Cluster Computing (2025)28:667
https://doi.org/10.1007/s10586-025-05509-4

An organizational framework for multiple monitoring tools to improve the correctness of high performance computing datasets

Xiaoxuan Luo[1] · Weiwei Lin[1,2] · Fan Chen[3] · Haocheng Zhong[1] · Keqin Li[4]

Received: 12 December 2024 / Revised: 7 May 2025 / Accepted: 7 May 2025

Abstract
High performance computing (HPC) has seen significant advancements and widespread application in recent years. To conduct a comprehensive analysis of HPC systems, modern supercomputing environment requires the deployment of multiple monitoring tools to collect diverse HPC datasets. However, existing monitoring systems lack adequate mechanisms to facilitate the coordination among these tools. When data from various tools are aggregated into a single dataset, issues such as time misalignment, increased anomalous data, and low data representativeness arise, leading to datasets that fail to accurately reflect the actual system status. To mitigate the impact of these data correctness issues in the analysis of HPC systems, this paper evaluates the factors affecting data correctness and proposes an organizational framework for integrating multiple monitoring tools. The framework incorporates the time synchronization, anomaly detection, and change point detection to reduce the adverse effects on data correctness during the HPC dataset generation. Extensive experiments demonstrate that the proposed framework outperforms commonly used methods in terms of missing data rates, anomaly percentages, and stability, offering more reliable data support for HPC system analyses.

Keywords System monitoring · Supercomputing · Data science · Anomaly detection

1 Introduction

With the advancement of scientific computing, the scale and complexity of high-performance computing (HPC) systems have progressively increased. To enhance the efficiency of modern HPC systems, researchers analyze the runtime data of applications and system components to gain a deeper understanding of their operational characteristics, thereby informing the design of hardware/software optimizations. The most notable examples are the TOP500 and GREEN500 rankings. Researchers worldwide are dedicated to improving the performance and efficiency of supercomputers running the HPL, showcasing the overall capabilities of their supercomputers [1–3]. In HPC production environments, system analysis plays a crucial role in achieving efficient job management and resource allocation [4–6].

In the aforementioned studies on HPC system analysis, the outcomes are primarily influenced not only by the design of the algorithms but also by the quality of the dataset. A broader range of data types can more comprehensively reflect the characteristics of the HPC system,

✉ Weiwei Lin
linww@scut.edu.cn

Xiaoxuan Luo
202210188550@mail.scut.edu.cn

Fan Chen
787078310@qq.com

Haocheng Zhong
cshczhong@mail.scut.edu.cn

Keqin Li
lik@newpaltz.edu

1 School of Computer Science and Engineering, South China University of Technology, 382 Outer Ring Road East, Guangzhou 511436, Guangdong, China

2 Pengcheng Laboratory, 2 Xingke 1st Street, Shenzhen 518066, Guangdong, China

3 School of Software Engineering, South China University of Technology, 382 Outer Ring Road East, Guangzhou 511436, Guangdong, China

4 Department of Computer Science, State University of New York, 1 Hawk Drive, New Paltz, NY 12561, USA

Published online: 03 September 2025

while fine-grained data enables more precise capture of temporal state changes [7]. Additionally, factors such as anomalous noise and missing values significantly impact the results of data analysis. In conclusion, a high-quality dataset is essential for effectively analyzing and optimizing HPC systems and applications [8].

Obtaining a high-quality dataset for complex modern HPC systems is a challenging task. HPC systems comprise multiple hierarchical data layers (as shown in Fig. 1), requiring the monitoring system to concurrently observe various components, including the job layer, parallel framework layer, and node hardware layer. As a result, HPC monitoring systems often integrate diverse monitoring tools and store data in different formats and with varying sampling intervals, typically as logs for subsequent analysis. However, due to the independent operation of these monitoring tools, correlating features from different tools within the HPC dataset to enable comprehensive system analysis becomes difficult. Furthermore, the dataset contains substantial redundant and anomalous data, with only a small portion of relevant data being valuable for analysis [9], which further complicates the analytical process.

In this paper, we propose a novel monitoring framework from the perspective of integrating multiple monitoring tools. The framework aims to enhance the correctness of HPC datasets, i.e., datasets that accurately reflect the true operational state of the HPC system, while minimizing misinformation that may distort the results of data analysis. We focus on improving dataset accuracy at the generation stage rather than relying on more advanced data processing techniques (e.g., data compression) applied to the generated data. Given the availability of numerous robust engineering implementations for large-scale network transmission, data storage, and system visualization [10], these aspects fall outside the scope of our work. Specifically, the contribution of this paper is as follows:

- We analyze the data correctness problems of existing HPC public datasets, using partial data in a popular HPC dataset M100 as a case study. Additionally, we evaluate the factors affecting data correctness at each stage of dataset generation.
- We found that the time alignment problem caused by multiple monitoring tools has a significant impact on the quality of the dataset. Therefore, we implemented a synchronization algorithm for multiple monitoring tools using Linux signaling mechanism to reduce the probability of data inconsistency.
- We analyze the generation principles of anomalous data in the HPC system monitoring scenario and propose an anomaly detection algorithm based on power modeling. The algorithm can accurately identify anomalous data due to hardware failures, inconsistent acquisition periods and other problems, thereby minimizing the occurrence of erroneous information in the dataset.
- We propose a data aggregation method based on change-point detection to solve data representativeness issue. The method effectively addresses the inconsistent data distribution caused by load changes, ensuring that the log data is representative of the overall operating status of the servers.

The rest of this paper is organized as follows. Section 2 reviews related work in HPC system monitoring and M100 dataset. Section 3 analyzes the factors that may affect the correctness of the HPC dataset. Section 4 details the design of our monitoring organizational framework. Section 5 shows the experiments in a specified cluster environment. Section 6 discusses the significance of this paper. Section 7 summarizes this work.

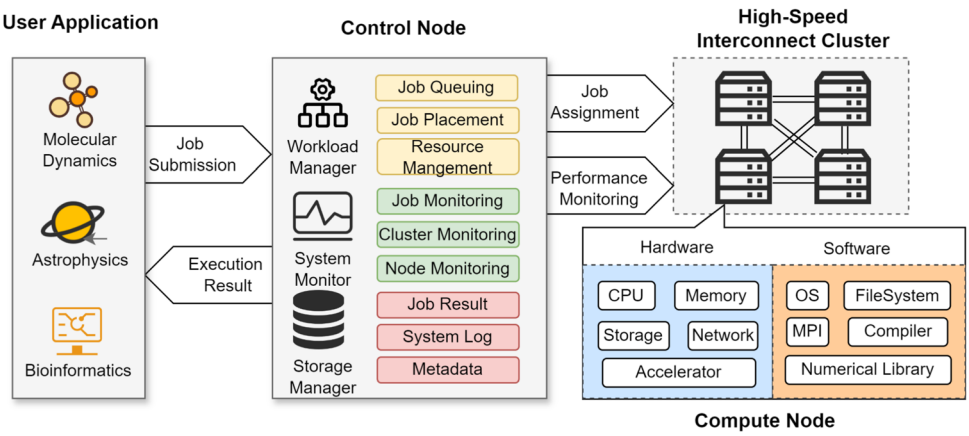


Fig. 1 A typical HPC system architecture diagram

2 Background

2.1 Monitoring tools

From the generation granularity, HPC metrics can be divided into serial metrics and aggregated metrics. Serial metrics are generated at a fixed time granularity, such as CPU utilization, power consumption, and other server-level data. In contrast, aggregated metrics are generated at granularities other than fixed time intervals, such as job status, runtime, and other metrics that produce a set of data for each HPC job. Aggregated metrics tend to have lower complexity and less potential for data quality improvement. This paper focuses on server-level serial metrics.

A wide variety of monitoring tools have been developed for acquiring different types of data to meet the different system analysis scenarios. Ganglia [11] is a mature tool widely used in HPC system monitoring. It efficiently collects information on the overall status of large-scale clusters, offering advantages such as high efficiency, low overhead, flexibility, and scalability. Collectd [12] is a lightweight system monitoring tool that implements a large number of independent plugins to support a wide range of monitoring functions. By adjusting the plugin types, it can easily adapt to different monitoring requirements. TACC Stats [13] is a monitoring tool developed by the Texas Advanced Computing Center, designed to collect and analyze various performance data in HPC systems, with a particular focus on resource utilization efficiency metrics. Darshan [14] is a monitoring tool dedicated to HPC I/O system analysis, providing a detailed view of I/O system performance and behavioral patterns. LIKWID [15] is a monitoring tool for hardware performance data, including performance counters, hardware topology and other information.

Table 1 summarizes the types of metrics collected by each tool, which are commonly used in various types of HPC system analysis and research. Including:

- Hardware Usage: CPU utilization, memory allocation, disk throughput, and other metrics that reflect overall load levels
- Sensor: power, temperature, fan speed, and other physical attributes of servers that need to be obtained using sensors
- PMU: instruction execution, cache hits, branch prediction, and other CPU micro-architecture information
- OS: average loads, number of processes, and other operating system information
- File System: capacity, bandwidth, latency, and other operational status of file systems used in HPC (e.g., Lustre)
- POSIX I/O: open count, read/write count, read/write latency, and other underlying file information
- Parallel I/O: bandwidth, data block size, number of parallel file operations, and other I/O information of parallel framework used in HPC (e.g., MPI)

It can be observed that no single tool is capable of covering all metric types. Due to varying design objectives, there are significant differences in the specific metrics accessible by different tools within the same metric category. When a comprehensive analysis of the HPC system is required, the concurrent use of multiple monitoring tools becomes necessary. Moreover, we found that none of the tools examined consider the collaboration between multiple tools, and even the multiple sub-monitoring tools integrated within some of the tools fail to achieve unified monitoring behavior. As a result, data patterns for individual metrics may appear normal, but when combined across multiple metrics, the data patterns may exhibit correctness issues (see Section 3 for further details).

2.2 Marconi100 dataset [16]

Marconi100 is a Tier-0 supercomputer hosted by CINECA. The M100 dataset encompasses its operational information over a span of more than two and a half years. This dataset is one of the most comprehensive datasets currently available in the HPC field. It includes not only system-wide information about more than 980 compute nodes but also

Table 1 Summary of the metric types that can be collected by the monitoring tools

Tool	Hardware Usage	Sensor	PMU	OS	File System	POSIX I/O	Parallel I/O
Ganglia	✓	×	×	✓	×	×	×
Collectd	✓	✓	•	✓	×	×	×
TACC stats	✓	×	•	✓	•	×	×
Darshan	•	×	×	×	✓	✓	✓
LIKWID	•	•	✓	×	×	×	×

• indicates the data has some limitations compared to other tools, e.g. Colletcd only supports PMU collection with partial Intel CPUs

data on liquid cooling infrastructure, air conditioning systems, power supply units, workload manager statistics, job information, system alerts, and weather.

