between CPU load and GPU load across all jobs. The entire heuristic is analogous to the I/O blocking tests, except that only one correlation test between two metric vectors is required.

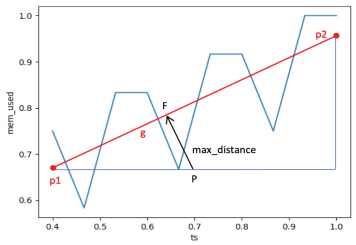
2.4 Memory Leaks

Memory leaks are a recurring and particularly detrimental problem in some HPC applications, occurring when memory is continuously allocated without being released. This is typically caused by code bugs rather than faulty resource allocation. However, it is definitely a problem for efficient HPC usage. Memory leaks of HPC jobs can be detected by observing a strong positive linear trend in memory usage, but it is not a guarantee. For example, if a job is processing increasingly large amounts of data over time, it may require more memory and exhibit a positive linear trend in memory usage. Nevertheless, this trend could be a potential indicator of a memory leak.

We have developed a heuristic (Fig. 2) that checks jobs with a memory leak suspicion based on the memory usage vector. Since a certain amount of memory is allocated during initialization, which is released at the end by the application itself or by the job, we focus our analysis between 25% and 75% of the runtime. After normalizing the timestamp and memory usage values by dividing them by their respective maximum values, we determine the linear trend of the memory usage by assigning the slope m and the y-intercept n of the linear function $f(x) = mx + n$ using NumPy's "polyfit"[1] method. For further memory leak investigation, the linear slope must be between 0.01 and 1. If the slope is less than 0.01, we consider the memory usage to be constant, if it is greater than 1, it is very likely that there is a very high abrupt slope and not a linear one. Note that for long jobs, up to 7 days, a rather small slope is expected, which is why we set the lower limit to $m = 0.01$. High slopes are to be expected for jobs that only have a few hours of runtime or run into "out of memory". Next we calculate the euclidean distance of all measuring points to the linear function. At the last step we determine the maximum of the euclidean distance vector, which must not exceed the maximum value *max_distance*. The *max_distance* value depends on the position of P, which is further to the left at higher slope, see the graphic and formula in Fig. 2. This allows to recognize both steady linear slopes and slopes in the form of stairs or zigzags.

Maximum Memory Leak describes the suspicion of a memory leak on the basis of the linear increase of the memory usage across all jobs. Internally we perform the heuristic tests described above for each job and attribute the slope m of the memory usage vector to the job if a memory leak is suspected. Finally, we provide the maximum slope m of all jobs.

[1] https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html.



$$max\_distance = d(P; g) = |\overrightarrow{PF}|$$
$$P(ts, mem\_used) = (p1_{ts} + ((p2_{ts} - p1_{ts}) * \frac{1}{m*10}), p1_{mem\_used})$$
$$if\ m < 0.1 \rightarrow P_{ts} = 1$$

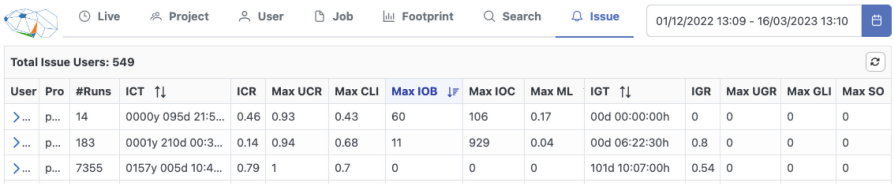
1. Divide each element of ts and mem_used by its maximum value.
2. Determine slope trend via $np.polyfit(ts, mem\_used, 1)$ to get the slope m and the y-intercept n of the linear function $f(x) = mx + n$
3. Determine $p1$ and $p2$ based on linear function for $m \in [0.01; 1]$
4. Calculate euclidean distance dis of each measuring point to the slope line
5. Determine P based on slope m and calculate distance $max\_distance$ for $m \in [0.1; 1]$
6. if $np.max(dis) < max\_distance \rightarrow$ suspected memory leak found

