Analysing Supercomputer Nodes Behaviour with the Latent Representation of Deep Learning Models

Analysing Supercomputer Nodes Behaviour with the Latent Representation of Deep Learning Models

Martin Molan[1 [0000−0002−6805−2232]], Andrea Borghesi[1 [0000−0002−2298−2944]], Luca Benini[1,2 [0000−0001−8068−3806]], and Andrea Bartolini[1 [0000−0002−1148−2450]]

[1] DISI and DEI Department, University of Bologna, Bologna, Italy
{martin.molan2, andrea.borghesi3, luca.benini, a.bartolini@unibo.it}@unibo.it
[2] Institut für Integrierte Systeme, ETH, Zürich, Switzerland

Abstract. Anomaly detection systems are vital in ensuring the availability of modern High-Performance Computing (HPC) systems, where many components can fail or behave wrongly. Building a data-driven representation of the computing nodes can help with predictive maintenance and facility management. Luckily, most of the current supercomputers are endowed with monitoring frameworks that can build such representations in conjunction with Deep Learning (DL) models. In this work, we propose a novel semi-supervised DL approach based on autoencoder networks and clustering algorithms (applied to the latent representation) to build a digital twin of the computing nodes of the system. The DL model projects the node features into a lower-dimensional space. Then, clustering is applied to capture and reveal underlying, non-trivial correlations between the features.
The extracted information provides valuable insights for system administrators and managers, such as anomaly detection and node classification based on their behaviour and operative conditions. We validated the approach on 240 nodes from the Marconi 100 system, a Tier-0 supercomputer located in CINECA (Italy), considering a 10-month period.

Keywords: supercomputer monitoring · deep learning · unsupervised learning · autoencoders · predictive maintenance.

1 Introduction

High Performance Computing systems have been steadily rising in size and complexity in the last years, as revealed by the exponential increase of the worldwide supercomputer installation[3]. HPC systems are typically composed by replicating a large number of components, usually, in the order of thousands of computing nodes, each of them constituted of a collection of smaller functional parts, such as CPUs, RAM, interconnections, storage, etc. Even if similar by design, each

[3] https://www.top500.org/

computing node is affected by manufacturing variability and variations in the operating conditions. The sheer size and complexity of supercomputers create huge challenges in terms of optimal management of the IT components and their significant energy footprint[1]. The race towards Exascale[4] continues to make these challenges ever more pressing[3–5].

Overall, it is a daunting task for system administrators and facility managers to optimize supercomputer performance and power consumption, identify anomalous behaviors faulty situations, and guarantee systems operate in optimal conditions. The scale of the problem motivates the development of automated procedures for anomaly detection and faulty node identification in current supercomputers and this need will become even more pressing for future Exascale systems[6]. The fact that most of today's HPC computing systems are endowed with monitoring infrastructures[7] that gather data from software (SW) and hardware (HW) components can be of great help toward the development of data-driven automated approaches. Historically, system management was performed through hand-crafted scripts and direct intervention of system administrators; most of the data is stored in log files, and anomalies are investigated a posteriori to find the source of reported problems (e.g., when many users recognize the failure and report it to administrators). At the finer granularity, each core of the processing element is equipped with performance counters which can monitor several micro-architectural events (i.e., cache misses, stalls, throughput) and physical means (i.e., temperature, power consumption, and clock frequency). Processing units as well as the motherboard, the power distribution units, the onboard voltage regulators, the PCIe devices, and the fans are equipped with hardware (HW) sensors and counters. Similarly, software components can provide useful information as well, ranging from the details about jobs submitted by users (e.g., information gathered by job dispatchers such as SLURM[8] or PBS[9]) to software tools performing health-check of various subsystems[10] and I/O monitoring[11].

As the amount of data is overwhelming for human operators, automated processes could be highly beneficial in improving the data center usage to ease the burden of human operators and lower the response time to failures. In this context, Artificial Intelligence (AI) can provide significant benefits, as it allows to exploit the available big data effectively and to create decision support tools for HPC system administrators and facility managers[12, 13]. In the past, many works from the literature and the practice demonstrated the possibility to extract useful information using data collected from HPC computing nodes and employing supervised Deep Learning (DL) models[14–16] and semi-supervised ones[17–19]. These methods have been applied to detect nodes' availability, defined as operation without anomalies. Availability and the corresponding error rate (1 minus availability rate) is a key metric of the node's performance, and a target for optimization of the HPC system operation [20]. Due to its importance, we focus on the availability rate in the experimental part of this paper.

[4] The supercomputer peak performance is expected to reach the ExaFlops (10^18) scale in 2023[2].