The M100 dataset consists of data from nine plugins: Ganglia, IPMI, Job Table, Logics, Nagios, SLURM, Schneider, Vertiv, and Weather. As previous mentioned, this work focus on the correctness of the server serial data. Therefore, data from the Ganglia and IPMI plugins are emphasized. Ganglia records the hardware status of CPU, GPU, memory, etc.. Key metrics include GPU utilization, GPU memory utilization, CPU utilization, network throughput, and average load. IPMI records the out-of-band server sensor data. Key metrics include server power consumption, CPU power consumption, GPU power consumption, and fan speed. Most data from these two plugins are collected at 20-second intervals.

Several studies have been conducted to analyze HPC system behaviors using the M100 dataset, including anomaly diagnosis [17], thermal management [18], and job scheduling [19]. However, most works considered only a few metric for analysis and did not jointly analyzed the system status across multiple metrics. As a result, the impact of data correctness issues on the analysis results has been minimal. As the demand for global and comprehensive analysis of HPC systems grows, the issue of dataset correctness should be given greater emphasis.

To increase the generalizability of the study, some of the experiments were also repeated using the SURFsara [20] and Acme [21] datasets. SURFsara contains data collected by SURF's Lisa cluster from December 2019 to August 2020. Acme contains data for the Shanghai AI Lab cluster from March 2023 to August 2023. Both are commonly used datasets in the industry.

3 Dataset correctness

HPC datasets are typically derived from log data, which is transformed from fine-grained monitoring data. Monitoring data is primarily used to characterize the HPC system over short periods for real-time decision-making tasks such as hardware diagnosis and optimization strategy adjustment. In contrast, log data is typically used for detailed analysis, with a broader time span and higher data granularity. The quality of HPC datasets can be assessed from several perspectives. In this work, we focus on the data correctness, which refers to the ability of the log data to accurately reflect the server's operating status, unaffected by factors such as load fluctuations and monitoring anomalies. This chapter discuss the impact of assurance diversity metrics on log data correctness in scenarios where multiple monitoring tools are employed simultaneously. We demonstrate that the correctness issue in the HPC public dataset is evident through partial data in the M100 dataset. Then, we will analyze the factors influencing data correctness in terms of monitoring data collection and aggregation, two key processes in log data generation.

3.1 Log data correctness

A comprehensive HPC dataset should capture the system status at multiple levels to support various aspects of data analysis. In the context of this paper's discussion of server serial data, data diversity requires that the dataset to encompass the operational state of various hardware and software, such as CPU, memory, and file system. As noted in Sect. 2, no monitoring tool can completely cover all metrics. An effective way to enhance data diversity is by concurrently utilizing multiple independent tools to collect data from specific components and aggregating them during the dataset generation phase. This aggregation forms comprehensive data that characterizes the operational status of HPC nodes at multiple levels.

The primary challenge in monitoring with multiple tools is the time alignment issue, where the collection times of data points acquired by different tools may not coincide, resulting in data reflecting the state of the server in a confusing way [22–24]. For tools with inconsistent collection intervals, there are differences in the time at which data samples are generated. For example, for tools with collection intervals of 10 and 20 s, respectively, one out of every two data samples is missing the server state information obtained by the latter. Even when the tools have the same collection intervals, misalignment can still occur due to varying start times of tools. This further exacerbates the differences in the time points of the data samples obtained by the different tools. Based on our observation, this issue is prevalent in the M100 dataset. For example, when analyzing the data from June 2021, we concatenated all the data from the Ganglia and IPMI plugins based on collection time and found that the data missing rate exceeded 70%.

A common approach to handling inconsistencies in collecting time is to integrate data points from similar times together, such as sliding window aggregation, nearest neighbor aggregation [25, 26]. However, this method poses a problem as the status reflected by each monitoring tool may differ due to variations in collection times. If the workload carried by the server changes during this time difference, it can result in significantly divergent data being acquired by multiple tools. As a result, integrating these data points together does not accurately reflect the overall server status, especially for datasets with long collecting intervals. For instance, using data from Server No. 100 in June 2022, we integrated data points within a time span of 20 s and extracted several key metrics for analysis. Table 2

Table 2 A partial data from the M100 dataset with 20-second intervals

Timestamp	Power			Utilization	
	Total	CPU	GPU	CPU	GPU
1654052460	480	88	271.8	5.3	24.8
1654052480	480	84	236.7	0.1	21.2
1654052500	460	92	161.5	0.1	11.7
1654052520	1620	120	175.2	4.5	13.3
1654052540	660	118	190.3	4.5	13.3
1654052560	760	116	523.3	4.5	50.4
1654052580	740	244	477.1	4.5	71.8
1654052600	640	90	437.5	4.5	48.0
1654052620	900	120	597.6	4.3	49.0
1654052640	740	252	600.5	4.3	77.1
1654052660	680	124	493.4	3.9	74.6

presents a partial data, which reveals multiple inconsistent information. At timestamps 1654052520 and 1654052540, the total power consumption of the server with timestamp 1654052520 consumes more than twice as much as the other, while the remaining metrics are nearly identical. At timestamps 1654052560 and 1654052580, CPU power doubles while CPU utilization remains nearly unchanged; GPU power decreases by 50W while GPU utilization increases by 20%. At timestamps 1654052640, the cumulative power of the CPU and GPU has exceeded the total power. The behavior of the metrics for these data points contradicts common sense and differs significantly from the patterns observed in other data.

To quantitatively analyze the proportion of such data and its impact on the analysis, we construct three power models using data from No. 100 Server in June 2021. CPU and GPU are the most energy-consuming components of the server, and an increase in their power consumption typically indicates increased load pressure on the server, which is accompanied by higher power consumption from other server hardware. Consequently, the server power model takes CPU power and GPU power as modeling features. For both the CPU and GPU, utilization serves as a critical indicator of load pressure, and the feasibility of utilization-based power modeling has been validated by numerous studies [27]. Accordingly, the CPU power model uses CPU utilization as the modeling feature and the GPU power model uses GPU utilization as the modeling feature. All three models are fitted using polynomial functions.

To ensure accurate modeling, we use relatively stable data and divide it into a training set and a validation set in an 8:2 ratio. Finally, we use all data (including unstable data) as the test set. Stable data is defined as data in which the maximum fluctuation of any metric across five consecutive points does not exceed 20%. Stable data accounts for 24.9% of the total dataset, covering a variety of power distributions ranging from 360W to 1260W for the server, from 28W to 366W for the CPU, and from 158W to 932W for the GPU. Table 3 summarizes the percentage of training error, validation error, and test error greater than a specific value for the three power models. It can be observed that while the validation error closely approximates the training error, the prediction error for most test data significantly exceeds the training error. Notably, 11.7% of the data exhibits an error of over 50% in predicting at least one type of power. Although this percentage appears small, considering the high percentage of redundant data (about 1/4 of the data in this dataset is stable) and the limited scope of evaluated metrics, the impact of inconsistent data can be more severe than these results suggest. In addition, we directly divide the entire dataset into training and test sets in the ratio of 8:2 to evaluate the impact of datasets containing inaccurate information on power prediction. The test errors of the new server power model, CPU power model and GPU power model are 12.8, 25.2 and 14.1%, respectively. Compared with the validation errors of the original models, the model accuracy is significantly degraded.

Table 3 Error metrics for power measurements in different datasets

Dataset	Metric	Total	CPU	GPU
M100	Training Error	3.9	13.8	6.6
	Validation Error	4.0	14.2	8.2
	Test Error>10%	29.8	55.1	37.6
	Test Error>20%	15.7	23.0	15.3
	Test Error>30%	10.2	10.4	7.6
	Test Error>50%	4.9	5.6	2.6
	Test Error>80%	1.9	4.1	0.9
SURFsara	Training Error	15.5	–	25.9
	Validation Error	15.6	–	26.1
	Test Error>10%	67.5	–	78.3
	Test Error>20%	37.8	–	55.7
	Test Error>30%	21.8	–	40.3
	Test Error>50%	8.0	–	17.8
	Test Error>80%	2.3	–	7.9
Acme	Training Error	15.9	2.3	–
	Validation Error	16.0	2.3	–
	Test Error>10%	46.3	24.9	–
	Test Error>20%	30.7	15.2	–
	Test Error>30%	17.5	7.2	–
	Test Error>50%	8.3	1.2	–
	Test Error>80%	2.4	0.2	–

The experiments are also conducted on the SURFsara and Acme datasets. Since the SURFsara dataset does not provide CPU power, CPU utilization, or GPU utilization, we use the one-minute average load (node_load1) as the feature to build both the server power model and the GPU power model. The accuracy of the model built in this way is of course not as high as the one built using the utilization, but similar experimental results can be reflected by the change in accuracy in different data sets. Additionally, the SURFsara dataset spans a long period with most of the time in an idle state. A simple baseline using the power value of the previous record as the prediction achieves over 90% accuracy. To avoid unreasonable errors caused by excessive redundant data, we applied a 10% sampling rate to obtain a streamlined dataset. In the Acme dataset, GPU power data is not available. Therefore, the experiments build the server power and CPU power modeling. The results from both datasets are consistent with those observed on the M100 dataset, supporting the generalizability of our experimental conclusions.

3.2 Monitoring data correctness

Besides the time alignment problem of data points, the monitoring data itself also affects the correctness of the log data. If the monitoring data contains incorrect information, it cannot reliably reflect the server status even if the collecting time of multiple tools are close. Factors such as hardware design, monitoring configurations, and sensor failures can lead to inherent inaccuracies in the raw data collected by monitoring systems [28–30]. These inaccuracies are often highly randomized and manifest as outliers. Table 4 provides an example of an outlier anomaly, where the second data point exhibits an order-of-magnitude increase in the number of cycles without corresponding significant changes in other metrics. Removing such outliers prevents excessively biased data from skewing the overall performance of averaged data.

However, not all outliers are anomalies, particularly for periodic IO metrics. Disk reads and writes, file system accesses, network communications, etc., often accumulate data until a certain threshold is reached before triggering, resulting in data spikes. For such outliers, it is obviously not reasonable to directly eliminate them; however, using the spike directly as a log data point is also not representative of the server's overall status. In any case, outlier data must be identified, and further analysis is needed to determine whether to eliminate them as anomalies.

Outliers can occur in various system monitoring scenarios. Based on our observations, a new anomaly emerges when multiple monitoring tools are used simultaneously, which we refer to as shift anomalies. Shift anomalies resemble the time alignment problem discussed in the previous section, except that they occur at a much smaller temporal granularity. Figure 2 illustrates the principle behind the generation of shift anomalies. Within a single collection period, multiple collectors do not gather metrics at the strictly same period due to differences in the timing of initiation as well as the response speed. This deviation between collection periods causes data to be shifted when the workload changes. In the depicted example, collectors A and B both acquire one-second metrics as monitoring data points. When the compute node completes its job at the time shown in the figure, the majority of data collected from collector A reflects the compute node's status during its workload execution. Conversely, the data from collector B mirrors the compute node's performance during idle periods. As a result, the collected data exhibits numerical variations similar to those shown in Table 5.