Fig. 2. Heuristic to detect a memory leak based on memory usage over time

3 HPC Support Case Studies

Administrators and the HPC support team of TU Dresden's HPC cluster have the possibility to get a weekly overview of HPC users who have sent a large number of inefficient jobs. For this purpose, they can use PIKA's new top-down approach as illustrated in Fig. 3. Starting from the issue table view with the ability to prioritize users by a specific performance issue metric, they can navigate from user issue data, to job issue data, to individual job issue data, and finally to a timeline view that reveals all identified performance issues in timeline charts. Figure 3a shows an interactive table of users with performance issues sorted by the number of I/O blocking phases. The issue data of an individual job run is shown in Fig. 3b. This view lists all job runs with associated job ID that belong to collection of a specific job name. Finally, Fig. 3c shows the job-specific visualization of all recorded performance metrics including the metadata and allows analysts to understand the root cause of the identified issues. At this point, the analyst also has the option to review the user's job submission script.

The following is a brief description of the user support process. Typically, our support team will contact users who have caused the same performance issues on a large number of jobs, or on a highly scalable job with up to 100 nodes. This is done using PIKA's interactive issue table, which is updated weekly by our analysis engine. Once a user with inefficient jobs has been identified, the first step is to create a problem hypothesis. It has proven very useful to provide the Slurm job scripts to the analyst to understand the root cause of performance issues and to derive helpful recommendations for the user. For example, by analyzing the job script and comparing it to the performance data collected by PIKA, analysts can check the job script for misconfigurations that may be causing a job inefficiency. To avoid the perception of spamming, we refrain from sending automated emails to our users. In an initial email, we make the user aware of the performance issue and provide initial instructions on how to resolve the



(a) Performance issue table showing HPC users sorted by the number of I/O blocking phases (Max IOB) according to the selected time period (top right). The third column (#Runs) contains the number of the corresponding user's jobs.

(b) Performance issue data of one job run. The ratio of idle and total CPU hours (ICR) is 53%. There are 60 phases with I/O blocking (IOB), a peak rate of 41 open+close operations (IOC), and a suspected memory leak (ML) with a linear slope of 0.17.

(c) A zoomed-in timeline visualization with six fully allocated Haswell nodes. One can see an inverse correlation between the CPU load (top left) and the read bandwidth (bottom left). The high CPU idle time ratio results from the I/O blocking phases as well as from CPUs that are almost idle, as can be seen in the orange graph (lowest). A memory leak can be assumed from the memory usage chart (top right). In addition, one can recognize a high rate of open operations across all nodes (right bottom).

Fig. 3. PIKA's top-down approach to analyze a performance issue job.

issue. Depending on the severity of the problem, we may also offer a face-to-face meeting to resolve the performance issues together.

4 Related Work

We have presented the core architecture of PIKA in [5], which did not deal with detecting inefficiencies as presented here. ClusterCockpit [6] is a web-frontend

based on the Likwid Monitoring Stack (LMS) [12] that provides a job issue detection in particular for pathological and inefficient jobs. TACC stats [7] automatically detects underperforming and misconfigured jobs. Julia Roigk describes in her master thesis [13] the determination of periodic resource utilization using FFT analysis. This work involves, among other things, creating a rule set to find HPC jobs that have distinct stages or periodic patterns. Monika Multani shows in her bachelor thesis [10] that system monitoring data, especially from the network area, often have a waveform, which can be determined by means of FFT. Jindal et al. specialize in the aspect of memory leak detection in cloud-based infrastructures based on the system's memory usage using machine learning algorithms [9].

In many publications, the authors have used machine learning techniques to classify jobs on the basis of performance data. Felix Fischer presents in his bachelor thesis [8] an approach for measuring the similarity of HPC jobs based on their hardware performance data using unsupervised clustering methods. Ozer et al. introduce an approach for characterizing performance variation in HPC systems using monitoring and unsupervised learning techniques [11].