Borghesi et al. in [18] show that semi-supervised anomaly detection models trained on individual nodes data outperform a single model trained on multi-node data. This suggests that the semi-supervised model can learn differences between nodes even if the nodes share the same design and composition. Theoretically, the learned model encapsulates the node's characteristics, however to the best of our knowledge, no one has ever evaluated the feasibility of using the disparities between trained DL models to evaluate the differences between the behavior of the corresponding nodes. In this work, we answer this question by introducing a novel approach that focuses on the latent representation of the trained DL models (in particular on the coefficients, the weights of the latent layer); the approach can identify clusters that deviate from the overall (node) population's average availability, relying on the DL model parameters.

We focused on a Tier0 supercomputer composed of 985 nodes for which we trained a series of per-node semi-supervised DL models based on autoencoders (AE), as proposed by authors in [19], the state-of-the-art for semi-supervised anomaly and fault detection in HPC systems. We focus on semi-supervised methods as the availability of labels cannot be taken for granted in a supercomputer due to the non-negligible cost of annotating the vast wealth of monitored data. We explored different approaches to extract features from the weights and biases of the latent layer of the AE model. The key idea is to apply a geometric transformation to the weight matrix underlying the latent layer of the trained AEs; we opted to explore a variety of transformations; namely, we compute: (1) the vector of singular values, (2) the singular vector corresponding to the largest singular value, (3) the map of the representative vector (with and without bias), (4) the weights matrix similarity in L1, L2, and absolute L2 norm, (5) the affine (augmented matrix) similarity in L1, L2, and absolute L2 norm. The empirical evaluation demonstrates that the vector of singular values identifies interesting clusters among the different methods to extract salient features from the latent representation.

We propose to use the deviation from population average availability to evaluate the goodness of the clustering results. The vector of singular values, extracted from the weights matrix of the latent layer of the trained autoencoder, identifies two clusters with average overall availability lower than 89% (compared to 96% population average). The proposed method's ability to identify these clusters is significant as the autoencoders have no access to the availability label during training.

2 Related work

Since anomalies in HPC systems are rare events, the problem of anomaly detection cannot be treated as a classical supervised learning problem [17, 21]; the majority of works that treat it in a fully supervised fashion have been tested using synthetic[14, 22] or injected anomalies[15]. Instead of learning the properties of both relevant classes, the standard approach is to learn just the properties of the system's normal operation - anything deviating from this normal operation

is then recognized as an anomaly. Machine learning models are trained only on normal data to learn the characteristics of the normal operation. This training of ML models on normal data is called *semi-supervised* training [18].

The state-of-the-art for anomaly detection on the HPC system is to train a particular class of neural networks – called autoencoders – in a semi-supervised way [19]. Autoencoders are a specific type of neural networks that are trained to reproduce an input signal while simultaneously learning the most efficient latent representation of the data [23]. The latent representation of the data has a lower dimension than the original data; this lower dimension of the latent layer naturally leads to the idea of using autoencoders as pre-processing step before applying clustering techniques [24–26], as most of the clustering algorithms have worse performance in high-dimensional spaces[27]. The autoencoders are first trained on the whole dataset when using autoencoders as a dimension-reduction step before clustering. Then the dataset is projected (by the encoder part of the network) into a lower-dimensional latent layer [24].

Current approaches that combine clustering and autoencoder neural networks use a single trained autoencoder to encode each instance into a latent space. The state-of-the-art for HPC anomaly detection, however, is to train multiple models (a different model for each node in the system) [19]. The fact that the models trained on individual nodes outperform the model trained on combined data of all nodes [19, 17, 18] suggests that there are significant differences between the behavior of the compute nodes and, consequently, the corresponding trained models. Thus, this paper's contribution is to explore the possibility of leveraging the fact that we are training multiple AE models to explore the relationship between the nodes themselves. Specifically, we explore the possibility to extract features from the trained neural networks to perform the clustering of the *whole operation history* of the compute nodes.

3 Methodology

In this section we present the architecture of the proposed approach. We start by providing the probabilistic perspective underlying the foundations of our approach in Sec. 3.1. We then describe in more detail the general architecture (Sec. 3.2) and the proved the more detailed description of the method in Sec. 3.3 and Sec. 3.4.