The table shows four performance metrics collected by three monitoring tools (with instructions and cycles originating from the same tool). We use a matrix computation job that periodically reads data from disk and computes the matrix multiplication result using multiple cores as the workload. The job ends at the timestamp 1712124614. The data before and after this point can accurately reflect the compute node status. However, at the moment the job ends, the power metric still indicates a full load status. The instructions, cycles, and CPU utilization show a low load status but with slightly different load levels. Since their values are still within the normal range, the hidden nature of shift data causes it to contain error messages that are difficult to detect and are misinterpreted as valid information for analysis.

Table 4 A data segment with an outlier anomaly

Timestamp	Power	Instructions	Cycles	CPU
1713000871	384	139487706639	18333224668	70.14
1713000872	360	187092397498	133521710981	61.59
1713000873	378	150124836103	18833479629	64.65

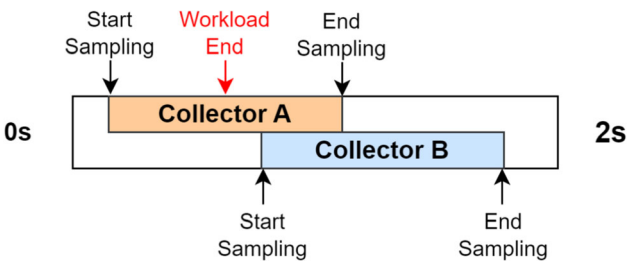


Fig. 2 Schematic diagram of shift data generation principle

Table 5 A data segment with a shift anomaly

Timestamp	Power	Instructions	Cycles	CPU
1712124612	396	128658957468	241130150965	95.89
1712124613	402	132943574789	247098868730	97.78
1712124614	396	19105036922	33946416640	21.58
1712124615	294	65342876	600040392	0.02
1712124616	294	221420944	778422372	0.05

To demonstrate the prevalence of shift anomalies, we also conducted the above experiment using Ganglia and Collectd, as shown in Table 6 and Table 7. CPU utilization and memory usage are collected in Ganglia. The timestamps 1745745825 and 1745745885 have obvious inconsistent information. The former maintains idle memory usage when utilization increases significantly, and the latter maintains a high level of memory usage when utilization decreases significantly. Disk throughput, CPU utilization and memory usage are collected in Collectd. The timestamps 1745752920 and 1745752990 have obvious inconsistent information. In the former case, the memory usage is close to its peak when the load has just been started, and in the latter case, the memory usage is close to its idle value while the computation is still in progress.

Shift anomalies usually occur during load switching. In this experiment, when a sub-matrix completes the computation and loads a new matrix into memory, or when the new matrix is loaded and the computation is started, the probability of shift anomalies will increase significantly. Since in real HPC environments, job executions are not completely stable, even if no load switching occurs. Normal load fluctuations may also cause some tools to collect data at relatively high load levels and some tools to collect data at relatively low load levels. Table 8 shows CPU utilization data for a segment of the computation using Top, Ganglia and Collectd, where Top contains the cpu_sys, cpu_usr and cpu_idle, while Ganglia and Collectd contain the cpu_usr and cpu_idle utilization respectively. From the utilization in Top, the sum of the three types of utilization is equal to (or close to) 1. However, if we take cpu_sys from Top, cpu_usr from Ganglia, and CPU(idle) from Collected, there are significant fluctuations in the sum of the three utilization. In general, CPU utilization collection is not given to multiple tools to complete at the same time. This experiment is just to show the possible effect of data shifting in load fluctuation, since the characteristic of sum of three CPU utilization is 1 can be used to determine the specific data shifting. We do not have precise criteria for determining data from multiple hardware sources, for example, we do not assume that a 20% drop in CPU utilization is accompanied by a 20% drop in memory usage. From the analogy of multiple tools obtaining different CPU utilization to multiple tools obtaining data from multiple hardware, load fluctuations can have some negative impact on their joint information, even if the impact is not so large as to be visible as to be judged anomalous.

Table 6 Shift anomalies in Ganglia

Timestamp	CPU utilization	Memory (used)
1745745810	0	26.16GB
1745745825	46.76	26.16GB
1745745840	50.1	40.69GB
1745745855	58.22	43.44GB
1745745870	97.25	57.83GB
1745745885	0.66	57.85GB
1745745900	0	29.20GB

Table 7 Shift anomalies in collected

Timestamp	Read	Write	CPU	Memory (used)
1745752910	0MB	3MB	0.21	23.77GB
1745752920	106MB	82MB	9.29	34.98GB
1745752930	454MB	82MB	33.09	36.28GB
1745752940	0MB	112MB	39.86	37.83GB
1745752950	0MB	95MB	40.38	30.34GB
1745752960	0MB	82MB	37.29	31.58GB
1745752970	0MB	82MB	30.42	36.47GB
1745752980	0MB	82MB	40.98	33.98GB
1745752990	0MB	84MB	39.53	25.74GB
1745753000	0MB	113MB	11.92	23.51GB
1745753010	0MB	28MB	1.34	24.17GB

3.3 Log data representativeness

Converting monitoring data into log data involves an aggregation process that reproduces the HPC system over a period of time through multiple discrete data points. Therefore, a major indicator of log data correctness is representativeness, i.e., the ability to accurately reflect the system's overall status.

The most common approach for generating log data data points is to directly average the monitoring data over the aggregation period. Ideally, this method handles minor load fluctuations and provides a consolidated view of server status. However, in complex HPC systems, simple averaging fails to mitigate the effects of the unbalanced

Table 8 CPU utilization shift in load fluction

Timestamp	cpu_sys (Top)	cpu_usr (Top)	cpu_idle (Top)	cpu_usr (Ganglia)	cpu_idle (Collectd)	Sum (Top)	Sum (Hybrid)
1745750535	1.40	52.86	45.74	59.93	42.74	100.00	104.07
1745750550	1.54	48.58	49.88	41.18	43.35	100.00	86.07
1745750565	0.66	47.16	52.12	54.00	62.19	99.94	116.85
1745750580	1.25	47.23	51.52	41.77	57.95	100.00	100.97
1745750595	1.20	31.92	66.88	30.86	60.60	100.00	92.66
1745750610	1.03	54.03	44.90	57.81	49.85	99.97	108.69
1745750625	3.42	29.58	66.94	27.60	57.87	99.94	88.89

distribution of monitoring data. When the load state changes during the aggregation period, the monitoring data may exhibit multiple data patterns, at which time the averaged data may deviate severely from any of them, as illustrated in Fig. 3.

Even though aggregation of data using the averaging method results in a significant difference between the data distribution and the actual distribution, the impact of this difference on data analysis is dependent on the specific scenario. Table 9 shows the accuracy of building the server power model, CPU power model, and GPU power model after averaging every 30 stable data. Comparison with Table 3 shows that the accuracy of the aggregated models is not significantly different from the original models. This is because although the distribution of the data changes, the mapping relationship between the data and the power remains the same.

However, the impact of data aggregation will be more pronounced for some resource use statistical analysis studies [4, 31, 32]. Figure 4 shows the distribution of CPU utilization and GPU utilization for the M100 dataset before and after the average data aggregation. After data aggregation, lower utilization levels tend to exhibit value increases (more apparent in CPU), whereas higher utilization levels tend to exhibit value decreases (more apparent in GPU). In most case, the aggregated data shows substantial distributional deviation from the original data. If the cluster manager performs operations such as job scheduling, resource sharing, and power management based on the aggregated data, there is a risk of server overloading and thus affecting the performance of user jobs. It should be noted that due to the lack of more granular monitoring data from datasets such as M100, the above experiments can only be realized by further aggregation on log data. However, the overall logic is consistent with the process of obtaining log data by aggregating monitoring data.

In summary, factors in various aspects such as time alignment, raw data anomalies, and aggregation methods affect the correctness of the log data. Since a single monitoring tool does not need to consider the time alignment issues and shift anomalies, while having low data dimensions and outlier probabilities, the correctness issue is less serious. However, single data is difficult to support in-depth HPC system analysis, and multi-source data plays an increasingly important role [33]. For HPC datasets generated using multiple monitoring tools, the impact of the above factors will be magnified. Enhancing the correctness of HPC datasets is essential for advancing HPC data analysis research.

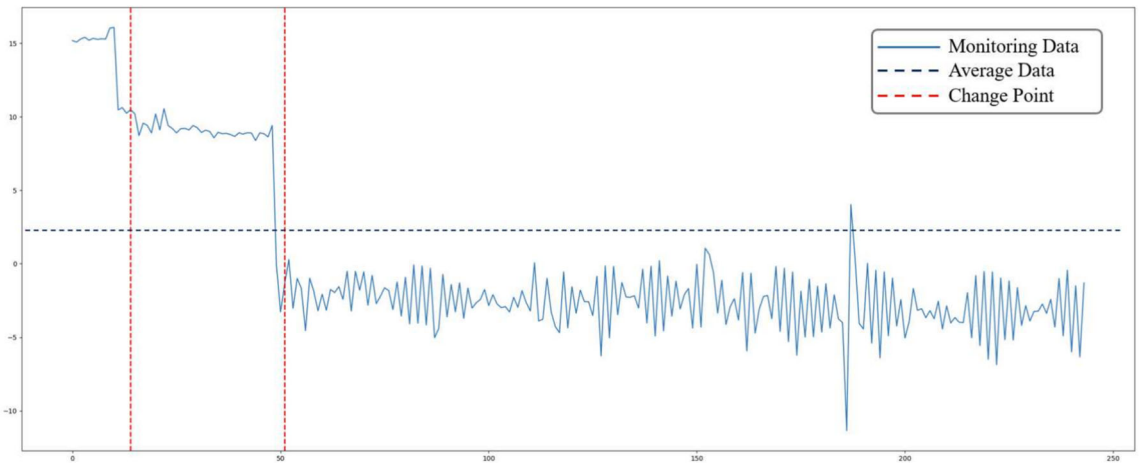


Fig. 3 Monitoring data from the LAMMPS program during execution, reduced to one dimension using PCA

Table 9 Error metrics for power measurements using data aggregation

Dataset	Metric	Total	CPU	GPU
M100	Training error	4.3	12.1	6.8
	Validation Error	4.4	14.1	8.7
	Test Error>10%	31.6	55.3	38.2
	Test Error>20%	14.8	22.1	14.9
	Test Error>30%	9.6	10.1	7.6
	Test Error>50%	4.7	5.4	2.5
	Test Error>80%	1.8	4.1	0.9
SURFsara	Training error	14.8	–	26.3
	Validation Error	15.6	–	28.4
	Test Error>10%	54.1	–	73.6
	Test Error>20%	24.8	–	49.9
	Test Error>30%	13.0	–	35.6
	Test Error>50%	4.9	–	18.4
	Test Error>80%	1.7	–	7.9
Acme	Training error	14.2	2.4	–
	Validation Error	15.0	2.4	–
	Test Error>10%	43.6	25.3	–
	Test Error>20%	26.7	12.1	–
	Test Error>30%	15.2	9.2	–
	Test Error>50%	6.9	1.5	–
	Test Error>80%	1.9	0.4	–

4 Framework design

Individual monitoring tools are well established. This framework does not take into account the development of custom monitoring tools, but focuses on the organization of multiple monitoring tools. Many factors that influence data correctness are difficult to avoid during the collecting process. We recommend using fine-grained collection intervals to provide more information for guiding data processing, and aggregating multiple monitoring data points into a single log data point to reduce storage overhead. Additionally, averaging allows for normal spikes (such as the IO metrics described above) to be distributed over over the entire collection period, resulting in more accurate information.

