Characterizing HPC Performance Variation with Monitoring and Unsupervised Learning

Gence Ozer[1(✉)], Alessio Netti[1,2], Daniele Tafani[2], and Martin Schulz[1]

[1] Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
gence.ozer@tum.de, schulzm@in.tum.de
[2] Leibniz-Rechenzentrum, Boltzmannstr. 1, 85748 Garching, Germany
{alessio.netti,daniele.tafani}@lrz.de

Abstract. As HPC systems grow larger and more complex, characterizing the relationships between their different components and gaining insight on their behavior becomes difficult. In turn, this puts a burden on both system administrators and developers who aim at improving the efficiency and reliability of systems, algorithms and applications. Automated approaches capable of extracting a system's behavior, as well as identifying anomalies and outliers, are necessary more than ever.

In this work we discuss our exploratory study of Bayesian Gaussian mixture models, an unsupervised machine learning technique, to characterize the performance of an HPC system's components, as well as to identify anomalies, based on sensor data. We propose an algorithmic framework for this purpose, implement it within the DCDB monitoring and operational data analytics system, and present several case studies carried out using data from a production HPC system.

Keywords: HPC systems · Monitoring · Operational data analytics · Clustering · Anomaly detection

1 Introduction

As the demand for more capable *High-Performance Computing* (HPC) systems increases, supercomputing centers keep adding hardware and software resources to build significantly more complex and heterogeneous platforms: on top of their extreme power consumption [19], these systems also introduce new major challenges regarding hardware reliability [6] and performance variability [13], which in turn hinder the optimization of system operations. To address these issues, monitoring frameworks can be used to capture fine-grained sensor data (e.g., power or temperature) from all components in a production HPC system, allowing users and administrators alike to understand and characterize the system's behavior, as well as identify potential anomalies: processes of this kind are referred to as *Operational Data Analytics* (ODA) [4]. This knowledge subsequently can support tuning strategies to improve energy efficiency and system reliability, among other aspects.

H. Jagode et al. (Eds.): ISC High Performance 2020 Workshops, LNCS 12321, pp. 280–292, 2020.
https://doi.org/10.1007/978-3-030-59851-8_18

The majority of HPC centers still relies on the domain knowledge and expertise of system administrators, who manually analyze key monitoring metrics and log streams, to steer operations and thus enact ODA [5]. As systems become more complex, however, manual tuning becomes impractical and correlations between metrics become both more critical and at the same time difficult to spot: this can be addressed by partially automating the ODA processes, using data mining algorithms to construct a global overview of the performance of a system and to find outliers, allowing system administrators to focus on root cause analysis and mitigation actions. In the context of this paper, we focus on *performance variation* and *anomaly* detection: the first is a systematic approach to characterize the performance of an HPC system and extract behavioral patterns both in time and across system components. The latter can be seen as an extreme case of performance variation, where the behavior of a set of components severely deviates from the others due to abnormal events, such as faults or failures [11].

Related Work. Many approaches to performance variation and anomaly detection have been proposed, relying on either text log streams or monitoring sensors as sources of data. These approaches can be classified according to the techniques they employ, starting from supervised machine learning: Tuncer et al. proposed an approach to detect performance anomalies based on monitoring data, which is processed and fed to classifiers [18]. Similarly, Baseman et al. proposed a framework using density estimation and random forest classification to detect anomalies [2]. Wang et al. combined instead independent component analysis with Bayesian classifiers to spot anomalies in virtual machines [20].

In contrast, others employ unsupervised machine learning techniques: Dani et al. performed classification using the K-means algorithm to find abnormal log streams [8]. Similarly, Münz et al. applied K-means to network traffic data in order to detect anomalous activity [14]. Zhang et al. targeted the domain of cloud computing and applied DBSCAN clustering to fine-grained, thread-level resource usage metrics to detect performance anomalies [21]. Finally, Borghesi et al. developed a semi-supervised approach based on auto-encoders, which leverages the reconstruction error to detect anomalous states [3].