3.1 Probabilistic Background

The idea of extracting information and comparing trained neural networks extends the standard methodology of statistical modeling where two (or more) populations (or generally a collection of instances) are compared by contrasting parameters of fitted distributions. Comparing the parameters of fitted functions is the key idea underlying the proposed approach. Let us consider as an example the common statistical problem of comparing two populations of individuals - specifically, we want to compare a specific random variable X in two distinct

populations (e.g., height in two different countries). The first point of comparison in such cases is to calculate *empirical mean* $\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ and *empirical variance* $\frac{1}{n-1}\sum_{i=1}^{N}(x_i - \overline{x})^2$. Two populations can be compared by looking at the empirical mean and variance of the random variables of interest (observed variables present inside each population).

The mathematical foundation of comparing mean and variance between two populations is directly in line with the idea of this paper. If we are observing two large populations, we know (from the central limit theorem [28]) that the sum of the variables will tend towards a Gaussian distribution. Two parameters determine Gaussian distribution: expected value μ and variance σ^2 [28]; to fit the Gaussian distribution to the data (population), we thus have to estimate these two parameters. If we fit the distribution via the Maximum Likelihood Estimation (MLE) method, [29], we see that the best estimator for the expected value is *empirical mean* and for variance, the best estimator is *empirical variance*. From the probability theory, we know that the difference of two random Gaussian variables is Gaussian variable with mean that is the difference of means and variance that is the sum of variances [28]. Comparing population mean and population variance is thus actually equivalent to comparing the *Gaussian distributions* fitted to the data.

Another perspective from which to examine the problem of comparing populations is that we fit a *function* to the data (this function being the Gaussian distribution). For some problems - like High Performance Computer (HPC) system monitoring - autoencoders (type of neural networks) achieve state-of-the-art results [19, 30]. As autoencoders are the class of functions that best describe this specific class of problems (behavior of compute node in an HPC system), we examine if we can compare the *compute nodes* by comparing the parameters of the fitted autoencoders.

3.2 General overview of the approach

Figure 1 reports the block diagram of the proposed methodology. We can identify the following steps:

1. On each node, a separate autoencoder model is trained. Semi-supervised training of per-node autoencoder models is adopted from the state-of-the-art paper [19].
2. After models are trained on each node, features are extracted (as described in Section 3.4) from the deep learning models.
3. Based on these extracted features, the similarity between nodes is calculated. Calculation of similarity can be done as the autoencoder projects the input features into a *latent* representation where only the most salient correlations between the input variables are preserved. The similarity measure is calculated by comparing the representation maps - specifically, the parameters of the latent layer.
4. This similarity measure is then used in hierarchical clustering.

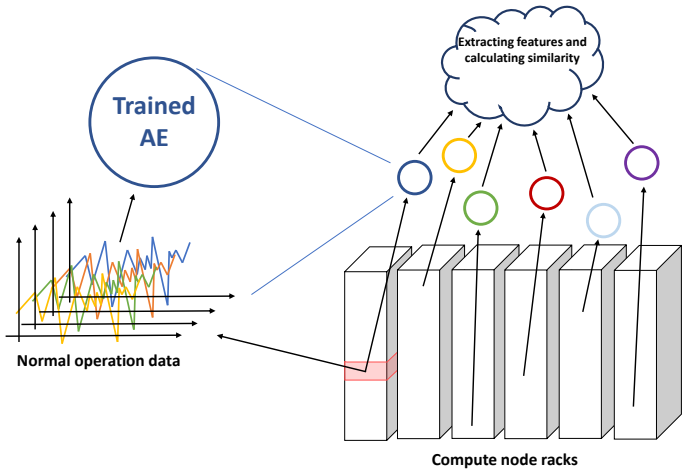

Fig. 1: Data flow schema. On each of the nodes (red in the picture), organized into racks, we train a separate autoencoder model (circles). From these trained models we extract features that are then used in the clustering of nodes.

3.3 Autoencoder models

Dense autoencoders are a type of deep neural network, which can be characterized by different topology; those used in this work have a distinct hourglass shape, a choice motivated by the results obtained by previous works in the state-of-the-art[5]. The most relevant information of the network is encoded in the latent layer. In this particular type of autoencoder, the latent layer is the layer in the middle of the network and contains the fewest neurons. It is preceded by the encoder and succeeded by the decoder, each composed of one or multiple layers. The encoder and decoder layers used in this work have a symmetrical architecture, which, generally speaking, is not strictly required. The fundamental role of the network is to efficiently encode the information from the input in a compressed representation in the latent layer. Training of the autoencoder is driven by the reproduction error produced by the decoder; reproduction error, which is the difference between the real input and the reconstructed signal, is minimized during training. The architecture of the network used in this work is presented in Figure 2. It is adapted from the work by Borghesi et al. [19] where it has been shown to produce state-of-the-art results in detecting anomalies on an HPC system.