Although our framework increases the monitoring overhead, for mature tools that are designed for 24/7 monitoring, the running overhead has been kept within acceptable limits [14, 34, 35]. Thus, the overhead of our framework remains within acceptable limits, a conclusion that will be further verified in the subsequent evaluation section.

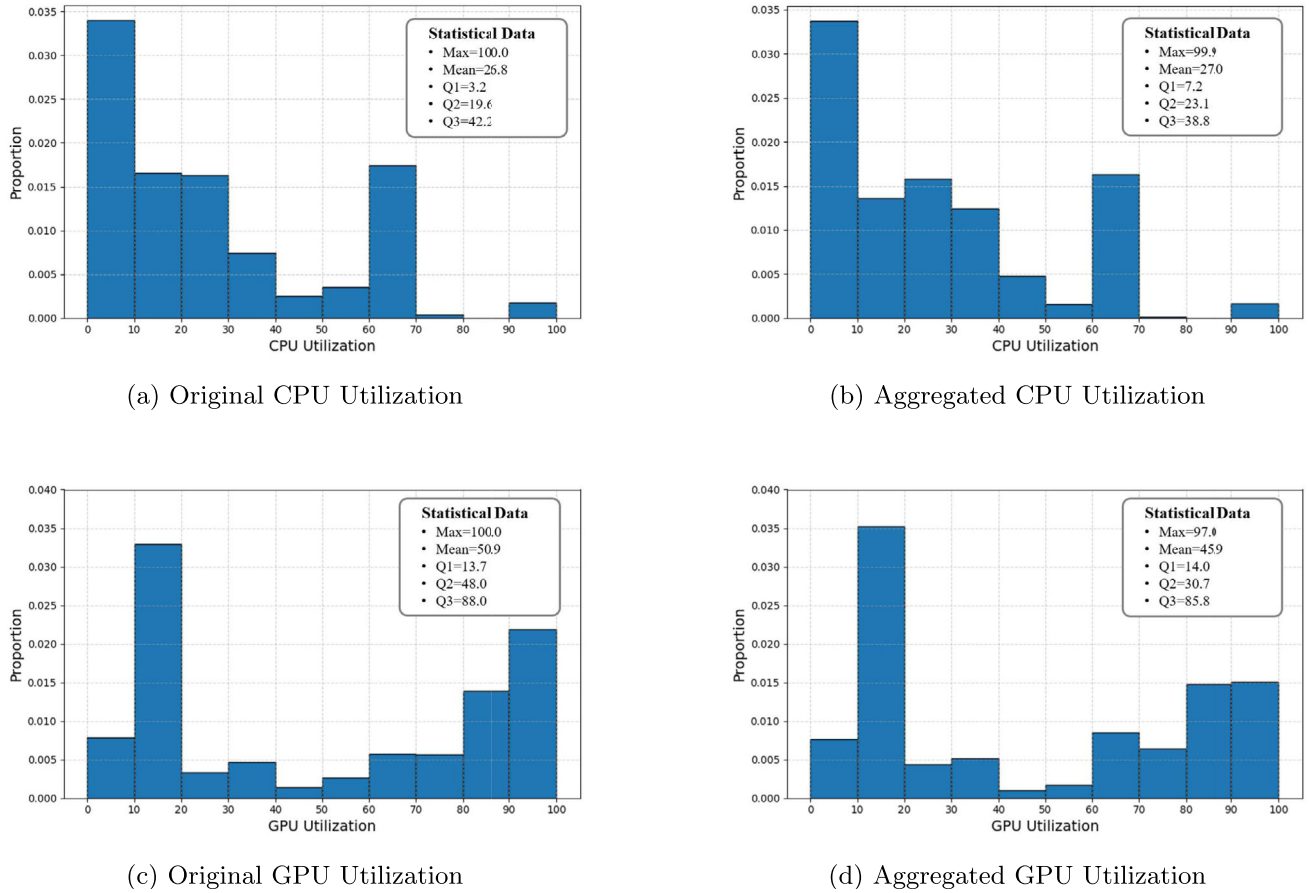
4.1 Time alignment

Since the time alignment problem cannot be solved at the data processing level, we can only ensure that the collecting time of different tools are as close to each other as possible. A logical solution would be to start all tools simultaneously. However, even if all tools are launched exactly at the same time, various factors can still cause the monitoring data to be misaligned.

First, monitoring tools need to perform preparatory tasks such as warm-up, cache initialization, and kernel module loading before collecting data [36]. The time required for these preparations varies across tools. resulting in the first data collection times differing even if the tools are started simultaneously. For example, on our experimental platform, it takes approximately 0.4 s of preparation time to fetch 16 PMU events, including instructions, cycles, LLC-loads, etc. using perf, whereas it takes about 1.6 s of preparation time to monitor similar PMU events using LIKWID. Although these values are seriously affected by the environment, they still reflect the differences in the preparation time of various monitoring tools, particularly when the metrics collected by different monitoring tools very significantly.

Secondly, the monitoring tool does not strictly follow the configured time interval during the monitoring process. Due to factors such as the time to perform collection, process scheduling delays in the operating system, and inaccurate timers, each data acquisition tends to take longer than the configured time interval [37, 38]. In particular, server load may have a significant impact on the monitoring system. On our experimental platform, when using Perf to capture 16 PMU events per second in an idle state, the average interval for each collection is approximately 1.02 s. However, when performing the same monitoring operation during the execution of an HPC program, the average interval for each data collection increases to around 1.08 s. This means that during the job execution, a piece of monitoring data is missed approximately every 15 s (since the 1.2-second delay exceeds the configured data acquisition interval by 1 s). As the monitoring continues, the time of the data points from multiple monitoring tools gradually shift because of the differences in the latency.

In summary, due to the differences in monitoring initialization and monitoring process, there is no method to make the time of data points obtained by multiple tools aligned over extended periods. Our solution is to continuously correct this misalignment during the monitoring process using the SIGSTOP and SIGCONT signals, ensuring that the moments when log data points are generated by each monitoring tool are synchronized. SIGSTOP



(a) Original CPU Utilization (b) Aggregated CPU Utilization

(c) Original GPU Utilization (d) Aggregated GPU Utilization

Fig. 4 Numerical density plot of CPU utilization and GPU utilization in the M100 dataset before and after data aggregation

and SIGCONT are signals used for process control in Linux. When a process receives the SIGSTOP signal, it saves its current state and halts. It resumes execution upon receiving the SIGCONT signal. These signals are widely used for multi-process coordination [39] and are also well-suited for scenarios where multiple monitoring tools need to operate in synchronization.

Algorithm 1 Time synchronization algorithm for multiple monitoring tools

Input: Monitoring tool set M, monitoring tool collecting interval I_{tool}, monitoring data generating interval I_{data}
Run all the monitoring tools in M simultaneously to get the process set P
Set T_{start} to the current time
The next log data time $T_{next} = \lceil T_{start} + 2I_{data} \rceil$
while is monitoring **do**
  Set T_{start} to the current time
  if $T_{start} >= T_{next}$ **then**
    $T_{next} = T_{next} + I_{data}$
  end if
  for ith process in P **do**
    Get the collecting time T_i of the latest monitoring data
    Hang time $T_h = T_{next} - T_i - I_{tool}$
    Send SIGSTOP to P_i and send SIGCONT after T_h (non-blocking)
  end for
  Set T_{end} to the current time
  Sleep $I_{data} - (T_{end} - T_{start})$
end while

The time synchronization mechanism is shown in Algorithm 1. The algorithm is used to correct the run-time behavior of the tools and therefore does not contain any output. Our framework sets the monitoring tool collecting interval I_{tool} slightly larger than the actual time interval for generating monitoring data I_{data}. During the startup phase, all monitoring programs are launched simultaneously, and the timing of subsequent monitoring data is determined. In each monitoring period, the delay required to generate the next data is calculated based on the timestamp of the most recent data. To prevent ant single synchronization operation from consuming excessive time, the delay operation is implemented by non-blocking SIGSTOP and SIGCONT singles. Specifically, the signals are sent outside the synchronization algorithm, ensuring the synchronization process does not block the monitoring operations.

The above method continuously calibrates the data collection times during monitoring, ensuring that fine-grained monitoring data from different tools is generated within a similar time frame. This mechanism significantly reduces inconsistencies caused by the time alignment problem and mitigates the adverse effects on subsequent

analyses, such as anomaly detection and change-point detection.

4.2 Anomaly detection

The anomalies discussed in this section differ from those addressed in existing studies. Most HPC anomaly detection studies focus on identifying job performance anomalies caused by incorrect configurations or hardware inefficiencies [40, 41]. In the context of HPC system monitoring, such anomalies accurately reflect the actual operating status of the HPC system, even if the operating results deviate from the user's expectations. For example, if a user misconfiguration causes a job that should use 30 CPU cores to use only 1 CPU core. From the monitoring system's perspective, the low power consumption and CPU utilization are reasonable. Such data can accurately reflect the server's operational status and allow administrators to identify the presence of application-level anomalies that do not meet the user's expectations. In our view, the anomalies that should be eliminated from the dataset are those that fail to accurately represent the system status, such as the outlier anomaly and the shift anomaly described above. These inaccuracies introduce erroneous information into the dataset, undermining the accuracy and reliability of subsequent data analyses.

Outlier anomalies and shift anomalies share the characteristic that a few metrics contain erroneous information significantly deviating from the server status reflected by other metrics. Based on the this property, we propose an anomaly detection algorithm leveraging multiple power models. Specifically, we train a power model for each monitoring tool to establish a mapping between metrics and server power. This design is motivated by the absence of shift anomalies within the metrics collected by a single monitoring tool, as these metrics are typically gathered simultaneously (assuming they are fetched by the same process). Differences in power predictions arise only when comparing metrics across different monitoring tools.