Less common is the use of traditional statistical analysis techniques: Gabel et al. developed a fault detection model for cloud services using statistical tests such as the Tukey test or the sign test [10]. Cohen et al. used tree-augmented Bayesian networks to achieve anomaly detection based on performance data [7]. Guan et al. [12], instead, used a most relevant principal components method to characterize faults based on the correlations between monitoring sensors. Among commercial solutions, *Datadog* utilizes median absolute deviation algorithms to identify anomalous states in the servers.[1] However, while there is an abundance of anomaly detection techniques, there is to our knowledge a lack of techniques combining the former with a systematic performance characterization of an HPC system's components. Moreover, the feasibility of current approaches in an online context and their associated details are not clear, yet.

[1] https://docs.datadoghq.com/monitors/monitor_types/outlier/.

Contributions. In this work we explore and evaluate the effectiveness of unsupervised machine learning for characterizing performance variation in HPC systems. In particular, we present an experimental approach using *Bayesian Gaussian mixture models* applied to HPC monitoring data, which allows us to extract statistical descriptions of the behavior of components at any level in an HPC system via Gaussian distributions. Based on this, we also propose an anomaly detection method that employs the *Mahalanobis distance*. We conduct an exploratory analysis on available monitoring data from the *CooLMUC-3* cluster operated by the *Leibniz Supercomputing Centre* (LRZ), and present the insights on early experiences with our approach. Monitoring is performed via the *Data Center Data Base* (DCDB) framework [15], and the online implementation of our approach is realized with its *Wintermute* ODA extension [16].

Organization. The paper is structured as follows. In Sect. 2 we present our exploratory analysis of the CooLMUC-3 monitoring data. In Sect. 3 we then describe our approach to performance variation characterization, and in Sect. 4 we present several case studies. Section 5 concludes the paper.

2 HPC Environment and Monitoring Data

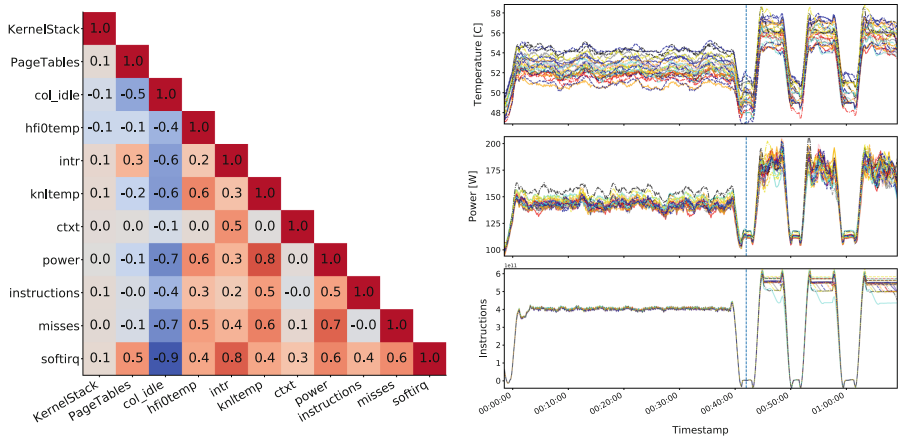
First, we introduce the CooLMUC-3 system on which we conduct our exploratory analysis and the associated monitoring metrics. In particular, we highlight the relationships between metrics as well as variation across components.

2.1 HPC Environment

We employ the CooLMUC-3 HPC system hosted at LRZ:[2] it is composed of 148 compute nodes, each equipped with an Intel Xeon Phi 7210 CPU with 64 physical cores each having 4 threads. Each node has 96 GB of RAM, 16 GB of high-bandwidth memory (HBM) and a dual-rail Intel OmniPath network interface. In addition, CooLMUC-3 uses warm water cooling for all of its components: this includes, for example, the network switches and allows all racks to be completely isolated from the environment, reducing heat dispersion to ambient air. We leverage the *Data Center Data Base* (DCDB) framework developed at LRZ [15] to perform monitoring on CooLMUC-3. DCDB operates continuously on this system, collecting fine-granularity production sensor data from both compute nodes and the system's infrastructure. The data is collected by plugin-based daemons called *Pushers* and is sent via the MQTT protocol to *Collect Agents*, which act as data brokers. The data is finally stored in an *Apache Cassandra* database, from which it can be queried.