The set of autoencoders - as in original work [19] - are individually trained on each node in a semi-supervised setting. Semi-supervised training means that the

[5] They are also referred to as *contractive autoencoders*

data for training is filtered of all anomalies and that only the normal instances are used in training the model.

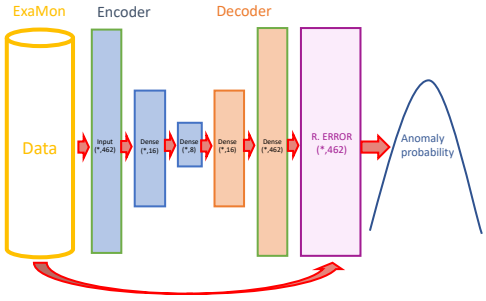

Fig. 2: Architecture of the state-of-the-art model, proposed by [19]. In this paper, relevant information is extracted from the latent layer Dense (*,8). Data is collected for the ExaMon monitoring system [31].

3.4 Feature extraction

Due to the architecture of the neural network used in this work - as discussed in Section 3.3 - we extract the relevant features from each node (each one with its data set); these features are embedded in the weights of the latent layer. The latent layer is described by the weights matrix W and the bias vector $\vec{b}$. The activation of the latent layer is given by $\vec{a}' = f(W\vec{a} + \vec{b})$ where f if a nonlinear activation function. In the next subsections, we will describe different encoding approaches of the latent layer information, which will then be used to extract features.

Singular value decomposition Singular value decomposition represents matrix M as $M = USV^*$ where S is a diagonal matrix containing singular values [32]. In this work, we used singular value decomposition on W, and we extracted the *vector* of singular values (abbreviated to singular values in the future) and a singular *vector* corresponding to the largest singular value (abbreviated singular vector).

Representative vector A vector of ones $\vec{1}$ is used as a representative vector as it corresponds to the activation of all neurons in a latent layer. It can serve as a proxy for the transformation of the (linear part) of the latent layer. For each node, we have thus calculated the product of $W\vec{1}$ (abbreviated vector of ones) and $W\vec{1} + \vec{b}$ (abbreviated vector of ones plus bias).

Matrix measures In this work, we leveraged the L1 and L2 norms induced in the matrix space (induced by p norms for vectors) [33]. Based on these norms we propose two ways to calculate distance between two matrices: *distance* $= ||A - B||_p$ and absolute distance *abs_distance* $= |||A - B||_p|$ where p is 1 or 2. Since the $L1$ measure is already symmetric, we do not separate a case with absolute distance. We introduce the absolute value as we want our distance measure to be symmetric.

We calculate the distance between nodes as a distance between the weights matrices of autoencoders trained on them. Additionally, since the linear part of the neural network is an affine transform, we introduce an augmented matrix A:

$$A = \left(\begin{array}{c|c} W & \vec{b} \\ \hline 0...0 & 1 \end{array}\right).$$

This matrix A captures the affine transform since $\vec{a}' = W\vec{a} + \vec{b}$ is equivalent to

$$\begin{pmatrix} \vec{a}' \\ 1 \end{pmatrix} = A \begin{pmatrix} \vec{a} \\ 1 \end{pmatrix}.$$

Another way to calculate the distance between nodes is to calculate the distance between an affine transform that is determined by the (affine $W\vec{a} + \vec{b}$) part of the latent layer of the corresponding autoencoder.

3.5 Clustering

The calculated distance between clusters is an input for clustering. In this work, we use agglomerative hierarchical clustering: each instance (in our case node) starts as its cluster. At every step of the iteration, the two closest clusters are connected. The connection between clusters is the closest distance between two instances in corresponding clusters. The combining of clusters is repeated until we reach the predetermined number of clusters.

3.6 Evaluating clustering

There are several possible measures to evaluate the goodness of the clustering (e.g., Silhouette score) [34]. These scores, however, are not applicable in the scenario explored by this work. We evaluate different possible feature extraction methods from the trained autoencoders; these different feature extraction approaches produce different feature spaces. Thus we cannot compare the clustering score (like Silhouette score) *between different spaces*. For this reason, we evaluate the relevance of our clustering approaches by evaluating how "interesting" the created clusters are.

The interest of clusters is reflected by how well they separate a specific variable. Since clustering is an unsupervised method, it is reasonable to assume that not all clusters will separate the same variable (such clustering would produce distinctly *uninteresting* clusters). However, we expect that there would be at

least one cluster where the distribution of the target variable would be significantly different than it is in the whole dataset. In this work, the target variable is system availability. In other words, clusters will separate computing nodes based on the autoencoder model's latent layer encoding in groups having similar availability, thus similar failure rate. We stress that from a practical point of view, this means that an autoencoder model for each node is trained only on "normal" operation samples and contains the information on the likelihood of the node to be available (or not to fail). Clusters of nodes sharing the same failure's likelihood can be used to rationalize the maintenance procedure.