For training data, we smooth the monitoring data to reduce the interference of anomalous data on the model, which is a common method of unsupervised anomaly detection to acquire training data [42, 43]. Regarding the fitting algorithm, we adopt ridge regression, as our focus is on the differences between power models rather than absolute accuracy in power prediction. Ridge regression smoothes the metric weights, preventing extreme values while maintaining sensitivity to all relevant metrics. This dual effect ensures that load fluctuations do not disproportionately influence power predictions while still capturing the impact of less significant metrics, allowing their anomalies to be reflected in the power predictions.

When generating log data using multiple monitoring data sources, we utilize multiple power models to predict the server status from the perspectives of different monitoring tools (Algorithm 2). If there is a significant spike in the power variation of a single monitoring tool, it indicates that at least one metric is considerably higher than the others for that data point. For monitoring tools with metrics exhibiting periodic characteristics (e.g., file system reads and writes), such spikes are treated as reasonable and retained. Otherwise, they are removed as outlier anomalies.

To identify shift anomalies, we take the predicted power sequence from one monitoring tool as the baseline and evaluate whether other monitoring tools have shifted from it. Not all monitoring tools can reflect the actual power status. For instance, when the server runs a single-node program, the power model from network monitoring tool may output low values. To address this, we analyze relative power data to identify anomalies. Significant peaks or valleys in the relative power indicate that the power ratios for certain tools deviate markedly from the rest of the data within a sliding window, signaling a shift anomaly.

Algorithm 2 Anomaly detection algorithm based on multiple power models

Input: Monitoring tool set M, monitoring data D, power model set P
Output: Anomaly set A
for ith tool in M **do**
  Get the corresponding data D_i in D
  Predict power sequence S_i using P_i and D_i
  Calculate the z-score for each power value with a window size of 10
  for jth value v in power z-score sequence Z_p **do**
    if $v > 3$ and M_i not periodic **then**
      Add index j to A
    end if
  end for
  Calculate relative power sequence $\hat{S}_i = S_i / S_0$
  Calculate the z-score for each relative power value with a window size of 10
  for jth value v in relative power z-score sequence Z_r **do**
    if $v > 2$ and M_i not periodic **then**
      Add index j to A
    end if
  end for
end for
return A

Spikes in the power sequences and relative power sequences are identified using the z-score, calculated within a sliding window. The window size is affected by the monitoring data collection interval and the log data generation interval. The z-score threshold varies depending on the type of monitoring tool and the metrics being analyzed. On our experimental platform, variations in the window size and threshold within reasonable ranges have minimal impact on detection accuracy. According to our practice, power differences caused by outlier anomalies are generally larger than those from shift anomalies. Therefore, slightly lowering the z-score threshold for relative power sequences improves detection performance.

By employing the above algorithm, our framework effectively mitigates the error information introduced by outlier and shift anomalies, thereby enhancing the correctness of the log data. Additionally, due to the simplicity of ridge regression and the relatively small data volume, this process incurs minimal overhead when applied in an online environment.

4.3 Aggregation

After time synchronization and anomaly detection, the correctness of the monitoring data is significantly improved. Guaranteeing the correctness of log data primarily depends on the aggregation process (Algorithm 3). As discussed in Sect. 3.3, the main factor affecting the data correctness during the process of monitoring data aggregation into log data lies in the inability of simple smoothing operations to handle multiple data distributions effectively. When server load states change during the log generation periods, directly averaging all the monitoring data may result in the log data completely deviating from the actual server status. While averaging may represent the overall status, considering the long runtime of HPC applications, the server status after a load change will still be captured by subsequent log data. To avoid a temporary fluctuation being misrepresented as a new data pattern in the log, it is more reasonable to use the segment with the largest percentage of data as the log data, rather than the averaged data.

To identify the status changes of the server, our framework incorporates the Windows change point detection algorithm. This algorithm uses a sliding-window to calculate differences in metrics such as the mean and variance of data points, effectively identifying moments when significant changes occur in the time series data. Change point detection techniques are commonly employed in signal processing, and the Windows algorithm is particularly well-suited for online identification of server status changes due to its low computational complexity [44].

If the server's load state undergoes significant changes, applying the Windows algorithm to the monitoring data sequence will generate multiple change points, thus dividing the sequence into several sub-monitoring data segments. The longest sub-sequence is then averaged to produce the log data. Since anomaly detection has been performed, the probability that the Windows algorithm will misclassify the mutated monitoring data as change points is low. Furthermore, mutated data segments are typically short in duration, making it unlikely for them to qualify as the longest sub-sequence for generating log data.

Algorithm 3 Data aggregation algorithm based on Windows change point detection.

Input: Log data time interval T_{log}, sliding-window size S, serial monitoring data D_{mon}
Output: Log data D_{log}
1: Decompose D_{mon} to subsequences $D_1, D_2, ..., D_n$ according to T_{log}
2: **for** ith subsequence D_i in D_{mon} **do**
3: Calculate the variance of D_i under each sliding-window of size S
4: Get the change point set C based on trends in variance
5: Split D_i based on C
6: Select the longest period in D_i and get average data D_i'
7: Add D_i' to D_{log}
8: **end for**
9: **return** D_{log}

The method described above effectively addresses the issue of log data representativeness arising from multiple data distributions. Consequently, the log data can reflect the operating status of the server in the vast majority of the time, avoiding situations where transient load changes create misleading patterns in the log dat.

5 Evaluation

5.1 Experimental environment

Due to the unavailability of more granular monitoring data for the M100 dataset, we use a self-built environment to evaluate the performance of the proposed monitoring framework in terms of data correctness. Table 10 outlines the hardware and software configuration. The test setup consists of a five-node cluster simulating an HPC system

Table 10 Experimental environment

Node Num	5
CPU	Kunpeng920 7260
Memory	16*HMA84GR7JJR4N-WM (32GB)
Main Disk	SAMSUNG MZ7LH480 (480GB)
Network	HNS GE/10GE/25GE RDMA Network Controller
OS	CentOS 7.7
Kernel	linux 4.18.0
MPI	MPICH 4.1.2
Workload	LAMMPS 2023-Aug-Stable2

environment, where one node functions as the control node and the remaining nodes serve as compute nodes. Since our study is focus on single server, aiming to improve the accuracy of the data reported by the server to the cluster manager. Therefore, the problems do not become progressively more complex as the cluster size increases. For the test workloads, the platform uses MPICH as the parallel framework to run five samples (Chain, EAM, LJ, Chute, and Rhodo) from the atomistic molecular simulation software LAMMPS. It is important to note that the workloads used in the experiments are solely intended to generate varying monitoring data, and workload analysis is not part of this study. Therefore, the experiments involve simply repeating the five examples to construct the HPC job environment.

Based on the experimental environment, we selected the following tools for acquiring monitoring data (169 features in total) from the HPC compute nodes:

- **Redfish:** node power, fan power, inlet and outlet temperature, core and memory temperature, fan speed.
- **Procfs:** CPU utilization, memory usage, disk throughput, disk queuing, disk utilization, network throughput, network packet, NUMA status.
- **Perf:** instruction execution, context switch, cache, branch prediction, TLB, prefetch, stall.

5.2 Data overview

The experiment used five LAMMPS examples, with each example repeated five times to simulate load flow. Monitoring was initiated during the program runtime. We tested three monitoring and log generation methods separately. The first method follows the approach used by most monitoring tools, where monitoring data is collected at fixed time intervals (30 s in this experiment) and saved directly as log data. The second method collects monitoring data at a fine-grained time interval (1 s in this experiment), and then averages all the monitoring data within a fixed time interval (30 s in this experiment) to generate log data. Each tool operates independently in this method. The third method introduces the data synchronization mechanism from our framework (excluding anomaly detection and change point detection), building upon the second method. In this case, the monitoring tools collect data at 1-s intervals, while the log data is generated at 2-s intervals.

Table 11 summarizes the collection results of each method, including the number of data rows collected by each tool, the missing rate of merged data, and the CPU occupied by the monitoring program. The missing data rate ($NA\%$) refers to the percentage of rows with missing metrics after merging the data collected by the three tools based on time. The CPU overhead is calculated by measuring the single-score occupancy of all monitoring tools, along with any overhead from higher-level monitoring organization programs. While there are fluctuations in the execution of the load flow, the run times remain similar overall: 23,395 s for the coarse-grained collection method, 23,357 s for the unsynchronized fine-grained collection method, and 23,388 s for the synchronized fine-grained collection method. For comparison, the load flow without monitoring took 23,360 s. Given the low CPU occupancy and minimal differences in load flow runtime, the additional overhead introduced by monitoring is deemed acceptable for HPC applications. Considering the low memory occupation of the three methods (all less than 500MB) and the generally large memory capacity of modern servers, the additional memory occupation introduced by the monitoring does not have a significant performance impact on the HPC job in the vast majority of cases.

Coarse-grained collection does not involve the process of aggregating monitoring data into log data. Its monitoring results are consistent with the log results. However, due to the fact that the actual time spent on a single collection is slightly more than 30 s and the latency differences between tools, the timing of the data points gradually shifts during monitoring. Additionally, the startup time for Perf is significantly longer than for other tools, leading to deviations in the first data point. These time alignment issues result in a high missing data rate in the coarse-grained collection dataset. The unsynchronized fine-grained collection performs the monitoring process independently from the log generation process. The log data is generated in strict accordance according to the configured time interval, eliminating missing data. However, there is an obvious alignment problem with its monitoring data. This increases the likelihood of shift anomalies and negatively impacts the detection of subsequent anomalies and change points. Although our method experiences some loss in the amount

Table 11 Data results of different monitoring methods

Method	Tool	Rows (Mon)	Rows (Log)	NA% (Mon)	NA% (Log)	CPU%
Coarse-grained collection	Redfish	770	770	92.1%	92.1%	1.13%
	Procfs	767	767			
	Perf	758	758			
Unsynchronized fine-grained collection	Redfish	22,110	778	21.4%	0%	5.82%
	Procfs	21,476	778			
	Perf	19,792	778			
Synchronized fine-grained collection	Redfish	11,696	779	0%	0%	4.77%
	Procfs	11,696	779			
	Perf	11,696	779			

of monitoring data, the collected monitoring data and aggregated log data are fully aligned. Compared to unsynchronized fine-grained collection, there is only a difference in time synchronization, demonstrating the synchronization algorithm in our framework can effectively solve the alignment problem of multiple monitoring tools.

5.3 Anomaly detection experiment

This section evaluates the anomaly recognition capability of our framework using the dataset acquired by synchronized fine-grained collection from the previous section. We also compare our anomaly detection algorithm with several unsupervised time-series anomaly detection algorithms, including DeepSVDD [45], AnomalyTransformer [46], and DeepIsolationForest [47]. All algorithm implementations are derived from the original paper and run with default hyper-parameters.