Monitoring data is collected from a variety of sources: in compute nodes, we collect CPU performance counters (e.g., number of instructions) via a *Perfevents* plugin, paired with additional CPU activity information from the *stat* interface,

[2] https://doku.lrz.de/display/PUBLIC/CoolMUC-3.



(a) Correlation matrix. (b) Time series of the sensors.

Fig. 1. The correlation matrix associated to the sensors monitored in the CooLMUC-3 system, and an excerpt of the time series for the compute node power, CPU instructions and temperature sensors.

as well as memory usage information from *meminfo* and *vmstat* with a *ProcFS* plugin. Moreover, we collect network-related metrics and additional sensors (e.g., power consumption or temperature) via a *SysFS* plugin. We also collect out-of-band data for rack-level power consumption via a *REST* plugin, as well as a series of metrics related to the warm water cooling system (e.g., inlet water temperature) with an *SNMP* plugin. All sensors follow a numerical time series format and are sampled every 10 s.

2.2 Analysis of Monitoring Data

In order to characterize the DCDB metrics and their behavior under a uniform workload, we execute a series of proxy applications from the Coral-2 suite[3] on a set of 32 CooLMUC-3 nodes, with DCDB activated. Specifically, we use *LAMMPS*, *Kripke* and *Nekbone*, which were configured to use one MPI rank per node and 64 threads, as many as the physical CPU cores per node. In our experience on this system, this type of configuration leads to good performance and makes full use of available resources such as the HBM memory. As the applications have diverse performance profiles, we expect our analysis to be general, with minimal resulting bias. The results are shown in Fig. 1: for space reasons, we do not present the analysis in its full extent, but focus on the aspects that are most relevant in the context of this work.

Figure 1a shows the correlation matrix for a subset of DCDB metrics that exhibit interesting behaviors. Strong correlation patterns can be observed in this subset, with most metrics related to computational intensity such as CPU

[3] https://asc.llnl.gov/coral-2-benchmarks.

instructions or idle time (*col_idle*) having a direct impact on power consumption and temperature (*knltemp*). A similar behavior is observed for other CPU metrics, such as cache misses (*misses*), whereas OS-level metrics such as number of context switches (*ctxt*) and size of the kernel stack (*Kernelstack*) have weaker correlations. Due to their clear interactions, these metrics provide a robust base for statistical analysis and anomaly detection.

In Fig. 1b we show instead an excerpt of the time series for the power, CPU instructions and temperature sensors respectively, for each of the 32 nodes, while running LAMMPS and then Nekbone. The two runs are separated by a vertical line: on top of the applications' behavior, we observe a large variance in the metrics across nodes. While instructions show only light variance, mostly under Nekbone, power consumption exhibits a spread of up to 30W, and temperature of up to 3°. If expressed in a concise, clear manner, this information could be leveraged by runtime tuning frameworks, for example, to distribute power budgets across a set of nodes [9], proving that characterizing the performance of components in an HPC system in a systematic way is indeed necessary.

3 The Variation Detection Framework

Based on our experience described in Sect. 2, we propose an approach for capturing HPC performance variation in a generic fashion. Our variation detection framework is based on unsupervised machine learning applied to monitoring sensor data and, as shown in Fig. 2, comprises three steps: data preparation, clustering and outlier detection. The framework is designed to operate in online and offline scenarios, and can be easily integrated in any monitoring system.

Data Preparation. First, the sensor data collected by the underlying monitoring system must be prepared for the clustering stage. We fill in missing values in the time series of each sensor by means of linear interpolation, and we transform all monotonic sensors that come in the form of accumulators (e.g., instructions or energy) into their first-order derivative, by applying the delta operation to consecutive readings. This way, we are able to count the number of occurrences of a certain event in each time range rather than the total number of events since boot time. Finally, the data is smoothed over specified aggregation windows, depending on the desired type and granularity of the analysis, supplying the input that will be used for the clustering process.