In the whole dataset, the system is available 0.96179% of the time. The most interesting cluster is thus the one where the average availability of a cluster will be as far away from this population average. The best clustering method is the one producing the most interesting cluster.

3.7 Random sampling baseline

The relevance of the produced clusters determines the relevance of feature extraction and, consequently, of clustering approaches. Specifically, we observe how well the clusters separate a target variable (in the case of this work, the node's availability). To claim the relevance of the clustering approaches, we compare them to random sampling. We compare how well the target variable is separated by random sampling to how well clustering methods separate it. We are particularly interested in clustering methods that produce clusters and separations that do not (are very unlikely to occur) in random separation.

This paper implemented random clustering by producing a random matrix (of the same size and range as extracted features) that is then passed to clustering algorithms. The produced clusters are thus equivalent to random sampling without replacement. The generation of random clusters is repeated several (in this work 10) times. For each cluster, the distribution of the target variable is calculated; this distribution is then compared to distributions given by clustering methods. In the results (section 4), the range of randomly generated distributions is presented as a box with whiskers plot. Distributions outside the range of random distributions represent interesting patterns uncovered by the clustering method.

4 Results

This section presents the results of the experimental analysis conducted on a tier-0 supercomputer, Marconi100, hosted at CINECA, the largest Italian computing center. The results were conducted on a statistically significant fraction of the supercomputing nodes (more than two hundred) and cover a 10-months time span of production activity of the system.

4.1 Experimental setting

As explained in the methodology section 3, an individual model was trained on each of the 241 randomly selected nodes of Marconi100. Models were trained semi-supervised, meaning that only normal operation data was used for training. The whole dataset consists of 10 months of operational data collected on Marconi100. The first eight months of the data were used as a training set and the *last two* as a test set. Autoencoder models were trained on the train set. The cluster analysis was performed only on the test set.

The dataset used in this work consists of a combination of information recorded by Nagios (the system administrators tool used to visually check the health status of the computing nodes) and the Examon monitoring systems; the data encompasses the first ten months of operation of the M100 system. The features collected in the dataset are listed in table 1. In order to align different sampling rates of different reporting services, 15 minute aggregates of data points were created. 15 minute interval was chosen as it is the native sampling frequency of the Nagios monitoring service (where our labels come from). Four values were calculated for each 15 minute period and each feature: minimum, maximum, average, and variance.

Source	Features
Hardware monitoring	ambient temp., dimm[0-15] temp., fan[0-7] speed, fan disk power, GPU[0-3] core temp. , GPU[0-3] mem temp. , gv100card[0-3], core[0-3] temp. , p[0-1] io power, p[0-1] mem power, p[0-1] power, p[0-1] vdd temp. , part max used, ps[0-1] input power, ps[0-1] input voltage, ps[0-1] output current, ps[0-1] output voltage, total power
System monitoring	CPU system, bytes out, CPU idle, proc. run, mem. total, pkts. out, bytes in, boot time, CPU steal, mem. cached, stamp, CPU speed, mem. free, CPU num., swap total, CPU user, proc. total, pkts. in, mem. buffers, CPU idle, CPU nice, mem. shared, PCIe, CPU wio, swap free

Table 1: An anomaly detection model is created only on hardware and application monitoring features. More granular information regarding individual jobs is not collected to ensure the privacy of the HPC system users.

Features extracted from trained autoencoders are passed to hierarchical clustering. Hierarchical clustering has been chosen as it only requires the pairwise distance between the instances without making any assumptions about the space induced by the distance measure. The number of clusters is set to 20 for all experiments. A number of clusters is not a tuned parameter; 20 clusters represents roughly 10% of all nodes and is a randomly chosen number.

4.2 Trained autoencoder

The trained autoencoder is adopted from the current state-of-the-art semi-supervised approach for anomaly detection [19]. The structure of the autoencoder is presented in Figure 3. The autoencoder used as a binary classifier (form the normalized reconstruction error) on the test set achieves the AUC (area under the receiver-operator characteristic curve) of 0.7602.

Normal operation (the data where the autoencoder is trained) is determined by the label (system availability) provided by the monitoring systems.