The original dataset was smoothed every 15 data to generate training data for anomaly detection models. The trained models performed anomaly detection on the original monitoring data. To quantitatively assess the effectiveness of different algorithms, we excluded the anomalous data identified by different algorithms and used the remaining monitoring data to train power prediction models. Higher model accuracy indicates that the dataset contains less error information. R-squared was used to evaluate the model accuracy. Figure 5 shows the average results of the experiment repeated ten times, including the accuracy of the model trained with original test data, the accuracy of the model trained with test data after removing anomalies, and the number of anomalies identified by different algorithms.

In respect of modeling accuracy, our method performs best among all algorithms and can improve model accuracy by 14.52%. The DeepSVDD struggles to accurately identify the anomalies, resulting in decreased model accuracy. DeepIsolationForest has an accuracy improvement effect similar to that of our framework, but it identifies more than three times as many anomalies. This suggests that DeepIsolationForest not only removes outlier and shift anomalies but also eliminates a significant amount of normal data.

To present the results of different algorithms more intuitively, Fig. 6 demonstrates a segment of time-series data (length 500, downscaled to 1 dimension using PCA) and the anomalies identified by each algorithm (with the numbers of anomalies are shown in the legend). The figure highlights that AnomalyTransformer and DeepIsolationForest tend to recognize long segments of continuous anomalous data, resulting in a higher total number of anomalies across the entire test set. Specifically, DeepIsolationForest identified even 46% of the data in the figure as anomalous. The long segments of anomalous data identified by AnomalyTransformer were not concentrated in the period of the listed data, so the anomalous data accounted for only 11%. However, both outlier and offset anomalies are unlikely to occur for long periods. AnomalyTransformer and DeepIsolationForest happen to eliminate some anomalies in the process of removing a large amount of data. On the other hand, DeepSVDD's anomaly distribution is more plausible, but it performs poorly overall, making it difficult to accurately detect genuine anomalies.

From the distribution perspective, the anomalies identified by our algorithm exhibit characteristics of random triggers. The figure also shows the variation curve of the data after removing the anomalies detected by our algorithm. Compared to the raw data, the corrected curve is smoother at the points of abrupt changes, which are likely caused by outlier anomalies. In addition, many of the anomalies removed show little difference from the original data (which accounts for 41.7% of the total in the figure). This subset of anomalies is most likely due to data shifts.

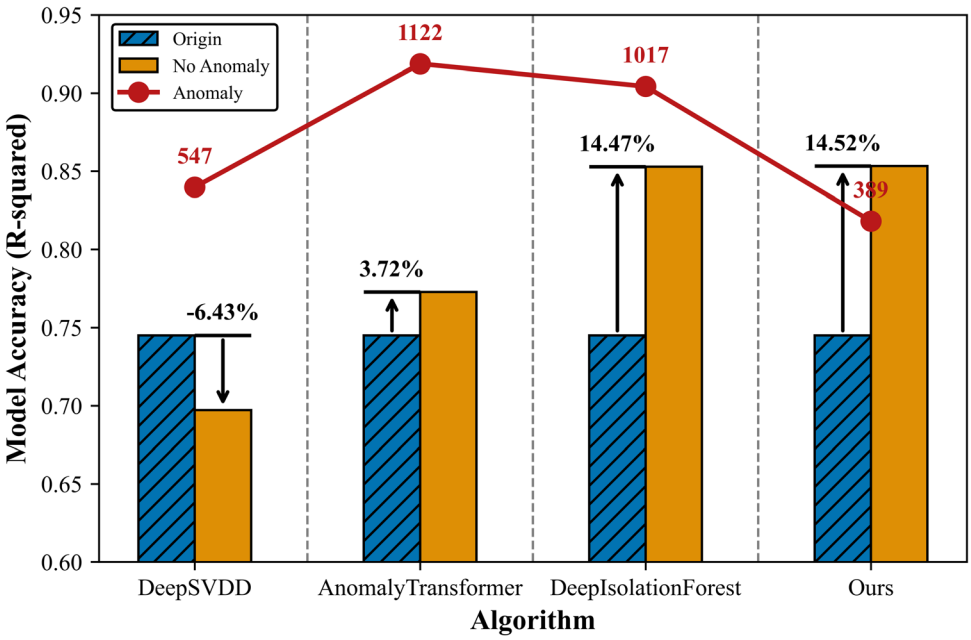


Fig. 5 Changes in model accuracy after removing anomalous data using different algorithms

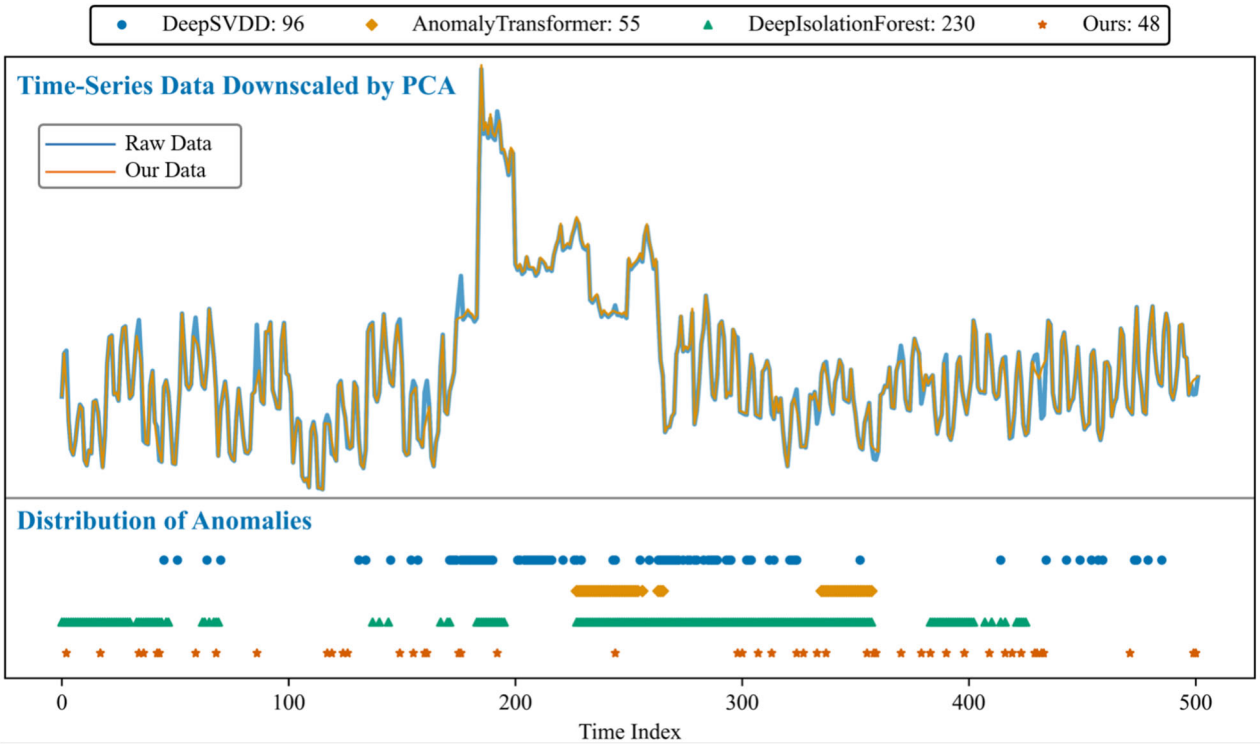


Fig. 6 Distribution of raw data and anomaly detection results

Since the other data behave similarly to those in this figure, we do not repeat the plotting. By considering both model accuracy and anomaly distribution, we can conclude that our anomaly detection algorithm effectively identifies a wide range of anomalies that may occur in the HPC environment.

5.4 Log aggregation experiment

This section evaluates the impact of the aggregation method proposed in this framework, comparing it with the full average aggregation method. Both algorithms keep the same window size for comparison. To visualize the difference between the two methods, Fig. 7 plots the change

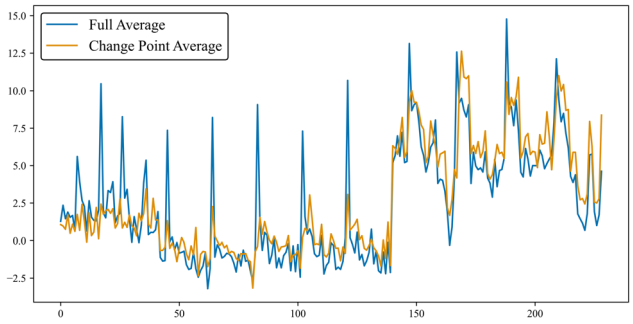


Fig. 7 Partial log data generated by different aggregation methods, downscaled to one-dimensional using PCA

curve when a sub-log sequence is downscaled to one dimension. The blue curve represents the log data generated by directly averaging the 30-second monitoring data (with outliers removed), while the other curve shows the log data obtained by averaging the longest sub-series of monitoring data based on detected change points.

It is intuitively evident that the log data generated by the full averaging method contains numerous spikes. Most of these spikes result from unrepresentative averaged data caused by workload state changes. Such log data points appear only briefly and fail to reasonably represent a new server status. In contrast, our method effectively mitigates this issue, producing relatively smooth log data even during workload state transitions. Since the other data behave similarly to those in this figure, we do not repeat the plotting.

To further quantify the volatility of the log data, we employed a sliding window to calculate the standard deviation (STD), the mean absolute deviation (MAD) and the difference between the maximum and minimum values (range) of the log data generated by the two methods. These metrics are commonly used to assess the volatility of data sequences. The experimental results, shown in Table 12, were evaluated for sliding window sizes of 10, 20, and 30, with the log data normalized for consistency. Across all window sizes and metrics, our aggregation method demonstrates significant advantages over direct averaging. On average, the STD, MAD and Range metrics are reduced by 7.77, 7.48 and 7.52%, respectively. Since workload state changes are infrequent, these improvements primarily reflect the overall effect. During periods of workload transitions, the benefits of our method are even more pronounced.

Table 12 Volatility comparison of log data obtained by different aggregation methods

Windows size	Method	STD	MAD	Range
10	All	0.086	0.063	0.270
	Ours	0.080	0.058	0.249
20	All	0.097	0.068	0.369
	Ours	0.088	0.063	0.337
30	All	0.099	0.070	0.410
	Ours	0.092	0.065	0.385

The above experiments demonstrate that the proposed aggregation method provides a more stable representation of server operational status, effectively reducing the impact of load fluctuations and state transitions on log data.

5.5 Example: Power capping optimization

In order to specifically demonstrate the impact of dataset quality, this experiment takes the power capping optimization scenario as an example. Considering that the probability of servers being fully loaded simultaneously are low, racks tend to overdeploy servers to increase power supply efficiency. When the total power of the servers exceeds the maximum power that the UPS can provide, the power capping technique is used to limit the maximum power consumption of some servers to ensure the electricity safety [48]. Due to the hysteresis of power collection, many researches use models to improve the response time of capping. [49, 50].