Clustering with Lookback. After the data transformation operations are complete, clustering can be performed in N dimensions in order to characterize performance variation. The selected subset of sensors out of the full dataset determines the number of dimensions in the clustering space as well as the scope of the analysis: an analysis focused on CPU performance variation will use metrics such as the per-CPU instructions or cache misses, whereas an energy efficiency-oriented analysis will consider metrics such as the compute node-level power consumption. Each HPC component involved in the analysis (e.g., CPU cores,

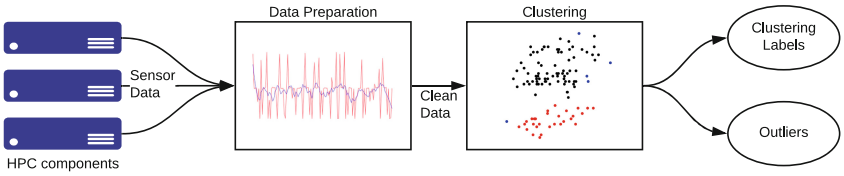


Fig. 2. Overview of the proposed variation detection framework.

compute nodes or racks) will represent a point in the clustering space whose coordinates are identified by the values of its selected metrics. In general, however, due to the limitations of most clustering algorithms, we expect our approach to be effective only when using a low number N (i.e., less than 10) of dimensions.

We use *Gaussian Mixture Models* (GMM) as a clustering approach: they try to explain the data by fitting multiple Gaussian distributions over it, typically using the *Expectation-Maximization* (EM) algorithm. Once the parameters of the individual distributions have been identified, each individual point in the data can be assigned to one of them, thus creating a series of clusters. This method generalizes well to different datasets, provides good results with minimal parameter tuning and produces compact statistical descriptions of the data clusters. Among existing GMM algorithms, we use *Bayesian Gaussian Mixture Models* (BGMM) [17]: compared to ordinary GMM algorithms, these are able to identify the optimal number of Gaussian distributions to use in the fitting process autonomously, further reducing model tuning and proving useful in online, continuous scenarios, where HPC systems exhibit highly diverse states. The maximum number of Gaussian distributions that are potentially generated by the BGMM algorithm can be specified as a parameter. We also experimented with algorithms such as DBSCAN, but we were not able to produce good results without tuning of parameters on a per-dataset basis.

We apply BGMM clustering to the points in the clustering space using a configurable *lookback* approach, considering not only the points corresponding to the most recent aggregation window for each HPC component, but also those corresponding to past ones. This approach has several benefits: it allows us to compare the characteristics of components over time, making in turn the clustering process more stable due to the increased number of points. Upon completion, the BGMM algorithm provides the mean vector and covariance matrix of each Gaussian distribution and assigns each point to one of them.

Outlier Detection. Since the BGMM algorithm does not label outliers automatically, we introduce a two-step outlier detection approach. First, due to the BGMM optimization process, small clusters grouping multiple outlier points can potentially be formed. As such, we label as outliers all points belonging to clusters that have a number of points lower than a configurable threshold. Second, some points might be assigned to certain clusters even though the associated probabilities are extremely low. To identify these points, we calculate their distance from the respective distributions using the *Mahalanobis distance*, which

is a scale-invariant metric that scales to multiple dimensions and that considers correlations in the data. Its equation is the following:

$$D_M(x) \quad = \quad \sqrt{(x-\mu)^T S^{-1}(x-\mu)} \tag{1}$$

In Eq. 1, μ and S respectively represent the mean vector and covariance matrix of the Gaussian distribution to which the point x is assigned by the BGMM algorithm. If the resulting distance is higher than a configurable threshold, the point is classified as an outlier. Since the Mahalanobis distance is proportional to the number of standard deviations that separate a point from a distribution's mean, this threshold is generic and does not need to be tuned ad-hoc for each experiment, but only in extreme scenarios.