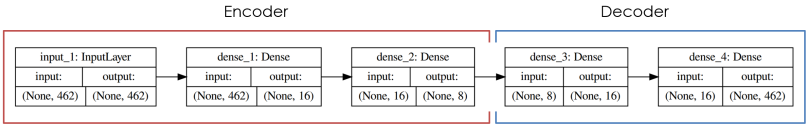

Fig. 3: Architecture of the autoencoder network, adopted from Borghesi et al. [19]

4.3 Cluster analysis: normal operation percentage

The proposed approach aims to identify interesting clusters of nodes that behave similarly. The similarity in behavior is also reflected in the fact that a cluster will have similar values for at least one relevant feature. In this section, we evaluate the similarity in average availability rate - in other words, we are interested in seeing if the clustering methods can identify clusters with particularly low availability (high failure rate). The average failure rate amongst 241 identified nodes (in the test set) is 0.96179. We wish to identify clusters with significantly lower availability rate.

In Table 2, the minimum average availability rates in a cluster, unidentified by a specific feature extraction approach, are reported. The table shows that the vector of singular values combined by the euclidean distance metric identifies a cluster with the minimum average availability. This availability is also lower than the random method's minimum availability (ever achieved).

In Figure 4 and Figure 5 average availability per node is plotted (red dots). Results of random sampling without replacement are presented as a box plot.

Distance measure:	Avg ava. in min. cluster:	Num. of nodes in min. cluster:
Sing. vector (Euc.)	0.9286	6
Vector of sing. values (Euc.)	0.8809	7
$W\vec{1} + \vec{b}$ (Euc.)	0.9126	8
$W\vec{1}$ (Euc.)	0.9367	5
W (absolute L2)	0.9191	7
A (absolute L2)	0.9276	5
W (L2)	0.9239	7
A (L2)	0.9124	10
W (L1)	0.9303	8
A (L1)	0.9303	8
Random sampling	0.9021	Not applicable

Table 2: Minimum average availability within clusters identified by different feature extraction methods. Vector of singular values identifies a cluster with the lowest average availability (highest anomaly rate). This is the most interesting method as it separates the target variable (node availability) the best. None of the proposed methods identify a cluster with a single node.

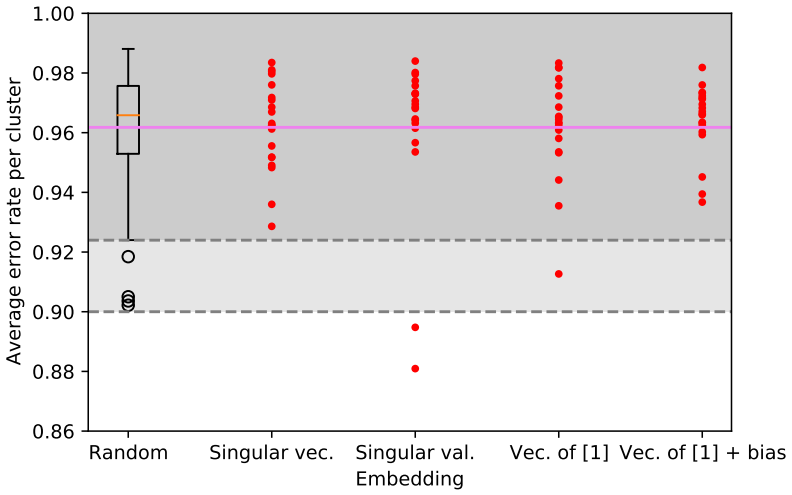

Fig. 4: Average error rate per cluster. Representation of nodes with a vector of singular values identifies two clusters with significantly higher anomaly rate than the whole population.

The average error rate across all nodes (0.96179) is marked with a violet dotted

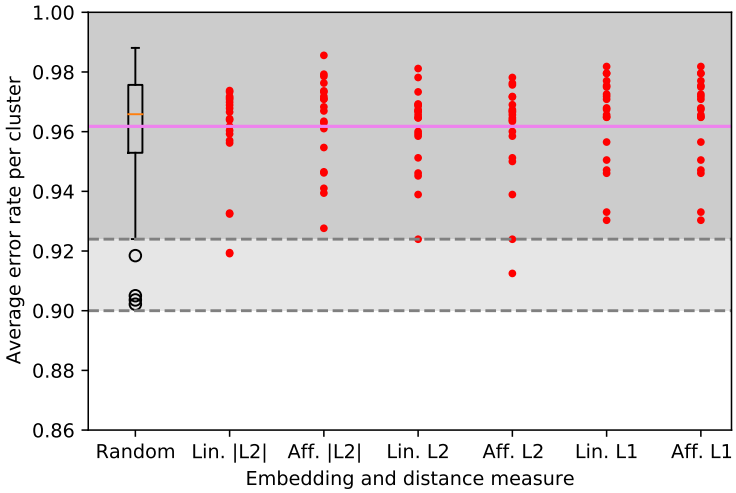

Fig. 5: Average error rate per cluster. Matrix-based feature extraction performs worse than the vector methods.

line. Area of values, observed in a random process, are marked with gray. Values *never* observed by the random process are left white.