Data sets of varying quality will affect the accuracy of the power model and thus the validity of the power capping. The three data collection methods in Sect. 5.2 are used to obtain the power models. This experiment uses CPU frequency adjustment to control server power, and three strategies are use for implementing power capping algorithms, i.e., capping the maximum power servers (Power), capping the minimum utilization servers (Performance), and capping the highest energy efficiency servers (Efficiency). The power value of single server in the experimental environment ranges from 200W to 450W (the total power of the four compute nodes ranges from 800W to 1800W). The power capping will be triggered when the total power exceeds 1500W. The five samples of LAMMPS are used to construct workload flow. Each sample is submitted 20 times and randomly assigned the number of nodes and threads.

Table 13 shows the optimization results under different combinations of monitoring methods and power capping algorithms, including the accuracy of the power model built from the monitoring dataset, the average total power of compute nodes during job execution, the percentage of instances where power capping was triggered (with the value in parentheses representing the percentage of instances where total power exceeded 1500W without triggering capping), the average total power of compute nodes during capping (with the value in parentheses representing the average power when total power exceeded 1500W without capping), and the total time required to complete all jobs.

Table 13 Comparison under different combinations of monitoring methods and power capping algorithms

Capping	Monitoring	Accuracy	Server power	Capping power	Capping ratio	Runtime
Power	Coarse	71.62%	1501W	1598W(1576W)	21.3%(36.8%)	16,949 s
	Fine(unsync)	87.99%	1496W	1589W(1544W)	28.5%(30.6%)	16,488 s
	Fine(sync)	94.73%	1479W	1562W(1518W)	30.6%(15.1%)	17,928 s
Performance	Coarse	71.62%	1495W	1604W(1566W)	19%(34.4%)	18,444 s
	Fine(unsync)	87.99%	1501W	1586W(1551W)	26.8%(30.4%)	18,667 s
	Fine(sync)	94.73%	1483W	1560W(1516W)	34.7%(13%)	18,547 s
Efficiency	Coarse	71.62%	1503W	1614W(1582W)	22%(37.7%)	16,545 s
	Fine(unsync)	87.99%	1491W	1590W(1547W)	26%(25.8%)	17,228 s
	Fine(sync)	94.73%	1482W	1562W(1521W)	33.1%(13.9%)	16,839 s

It can be found that the dataset has a large impact on the power model accuracy, which further affects the power capping. When the cluster power is in the rising period, the high-precision model has a faster response speed, thus triggering capping at lower power with a higher triggering ratio, while the miss triggering is basically located near the power threshold. Therefore, the high-precision model has advantages in terms of average power and trigger ratio for both trigger and miss trigger capping. The above trends appear in all three power capping algorithms, proving that the significant effect of dataset quality on the power capping is not an exception.

The job runtime is affected by multiple factors such as load fluctuations, scheduling, noise. The impact of power models with different accuracy on the runtime is mostly no more than 5%, i.e., it is within the range of reasonable fluctuations. However, the runtime of the Power algorithm for the highest precision model is significantly increased. The possible reason is that Power has more frequent frequency limitation on highly loaded servers. The limitation of other algorithms occurs on low and medium load servers, which has less impact on the overall runtime. Secondly, the job runtime of Performance algorithm is overall larger than the other algorithms. The possible reason is that the Performance algorithm selects the server with the lowest load for capping. A single capping results in insignificant power reduction, and more frequency limitations are required to decrease the total power consumption to a safe value. The job runtime in Performance has increased but the capping ratio has not decreased significantly, which reflects the fact that the absolute number of capping is higher in Performance.

6 Discussion

The fusion of data from multiple sources is gradually becoming the basis for realizing more in-depth data analysis. Differences in the organization of multi-source data affect data quality to a certain extent. As the data quality problems exist in different datasets vary, it is difficult to integrate data quality and data analysis organically. Most data quality research is to filter or correct the data through post hoc analysis, while data analysis research is not clear what quality problems exist in the data set. As a result, scholars can only directly use the original data for analysis.

In the HPC system analysis scenario studied in this paper, we analyzed the correctness issues that exist in data fusion of multiple monitoring tools with the M100 dataset. Due to the lack of fine-grained data, the post hoc analysis was only able to identify some data quality issues without improving the quality of the dataset. For example, there is no way to address time alignment issues from a data processing perspective, and eliminating the anomalous data leads to loss of system states. Therefore, we prefer to optimize the various factors affecting dataset quality during the data generation process. Benefited from the existence of more operational privileges and available information, this approach can efficiently identify inaccurate data and correct them. From the experimental results, the accuracy of system profiling data can be significantly improved by some simple mechanisms that are much less complex compared to most of the post hoc analysis methods.

Although our research focuses on HPC system monitoring scenarios, the basic idea applies to multiple data fusion scenarios. We are concerned that existing monitoring tools do not have sufficient measures to realize a reasonable mechanism for multi-tool collaboration, which leads to the problem of correctness of data from multiple sources. The raw data organization designed by this framework can be combined with the efficient data transfer and storage techniques provided by existing mature monitoring tools. We appeal to dataset providers to concentrate on similar issues and to provide some insights on data quality beyond the raw data, guiding data analysis researchers to further enhance the accuracy of their studies.

The quality of datasets has a significant impact on further analysis and optimization. We perform an

experimental analysis on the effectiveness of the power capping on the basis of the accuracy of power models. To derive more comprehensive conclusions, further experiments are required to validate the influence of different qualities of the data set in more complex optimization scenarios. However, the primary focus of our study is the improvement of the original dataset. The analysis of complex optimization scenarios will be considered as a future research direction.

In the future, we will focus on further enhancing data correctness. In terms of time alignment, the proposed method requires incorporating specific logic into the data collection code to coordinate suspension and resumption via Linux signals. To further enhance applicability, we will investigate non-intrusive synchronization mechanisms for multiple monitoring tools, aiming to minimize the time discrepancy between sampling points with minimal modification to existing tools.

Regarding anomaly detection, since the identification of anomalies heavily relies on expert knowledge, we plan to leverage the strong inductive capabilities of large language models to reduce detection overhead and improve accuracy. Compared to obvious outliers, inconsistencies caused by data shift are more challenging to detect. We will explore the use of fuzzy computing, Bayesian inference, and related techniques to systematically study such subtle erroneous information.

In terms of data aggregation, different application scenarios have varying requirements, and no universally optimal aggregation algorithm exists. We will analyze the characteristics and applicable scenarios of commonly used aggregation methods, such as mean aggregation, change-point aggregation, and other aggregation techniques. By combining automated machine learning (AutoML) approaches, we aim to evaluate the performance of different aggregation strategies in specific scenarios and dynamically select the most suitable method.

For data analysis and optimization, this paper validates the proposed framework using two case studies: power consumption modeling and power capping. To further enhance the generality of our research, we will extend our investigations to additional scenarios, examining how data quality affects analysis and optimization performance. Example directions include predicting the temporal evolution of system loads and evaluating workload requirements for job scheduling, both of which heavily rely on models constructed from data-driven insights.

In summary, future work will focus on broader application and validation, with the goal of improving the applicability and generality of the study.

7 Conclusion

This paper proposes an HPC system monitoring framework designed to deliver accurate datasets for HPC system analysis research. Using the M100 dataset as a case study, we investigate common data correctness issues in HPC public datasets and identify factors influencing data accuracy. To address challenges in generating reliable log data, such as time alignment, anomaly detection, and log representation, we propose an organizational framework that integrates multiple monitoring tools. Experimental results demonstrate the framework's notable advantages over widely used methods in several key aspects.

Author Contributions All authors contribute to paper through either code, experiments or writing.

Funding This work is supported by Guangdong Provincial Natural Science Foundation Project (2025A1515010113), The Innovative Development Joint Fund of Natural Science foundation in Shandong Province (ZR2024LZH012), Guangzhou Development Zone Science and Technology Project (2023GH02) and the Major Key Project of PCL, China under Grant PCL2023A09.

Data availability Data available on request from the authors.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Miyazaki, T., Sato, I., Shimizu, N.: Bayesian optimization of hpc systems for energy efficiency. In: High Performance Computing: 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24–28, 2018, Proceedings 33, pp. 44–62 (2018). https://doi.org/10.1007/978-3-319-92040-5_3
2. Kodama, Y., Odajima, T., Arima, E., et al.: Evaluation of power management control on the supercomputer fugaku. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER), pp. 484–493 (2020). https://doi.org/10.1109/CLUSTER49012.2020.00069
3. Chalmers, N., Kurzak, J., McDougall, D., et al.: Optimizing high-performance linpack for exascale accelerated architectures. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2023). https://doi.org/10.1145/3581784.3607066
4. Versluis, L., Cetin, M., Greeven, C., et al.: A holistic analysis of datacenter operations: Resource usage, energy, and workload characterization–extended technical report. arXiv preprint arXiv:2107.11832 (2021) https://doi.org/10.48550/arXiv.2107.11832
5. Zhou, Z., Sun, J., Sun, G.: Automated hpc workload generation combining statistical modeling and autoregressive analysis. In: International Symposium on Benchmarking, Measuring and Optimization, pp. 153–170 (2023). https://doi.org/10.1007/978-981-97-0316-6_10