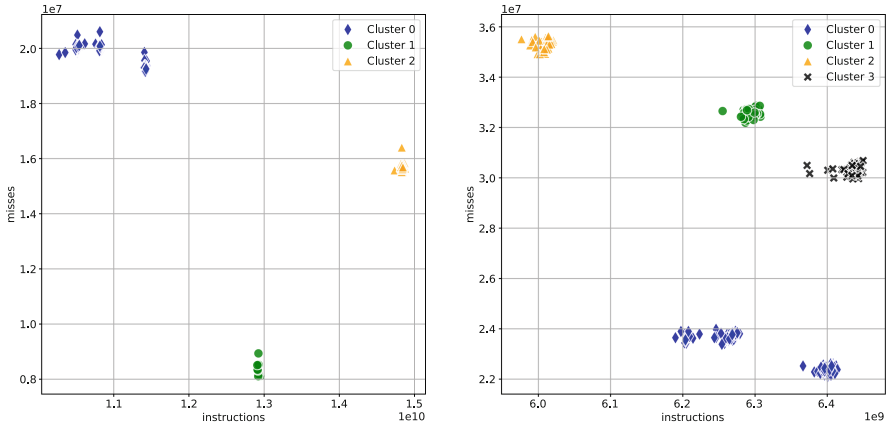
Implementation. We implemented our approach for both offline and online operation: the first offline implementation was made in Python, using the popular *scikit-learn* library. This implementation is suitable for experiments focusing on large historical datasets and requiring extensive data manipulation. Conversely, the second implementation is tailored for lightweight online operation in production HPC systems. This was carried out using the DCDB monitoring framework for HPC systems [15] and its Wintermute extension for online operational data analytics [16]. The resulting plugin is written in C++ and is based on the *OpenCV* GMM implementation: due to the lack of implementations in C++, we use a simple GMM model in the plugin in place of the more sophisticated BGMM one, relying on our offline implementation for hints on the ideal number of clusters to use in a certain scenario.

4 Case Studies

Here we present several case studies carried out using cluster-wide CooLMUC-3 data from Summer 2019: we target different aspects of the HPC system, starting from application phases at the CPU core level, up to power consumption and temperature at the compute node level and finally up to the rack level, analyzing the cooling system's efficiency. This way we evaluate the effectiveness of our framework with different granularities and data sources. For convenience, the experiments were carried out offline using archived data, but can be reproduced online using the DCDB Wintermute framework.

4.1 CPU Core-Level Analysis

We start with a short-term analysis of application behavior at the CPU core level. To this end, we execute the Kripke and LAMMPS proxy applications on 4 compute nodes of CooLMUC-3 and analyze the 1-min averages of the CPU instructions and cache misses (*misses*) metrics. Specifically, each point in the 2-D clustering space is a CPU core from a single node we selected out of the 4 available. We use the lookback feature to consider all data in the past 5 min and highlight the applications' behavior over time.

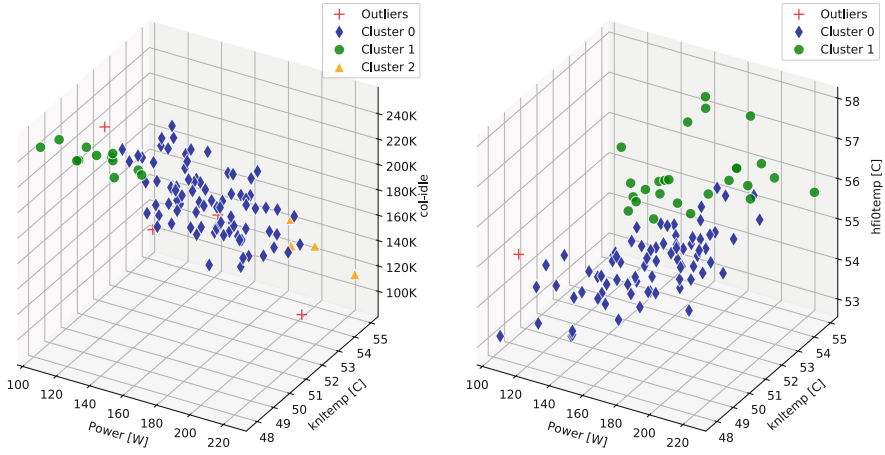


(a) Kripke. (b) LAMMPS.

Fig. 3. Results of clustering applied at the CPU core level to perform application phase detection in Kripke (a) and LAMMPS (b). Each point is a CPU core in a CooLMUC-3 compute node with its 1-min averages of the instructions and cache misses metrics.