5 Conclusion and Future Work

In this paper, we presented heuristics that enable administrators of TU Dresden's HPC cluster to identify users who have submitted a large number of inefficient jobs using the top-down approach in PIKA. A new analysis engine with heuristics and defined thresholds has been implemented to scan jobs for performance issues on a weekly basis. These heuristics can identify jobs that are using excessive idle CPU/GPU hours or have load imbalances, periodic blocking I/O phases, or suspected memory leaks. By detecting such performance issues, administrators can contact and advise HPC users on how to improve the performance of their jobs. By doing so, users can reduce their job runtime, freeing up resources for other users and increasing the overall efficiency of the HPC system. In addition, the assigned jobs with performance issues can be used to investigate former project proposals that have requested very high CPU/GPU hours. Both the allocated and idle CPU/GPU hours of jobs can also be assigned to a project. Taking this information into account, subsequent project proposals with the same request of CPU/GPU hours can be handled accordingly.

Further improvements to the PIKA web front-end are planned to better identify users with major performance issues. We intend to provide an additional severity column in the issue table that better prioritizes problem jobs according to defined characteristics, e.g., highly scalable or very long jobs. In addition, we plan to allow administrators to mark problematic jobs where users have already been contacted to see if future jobs have resolved those issues. Finally, we plan to enrich the recorded jobs with application-specific parameters to be able to classify jobs by application type. One promising tool for this task is XALT [4], which tracks job-specific executable information and the linkage of static shared and dynamically linked libraries.

No Blackbox AI Approach

The heuristics proposed in this paper allow an appropriate classification of inefficient jobs. The computational load is very manageable and the classification criteria are directly comprehensible for human analysts. We refrain from using this classified data set as starting point for machine learning approaches because this offers no additional benefits. The result would be another classification tool, yet our heuristics already provide a suitable tool. It would, however, cost substantial computational effort for training with little hope that inference would provide faster or better answers.

References

1. InfluxDB: Scalable datastore for metrics, events, and real-time analytics. https://github.com/influxdata/influxdb. Accessed February 2023
2. NumPy: The fundamental package for scientific computing with Python. https://numpy.org/. Accessed February 2023
3. SciPy: Fundamental algorithms for scientific computing in Python. https://scipy.org/. Accessed February 2023
4. Agrawal, K., Fahey, M.R., McLay, R., James, D.: User environment tracking and problem detection with XALT. In: 2014 First International Workshop on HPC User Support Tools, pp. 32–40 (2014). https://doi.org/10.1109/HUST.2014.6
5. Dietrich, R., Winkler, F., Knüpfer, A., Nagel, W.: PIKA: center-wide and job-aware cluster monitoring. In: Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications. HPCMASPA (2020). https://doi.org/10.1109/CLUSTER49012.2020.00061
6. Eitzinger, J., Gruber, T., Afzal, A., Zeiser, T., Wellein, G.: ClusterCockpit—a web application for job-specific performance monitoring. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–7 (2019). https://doi.org/10.1109/CLUSTER.2019.8891017
7. Evans, T., et al.: Comprehensive resource use monitoring for HPC systems with TACC stats. In: 2014 First International Workshop on HPC User Support Tools, pp. 13–21 (2014). https://doi.org/10.1109/HUST.2014.7
8. Fischer, F.: Metrics for job similarity based on hardware performance data. Master's thesis, Technische Universität München (2020)
9. Jindal, A., Staab, P., Kulkarni, P., Cardoso, J., Gerndt, M., Podolskiy, V.: Memory leak detection algorithms in the cloud-based infrastructure. CoRR abs/2106.08938 (2021). https://arxiv.org/abs/2106.08938
10. Multani, M.: Statistical characterization of HPC monitoring data (2021)
11. Ozer, G., Netti, A., Tafani, D., Schulz, M.: Characterizing HPC performance variation with monitoring and unsupervised learning. In: Jagode, H., Anzt, H., Juckeland, G., Ltaief, H. (eds.) ISC High Performance 2020. LNCS, vol. 12321, pp. 280–292. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59851-8_18
12. 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), pp. 781–784 (2017). https://doi.org/10.1109/CLUSTER.2017.115
13. Roigk, J.: Feasibility study for detecting different job stages using a system monitoring daemon. Master's thesis (2022)