6. Enes, J., Expósito, R.R., Fuentes, J., Cacheiro, J.L., Touriño, J.: A pipeline architecture for feature-based unsupervised clustering using multivariate time series from hpc jobs. Inf. Fusion **93**, 1–20 (2023)
7. Zhu, C., Han, B., Li, G.: Pac: A monitoring framework for performance analysis of compression algorithms in spark. Future Gener. Comput. Syst. **157**, 237–249 (2024). https://doi.org/10.1016/j.future.2024.02.009
8. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with using the parallel workloads archive. J. Parallel Distrib. Comput. **74**(10), 2967–2982 (2014). https://doi.org/10.1016/j.jpdc.2014.06.013
9. Stefanov, K.S., Pawar, S., Ranjan, A., et al.: A review of supercomputer performance monitoring systems. Supercomput. Front. Innov. **8**(3), 62–81 (2021). https://doi.org/10.14529/jsfi210304
10. Stanisic, L., Reuter, K.: Mpcdf hpc performance monitoring system: Enabling insight via job-specific analysis. In: Euro-Par 2019: Parallel Processing Workshops: Euro-Par 2019 International Workshops, Göttingen, Germany, August 26–30, 2019, Revised Selected Papers 25, pp. 613–625 (2020). https://doi.org/10.1007/978-3-030-48340-1_47
11. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Comput. **30**(7), 817–840 (2004). https://doi.org/10.1016/j.parco.2004.04.001
12. Developers: collected: The System Statistics Collection Daemon. https://collectd.org/. Accessed: 2024-11-09 (2024)
13. Evans, T., Barth, W.L., Browne, J.C., 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
14. Carns, P., Latham, R., Ross, R., Iskra, K., Lang, S., Riley, K.: 24/7 characterization of petascale i/o workloads. In: 2009 IEEE International Conference on Cluster Computing and Workshops, pp. 1–10 (2009). https://doi.org/10.1109/CLUSTR.2009.5289150
15. Treibig, J., Hager, G., Wellein, G.: Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In: 2010 39th International Conference on Parallel Processing Workshops, pp. 207–216 (2010). https://doi.org/10.1109/ICPPW.2010.38
16. Borghesi, A., Di Santi, C., Molan, M., Ardebili, M.S., Mauri, A., Guarrasi, M., Galetti, D., Cestari, M., Barchi, F., Benini, L., et al.: M100 exadata: a data collection campaign on the Cineca's marconi100 tier-0 supercomputer. Sci. Data **10**(1), 288 (2023). https://doi.org/10.1038/s41597-023-02174-3
17. Farooq, E., Milano, M., Borghesi, A.: Harnessing federated learning for anomaly detection in supercomputer nodes. Future Gener. Comput. Syst. **161**, 673–685 (2024). https://doi.org/10.1016/j.future.2024.07.052
18. Guindani, B., Molan, M., Bartolini, A., Benini, L.: Exploring the utility of graph methods in hpc thermal modeling. In: Companion of the 15th ACM/SPEC International Conference on Performance Engineering, pp. 106–111 (2024). https://doi.org/10.1145/3629527.3652895
19. Carastan-Santos, D., Costa, G., Nardin, I.F., Poquet, M., Rzadca, K., Stolf, P., Trystram, D.: Scheduling with lightweight predictions in power-constrained hpc platforms (2024)
20. SURFsara: SURF Machine Metric Dataset. https://doi.org/10.5281/zenodo.4459519 (2021)
21. Laboratory, S.A.: Acme Trace. https://github.com/InternLM/AcmeTrace. Accessed: 2025-4-28 (2023)
22. Liu, Z., Xiao, G., Liu, H., Wei, H.: Multi-sensor measurement and data fusion. IEEE Instrum. Measur. Mag. **25**(1), 28–36 (2022). https://doi.org/10.1109/MIM.2022.9693406
23. Sampath, A., Tripti, C.: Synchronization in distributed systems. In: Advances in Computing and Information Technology: Proceedings of the Second International Conference on Advances in Computing and Information Technology (ACITY) July 13–15, 2012, Chennai, India-Volume 1, pp. 417–424 (2012). https://doi.org/10.1007/978-3-642-31513-8_43
24. Lee, S., Yuan, Z., Petrunin, I., Shin, H.: Impact analysis of time synchronization error in airborne target tracking using a heterogeneous sensor network. Drones **8**(5), 167 (2024). https://doi.org/10.3390/drones8050167
25. Ramar, C., Rubasoundar, K.: A survey on data aggregation techniques in wireless sensor networks. Int. J. Mobile Netw. Des. Innov. **6**(2), 81–91 (2015). https://doi.org/10.1504/IJMNDI.2015.072843
26. Krishnamurthi, R., Kumar, A., Gopinathan, D., Nayyar, A., Qureshi, B.: An overview of iot sensor data processing, fusion, and analysis techniques. Sensors **20**(21), 6076 (2020). https://doi.org/10.3390/s20216076
27. Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Commun. Surv. Tutor. **18**(1), 732–794 (2015). https://doi.org/10.1109/COMST.2015.2481183
28. Intel: Intel Xeon E3-1200 v3 Processor Family. https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-e3-1200v3-spec-update.pdf. Accessed: 2024-11-09 (2020)
29. Intel: Intel Stratix 10 SX Device Errata. https://cdrdv2.intel.com/v1/dl/getContent/666645?fileName=es-1056-683399-666645.pdf. Accessed: 2024-11-09 (2022)
30. Nguyen, A.L., Kamioka, E., Nguyen-Duc, T.: To verify the correctness of iot sensor data in real-time. In: 2019 25th Asia-Pacific Conference on Communications (APCC), pp. 479–484 (2019). https://doi.org/10.1109/APCC47188.2019.9026398
31. Shvets, P., Voevodin, V., Nikitenko, D.: Approach to workload analysis of large hpc centers. In: International Conference on Parallel Computational Technologies, pp. 16–30 (2020). https://doi.org/10.1007/978-3-030-55326-5_2
32. Patel, T., Byna, S., Lockwood, G.K., Wright, N.J., Carns, P., Ross, R., Tiwari, D.: Uncovering access, reuse, and sharing characteristics of {I/O-Intensive} files on {Large-Scale} production {HPC} systems. In: 18th USENIX Conference on File and Storage Technologies (FAST 20), pp. 91–101 (2020)
33. Dai, Y., Yan, Z., Cheng, J., Duan, X., Wang, G.: Analysis of multimodal data fusion from an information theory perspective. Inf. Sci. **623**, 164–183 (2023)
34. Roehl, T., Treibig, J., Hager, G., et al.: Overhead analysis of performance counter measurements. In: 2014 43rd International Conference on Parallel Processing Workshops, pp. 176–185 (2014). https://doi.org/10.1109/ICPPW.2014.34
35. Hunold, S., Ajanohoun, J.I., Vardas, I., et al.: An overhead analysis of mpi profiling and tracing tools. In: Proceedings of the 2nd Workshop on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn Strategy, pp. 5–13 (2022). https://doi.org/10.1145/3526063.3535353
36. Kumar, N., Childers, B.R., Soffa, M.L.: Low overhead program monitoring and profiling. ACM SIGSOFT Softw. Eng. Notes **31**(1), 28–34 (2005). https://doi.org/10.1145/1108768.1108801
37. Reghenzani, F., Massari, G., Fornaciari, W.: The real-time Linux kernel: A survey on preempt_rt. ACM Comput. Surv. (CSUR) **52**(1), 1–36 (2019). https://doi.org/10.1145/3297714
38. Wei, Y.: Research on real-time improvement technology of linux based on multi-core arm. In: 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), pp. 1061–1066 (2021). https://doi.org/10.1109/ICAICA52286.2021.9498165
39. Hannon, C., Yan, J., Liu, Y.-A., Jin, D.: A distributed virtual time system on embedded linux for evaluating cyber-physical systems. In: Proceedings of the 2019 ACM SIGSIM Conference on

Principles of Advanced Discrete Simulation, pp. 37–48 (2019). https://doi.org/10.1145/3316480.3322895
40. Aksar, B., Schwaller, B., Aaziz, O., et al.: E2ewatch: an end-to-end anomaly diagnosis framework for production hpc systems. In: Euro-Par 2021: Parallel Processing: 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1-3, 2021, Proceedings 27, pp. 70–85 (2021). https://doi.org/10.1007/978-3-030-85665-6_5
41. Molan, M., Borghesi, A., Cesarini, D., et al.: Ruad: Unsupervised anomaly detection in hpc systems. Future Gener. Comput. Syst. **141**, 542–554 (2023). https://doi.org/10.1016/j.future.2022.12.001
42. Xu, R., Miao, H., Wang, S., Yu, P.S., Wang, J.: Pefad: A parameter-efficient federated framework for time series anomaly detection (2024). arXiv preprint arXiv:2406.02318, https://doi.org/10.48550/arXiv.2406.02318
43. García-Gil, D., López, D., Argüelles-Martino, D., Carrasco, J., Aguilera-Martos, I., Luengo, J., Herrera, F.: Developing big data anomaly dynamic and static detection algorithms: AnomalyDSD spark package. Inf. Sci. **690**, 121587 (2025)
44. Truong, C., Oudre, L., Vayatis, N.: Selective review of offline change point detection methods. Signal Process. **167**, 107299 (2020). https://doi.org/10.1016/j.sigpro.2019.107299
45. Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S.A., Binder, A., Müller, E., Kloft, M.: Deep one-class classification. In: International Conference on Machine Learning, pp. 4393–4402 (2018)
46. Xu, J., Wu, H., Wang, J., Long, M.: Anomaly transformer: Time series anomaly detection with association discrepancy (2021). arXiv preprint arXiv:2110.02642
47. Xu, H., Pang, G., Wang, Y., Wang, Y.: Deep isolation forest for anomaly detection. IEEE Trans. Knowl. Data Eng. (2023). https://doi.org/10.1109/TKDE.2023.3270293
48. Ramesh, S., Perarnau, S., Bhalachandra, S., Malony, A.D., Beckman, P.: Understanding the impact of dynamic power capping on application progress. In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 793–804 (2019). https://doi.org/10.1109/IPDPS.2019.00088
49. Wu, S., Chen, Y., Wang, X., Jin, H., Liu, F., Chen, H., Yan, C.: Precise power capping for latency-sensitive applications in datacenter. IEEE Trans. Sustain. Comput. **6**(3), 469–480 (2018). https://doi.org/10.1109/TSUSC.2018.2881893
50. Savasci, M., Ali-Eldin, A., Eker, J., Robertsson, A., Shenoy, P.: Ddpc: Automated data-driven power-performance controller design on-the-fly for latency-sensitive web services. In: Proceedings of the ACM Web Conference 2023, pp. 3067–3076 (2023). https://doi.org/10.1145/3543507.3583437

Xiaoxuan Luo is a ph.D. candidate in the School of Computer Science and Engineering, South China University of Technology. His research interests mainly focus on performance evaluation, distributed computing and energy-efficiency optimization.

Weiwei Lin (Senior Member, IEEE) received his B.S. and M.S. degrees from Nanchang University in 2001 and 2004, respectively, and his Ph.D. in Computer Application from South China University of Technology in 2007. Currently, he is a professor in the School of Computer Science and Engineering at South China University of Technology. His research interests include distributed systems, cloud computing, and AI application technologies. He has published more than 150 papers in refereed journals and conference proceedings. He has been a reviewer for many international journals, including IEEE TPDS, TSC, TCC, TC, TCYB, etc. He is a distinguished member of CCF and a senior member of the IEEE.

Fan Chen is currently pursuing the M.S. degree in the School of Software Engineering, South China University of Technology. His research interests mainly focus on performance evaluation, distributed computing and time series analysis.

Haocheng Zhong received his bachelor's degree from the School of Computer Science and Technology, University of Science and Technology of China in 2022. He is currently pursuing a master's degree in the School of Computer Science and Engineering at South China University of Technology. His research areas include cloud computing, cluster computing, and cluster performance modeling.

Keqin Li is a SUNY Distinguished Professor of Computer Science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. His current research interests include cloud computing, energy-efficient computing and communication, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, machine learning, intelligent and soft computing. He has authored or coauthored over 850 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High Performance Computing. He has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing. He is an IEEE Fellow and an AAIA Fellow. He is also a Member of Academia Europaea (Academician of the Academy of Europe).