The results are shown in the scatter plots in Fig. 3 for the Kripke and LAMMPS proxy applications respectively. In the scatter plots we show the outlier points as not belonging to any distribution, since this information is not reliable due to the very low probabilities involved. In Fig. 3a, Kripke exhibits a clear separation between its different phases on both axes, which are captured by 3 distinct Gaussian distributions. No outliers were identified, as variance across CPU cores appears to be limited. This is especially true for Clusters 1 and 2, which likely correspond to the compute-intensive phases of Kripke due to their higher instructions count. Cluster 0, on the other hand, may be associated with the application's initialization phase, due to its low instructions and high cache misses counts, and shows higher variance across CPU cores.

The behavior associated to LAMMPS, in Fig. 3b, is similar to what we observed with Kripke: Cluster 2 captures the initialization phase of the application, on the top left corner, while the other 3 clusters capture more compute-intensive phases. Interestingly, it can be observed that, while the point clouds associated to Clusters 1 and 3 were separated into two low-variance Gaussian distributions, Cluster 0 captures several distinct point clouds, resulting in a single distribution with higher variance. We attribute this behavior to the optimization process behind the EM algorithm. In general, it can be noted that LAMMPS exhibits both higher cache misses and lower instructions counts than Kripke, indicating stronger memory activity for this specific configuration. Since we only aim to characterize performance patterns, our approach's effectiveness at distinguishing the applications themselves is not clear: for this purpose, supervised learning models relying on a wide pool of metrics have been shown to be effective [1].



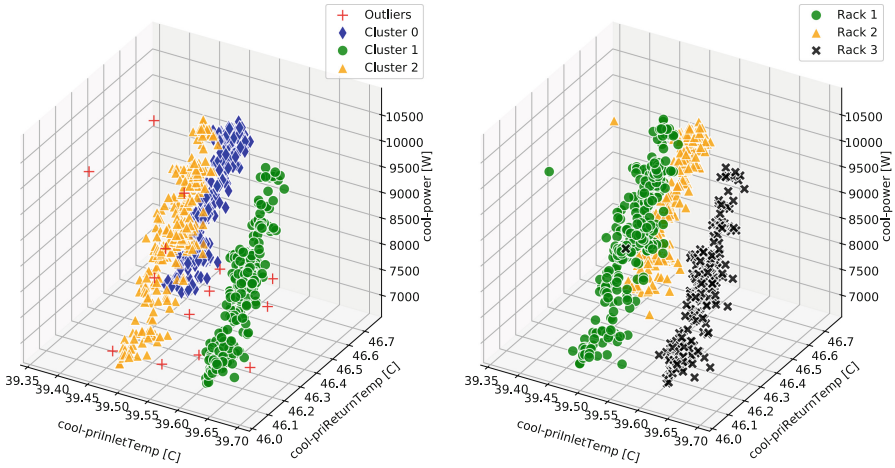
(a) CPU idle time. (b) Network interface temperature.

Fig. 4. Results of clustering applied at the compute node level. Each point represents a compute node in CooLMUC-3 with its 2-week average power consumption, temperature and CPU idle time (a) or network interface temperature (b).

4.2 Compute Node-Level Analysis

Here we perform a long-term compute node-level analysis, using data between September 2^{nd} and 15^{th} from CooLMUC-3. In particular, for each of its 148 compute nodes we use the 2-week averages of their power consumption, temperature (*knltemp*) and CPU idle time (*col_idle*), and the resulting points are clustered in a 3-D space with a maximum of 4 Gaussian distributions. The last of the three metrics is replaced by the network interface temperature (*hfi0temp*) in a second experiment. Although we have no knowledge of the running jobs in this time frame we expect the 2-week averages, given the maximum allowed job execution time of 48 h, to mitigate the bias of single applications and extract real node performance. We do not use lookback.

Results are shown in the scatter plots in Fig. 4a, which uses the CPU idle time as third metric, and Fig. 4b, which uses the network interface temperature. In both cases, a strong linear correlation can be observed among the metrics: as expected, a compute node with low CPU idle time consumes more power and has higher temperatures, due to the workload and communication of HPC applications. Despite the 2-week aggregation, a large spread in node behavior can still be observed: in Fig. 4a, compute nodes in Cluster 0 have higher CPU idle times with lower power and temperature values. Conversely, nodes in Cluster 2 exhibit heavier load, with up to 200W of average power consumption; this behavior is likely the symptom of a job scheduling policy that does not account for inter-node workload balance. Furthermore, a few nodes were classified as outliers: one of them shows a peculiar behavior, consuming 20% more power than other nodes with similar CPU idle time. We are currently monitoring this anomaly, in order



(a) Results of clustering. (b) Original per-rack data.