Analyzing Figures 4 and 5 we observe that only the vector of singular values produced cluster with averages never observed in random samples.

The clustering method based on a vector of singular values combined with euclidean distance identifies two clusters with particularly low average availability. Such low average availability has also never occurred in a random selection of clusters. Low average availability means that hierarchical clustering based on singular value decomposition of weights matrix produces non-trivial clusters that are extremely unlikely to be matched by a random selection of clusters.

Identifying interesting clusters regarding availability is a non-trivial result as a neural network has no access to that label during training.

This promising result suggests that the created clusters share similar availability, and thus clusters can be created based on autoencoder semi-supervised models latent layer information. This cluster can then be used during the system's lifetime to create canaries to focus the maintenance over nodes belonging to the same cluster of the canary node.

5 Conclusions

This work opens the possibility of extracting additional information from the state-of-the-art approach towards anomaly detection in the HPC setting. Besides

using per-node autoencoder models for anomaly detection [19, 17, 18], it is also possible to construct informative clusters from the parameters of the trained neural networks themselves.

We demonstrate the usefulness of the identified clusters on a concrete example: identifying clusters with the abnormal failure rate. This result is significant as the neural networks, from where the features are extracted, have no *access to that label during training*. Still, our approach can identify two clusters of nodes with lower availability (higher failure rate) than the population average.

We stress the fact that with this approach, clusters can be created based on a model trained on the first month of operations and then applied for the remaining lifetime of the system to focus maintenance to the nodes belonging to the same cluster containing the node which has experienced failures. System administrators focus their regular inspections only on canary nodes, each representative of one cluster.

6 Acknowledgments

This research was partly supported by the EuroHPC EU PILOT project (g.a. 101034126), the EuroHPC EU Regale project (g.a. 956560), EU H2020-ICT-11-2018-2019 IoTwins project (g.a. 857191), and EU Pilot for exascale EuroHPC EUPEX (g. a. 101033975). We also thank CINECA for the collaboration and access to their machines and Francesco Beneventi for maintaining Examon.

References

1. M. Garcia-Gasulla, B. J. Wylie, Performance optimisation and productivity for eu hpc centres of excellence (and european parallel application developers preparing for exascale): Best practice for efficient and scalable application performance, in: Platform for Advanced Scientific Computing (PASC) Conference, no. FZJ-2022-00887, Jülich Supercomputing Center, 2021.
2. P. Kogge, D. R. Resnick, Yearly update: exascale projections for 2013., 2013. doi:10.2172/1104707.
3. T. C. Germann, Co-design in the exascale computing project (2021).
4. J. Gao, F. Zheng, F. Qi, Y. Ding, H. Li, H. Lu, W. He, H. Wei, L. Jin, X. Liu, et al., Sunway supercomputer architecture towards exascale computing: analysis and practice, Science China Information Sciences 64 (4) (2021) 1–21.
5. O. Terzo, J. Martinovič, HPC, Big Data, and AI Convergence Towards Exascale: Challenge and Vision, CRC Press, 2022.
6. X. Yang, Z. Wang, J. Xue, Y. Zhou, The reliability wall for exascale supercomputing, IEEE Transactions on Computers 61 (6) (2012) 767–779.
7. K. S. Stefanov, S. Pawar, A. Ranjan, S. Wandhekar, V. V. Voevodin, A review of supercomputer performance monitoring systems, Supercomputing Frontiers and Innovations 8 (3) (2021) 62–81.
8. M. A. Jette, A. B. Yoo, M. Grondona, Slurm: Simple linux utility for resource management, in: In LCNS: Proceedings of JSSPP, Springer-Verlag, 2002.
9. A. P. Works, Pbs professional®14.2 plugins (hooks) guide, https://pbsworks.com/pdfs/PBSHooks14.2.pdf (2017).