Fig. 5. Results of clustering applied at the rack level. Each point represents a rack in CooLMUC-3 with its hourly averages of the cooling system's warm water inlet temperature, return temperature and heat removed, at different points in time.

to identify its root cause. The same considerations apply for Fig. 4b: in this case, nodes belonging to Cluster 1 show higher network interface temperatures, which are likely caused by cooling inefficiency or manufacturing variability. A single node is classified as an outlier, with a higher network interface temperature than other nodes under similar load.

4.3 Rack-Level Analysis

For this last case study, we analyze several infrastructure metrics related to the 3 racks composing CooLMUC-3. Each rack contains roughly 50 nodes, as well as sensors for its section of the warm water cooling system. We consider two weeks of data, from June 28^{th} to July 11^{th}, for three sensors: the water's inlet temperature in the racks (*cool-priInletTemp*), its return temperature (*cool-priReturnTemp*) and finally the amount of heat removed from the racks (*cool-power*) quantified in Watts. We compute hourly averages for each metric, obtaining the 3-D points on which clustering is performed, with a maximum of 4 Gaussian distributions. Finally, we use the lookback feature to extend the number of points, so as to cover the entire 2 week range of available data.

The results are shown in the scatter plots in Fig. 5a, with the points labeled according to their Gaussian distributions, and in Fig. 5b according to the rack they belong to. Unlike in Sect. 4.2, only the return temperature and the heat removed metrics appear to be strongly correlated: this is expected, as the greater is the temperature difference between the inlet and return water, the greater is the amount of heat removed from the system. As the inlet temperature is

enforced externally, this metric does not show any correlation with the others and shows little variance across racks. Furthermore, it can be seen how the labeling between the two figures matches perfectly, with every rack separated and modeled by a distinct Gaussian distribution: the implication of this is that our approach can supply a statistical description of each rack's cooling performance, which simplifies performance characterization of the system, as well as anomaly detection. It can also be seen that Cluster 1 (i.e., Rack 2) shows a consistently higher return temperature, and that some outliers are present: three points in particular, which show deviation with respect to the inlet water temperature, come from the same time frame. This hints at the presence of an anomaly in the cooling system at that time, likely caused by environmental factors. These examples demonstrate the effectiveness of our framework in identifying the performance variation of HPC components: while most of these effects could be identified by human operators, the clear-cut statistical indicators we use simplify both data visualization at scale and proactive control by ODA algorithms.

5 Conclusions

In this paper we have presented a framework to characterize performance variation in the components of HPC systems: we employ Bayesian Gaussian mixture models applied to sensor monitoring data, so as to extract the behavioral groups associated to the components and their statistical description. Based on this, we proposed an anomaly identification mechanism that uses the Mahalanobis distance of the single clustering points to the fitted Gaussian distributions and also provided the online implementation of our approach in the DCDB Wintermute framework. We then presented the early findings of an exploratory analysis using our approach on production monitoring data from the CooLMUC-3 HPC system at LRZ, discussing several case studies carried out at different granularity levels effectively: our approach can capture different behaviors both in time and across different components, as well as flag suspicious behaviors as anomalies.

As future work, we plan to validate our approach in a quantitative way, identifying use cases with clear accuracy metrics. Moreover, we plan to further test our approach in combination with dimensionality reduction techniques, in order to enrich the information encoded within the clustering space, as well as devise techniques to identify relevant metrics for clustering automatically.