10. T. Dang, N. Nguyen, Y. Chen, Hiperview: real-time monitoring of dynamic behaviors of high-performance computing centers, The Journal of Supercomputing 77 (10) (2021) 11807–11826.
11. B. Yang, X. Ji, X. Ma, X. Wang, T. Zhang, X. Zhu, N. El-Sayed, H. Lan, Y. Yang, J. Zhai, et al., End-to-end {I/O} monitoring on a leading supercomputer, in: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), 2019, pp. 379–394.
12. A. Bartolini, F. e. a. Beneventi, Paving the way toward energy-aware and automated datacentre, in: Proceedings of the 48th International Conference on Parallel Processing: Workshops, 2019, pp. 1–8.
13. N. Wu, Y. Xie, A survey of machine learning for computer architecture and systems, ACM Computing Surveys (CSUR) 55 (3) (2022) 1–39.
14. O. Tuncer, E. Ates, Y. e. a. et Zhang, Online diagnosis of performance variation in hpc systems using machine learning, IEEE Transactions on Parallel and Distributed Systems (9 2018).
15. A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, A. Borghesi, A machine learning approach to online fault classification in hpc systems, Future Generation Computer Systems (2019).
16. A. Bose, H. Yang, W. H. Hsu, D. Andresen, Hpcgcn: A predictive framework on high performance computing cluster log data using graph convolutional networks, in: 2021 IEEE International Conference on Big Data (Big Data), IEEE, 2021, pp. 4113–4118.
17. A. Borghesi, A. Bartolini, et al., Anomaly detection using autoencoders in hpc systems, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019.
18. A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems, Engineering Applications of Artificial Intelligence 85 (2019) 634–644.
19. A. Borghesi, M. Molan, M. Milano, A. Bartolini, Anomaly detection and anticipation in high performance computing systems, IEEE Transactions on Parallel and Distributed Systems 33 (4) (2022) 739–750. doi:10.1109/TPDS.2021.3082802.
20. A. Netti, W. Shin, M. Ott, T. Wilde, N. Bates, A conceptual framework for hpc operational data analytics, in: 2021 IEEE International Conference on Cluster Computing (CLUSTER), 2021, pp. 596–603. doi:10.1109/Cluster48925.2021.00086.
21. B. Aksar, Y. Zhang, E. Ates, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems, in: International Conference on High Performance Computing, Springer, 2021, pp. 195–214.
22. B. Aksar, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, E2ewatch: An end-to-end anomaly diagnosis framework for production hpc systems, in: European Conference on Parallel Processing, Springer, 2021, pp. 70–85.
23. D. Bank, N. Koenigstein, R. Giryes, Autoencoders, CoRR abs/2003.05991 (2020). arXiv:2003.05991.
URL https://arxiv.org/abs/2003.05991
24. C. Song, F. Liu, Y. Huang, L. Wang, T. Tan, Auto-encoder based data clustering, in: J. Ruiz-Shulcloper, G. Sanniti di Baja (Eds.), Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 117–124.
25. X. Li, Z. Chen, L. K. Poon, N. L. Zhang, Learning latent superstructures in variational autoencoders for deep multidimensional clustering, arXiv preprint arXiv:1803.05206 (2018).

26. W. Wang, D. Yang, F. Chen, Y. Pang, S. Huang, Y. Ge, Clustering with orthogonal autoencoder, IEEE Access 7 (2019) 62421–62432.
27. A. Alam, M. Muqeem, S. Ahmad, Comprehensive review on clustering techniques and its application on high dimensional data, International Journal of Computer Science & Network Security 21 (6) (2021) 237–244.
28. B. Davis, D. McDonald, An elementary proof of the local central limit theorem, Journal of Theoretical Probability 8 (3) (1995) 693–702.
29. L. L. Cam, Maximum likelihood: An introduction, International Statistical Review / Revue Internationale de Statistique 58 (2) (1990) 153–171.
URL http://www.jstor.org/stable/1403464
30. A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems, Engineering Applications of Artificial Intelligence 85 (2019) 634–644. doi:https://doi.org/10.1016/j.engappai.2019.07.008.
URL https://www.sciencedirect.com/science/article/pii/S0952197619301721
31. A. Bartolini, F. Beneventi, A. Borghesi, D. Cesarini, A. Libri, L. Benini, C. Cavazzoni, Paving the way toward energy-aware and automated datacentre, in: Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019, Association for Computing Machinery, New York, NY, USA, 2019. doi:10.1145/3339186.3339215.
URL https://doi.org/10.1145/3339186.3339215
32. M. E. Wall, A. Rechtsteiner, L. M. Rocha, Singular value decomposition and principal component analysis, in: A practical approach to microarray data analysis, Springer, 2003, pp. 91–109.
33. G. Belitskii, et al., Matrix norms and their applications, Vol. 36, Birkhäuser, 2013.
34. F. Murtagh, P. Contreras, Algorithms for hierarchical clustering: an overview, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2 (1) (2012) 86–97.