References

1. Ates, E., et al.: Taxonomist: application detection through rich monitoring data. In: Aldinucci, M., Padovani, L., Torquati, M. (eds.) Euro-Par 2018. LNCS, vol. 11014, pp. 92–105. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96983-1_7
2. Baseman, E., Blanchard, S., DeBardeleben, N., Bonnie, A., et al.: Interpretable anomaly detection for monitoring of high performance computing systems. In: Proceedings of the ACM SIGKDD 2016 Workshops (2016)
3. Borghesi, A., Libri, A., Benini, L., Bartolini, A.: Online anomaly detection in HPC systems. In: Proceedings of AICAS 2019, pp. 229–233. IEEE (2019)

4. Bourassa, N., Johnson, W., Broughton, J., Carter, D.M., et al.: Operational data analytics: optimizing the national energy research scientific computing center cooling systems. In: Proceedings of the ICPP 2019 Workshops, pp. 5:1–5:7. ACM (2019)
5. Bourassa, N., Ott, M.: EEHPCWG operational data analytics survey (2019). https://eehpcwg.llnl.gov/assets/sc19_11_425_525_operational_data_analytics_ott_bourassa.pdf
6. Cappello, F., Geist, A., Gropp, W., Kale, S., et al.: Toward exascale resilience: 2014 update. Supercomput. Front. Innovations **1**(1), 5–28 (2014)
7. Cohen, I., Chase, J.S., Goldszmidt, M., Kelly, T., Symons, J.: Correlating instrumentation data to system states: a building block for automated diagnosis and control. In: OSDI, vol. 4, p. 16 (2004)
8. Dani, M.C., Doreau, H., Alt, S.: K-means application for anomaly detection and log classification in HPC. In: Benferhat, S., Tabia, K., Ali, M. (eds.) IEA/AIE 2017. LNCS (LNAI), vol. 10351, pp. 201–210. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60045-1_23
9. Eastep, J., et al.: Global extensible open power manager: a vehicle for HPC community collaboration on co-designed energy management solutions. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D. (eds.) ISC 2017. LNCS, vol. 10266, pp. 394–412. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58667-0_21
10. Gabel, M., Gilad-Bachrach, R., Bjorner, N., Schuster, A.: Latent fault detection in cloud services. Microsoft Research, Technical report MSR-TR-2011-83 (2011)
11. Gainaru, A., Cappello, F.: Errors and faults. In: Herault, T., Robert, Y. (eds.) Fault-Tolerance Techniques for High-Performance Computing. CCN, pp. 89–144. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20943-2_2
12. Guan, Q., Fu, S.: Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In: Proceedings of SRDS 2013, pp. 205–214. IEEE (2013)
13. Inadomi, Y., Patki, T., Inoue, K., Aoyagi, M., et al.: Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In: Proceedings of SC 2015, pp. 1–12. IEEE (2015)
14. Münz, G., Li, S., Carle, G.: Traffic anomaly detection using k-means clustering. In: Proceedings of the GI/ITG Workshop MMBnet, pp. 13–14 (2007)
15. Netti, A., Mueller, M., Auweter, A., Guillen, C., et al.: From facility to application sensor data: modular, continuous and holistic monitoring with DCDB. In: Proceedings of SC 2019. ACM (2019)
16. Netti, A., Mueller, M., Guillen, C., Ott, M., et al.: DCDB Wintermute: enabling online and holistic operational data analytics on HPC systems. In: Proceedings of HPDC 2020. ACM (2020)
17. Roberts, S.J., Husmeier, D., Rezek, I., Penny, W.: Bayesian approaches to Gaussian mixture modeling. IEEE Trans. Pattern Anal. Mach. Intell. **20**(11), 1133–1142 (1998)
18. Tuncer, O., Ates, E., Zhang, Y., Turk, A., et al.: Online diagnosis of performance variation in HPC systems using machine learning. IEEE Trans. Parallel Distrib. Syst. **30**, 883–896 (2018)
19. Villa, O., Johnson, D.R., Oconnor, M., Bolotin, E., et al.: Scaling the power wall: a path to exascale. In: Proceedings of SC 2014, pp. 830–841. IEEE (2014)

20. Wang, G., Yang, J., Li, R.: An anomaly detection framework based on ICA and Bayesian classification for IaaS platforms. KSII Trans. Internet Inf. Syst. (TIIS) **10**(8), 3865–3883 (2016)
21. Zhang, X., Meng, F., Chen, P., Xu, J.: TaskInsight: a fine-grained performance anomaly detection and problem locating system. In: Proceedings of CLOUD 2016, pp. 917–920. IEEE (2016)